Yet another version control tutorial

This won't turn you into a Git sorceress or wizard but hopefully you will be less confused after it.

Contributors:

- Rémi Gau
- Sofie Van Den Bossche
- 22 January 2020 @OpenMR Benelux 2020 <3 - typo corrections by:
    - Zsuzsika Sjoerds
    - Stijn Denissen
- Add your name if you spot any typo and make improvements! :D
- ...

Goals

This short workshop will not turn you into Git and GitHub wizards but it should hopefully:

1) make the terminology less scary so that you won't be as confused when you try it on your own;
2) guide you on how to version control some simple code;
3) guide you on how to use some of GitHub's functions by quickly creating your own academic website (e.g. https://academicpages.github.io/).

What to install?

- Download the file editor Atom (https://atom.io/) if you don't already have a generic decent editor on your computer like emac, vim, sublime…
- Create a GitHub account if you don't already have one (https://github.com).
- Add your name and GitHub username to this file.
- Install GitKraken (https://www.gitkraken.com/**)** and login into GitKraken using your GitHub account (setup 1, see this cheat sheet for help)

If something is unclear...

- Something does not work. Ask a question: we are here to help!
- Some term is too confusing: we have added a glossary at the end of this Google Doc. If the word is not in there, let us know about it during the workshop and we will add a definition so it can help others.
- Feel free to add a comment to the relevant section and assign it to @sofie.vandenbossche93@gmail.com or @remi.gau@gmail.com.
- You can also post comments and suggestions on Slack!
- GitKraken cheat sheet
- GitKraken help

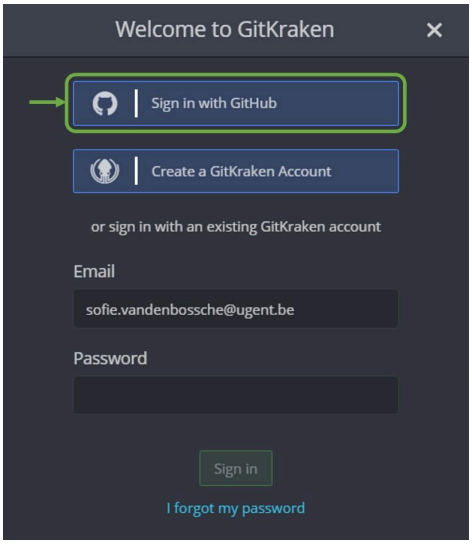
How to Git?

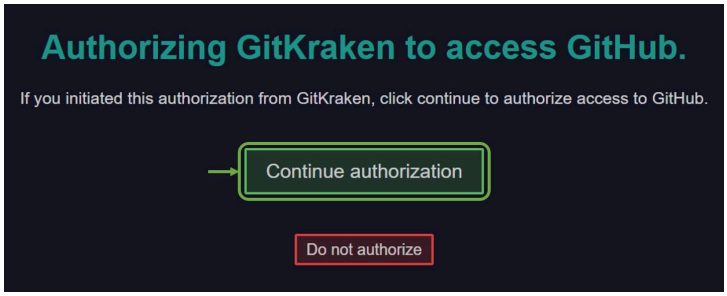
In this first section, we will see how you can use Git to keep track of changes you have been making to your code (i.e. version control) on your **LOCAL** machine (e.g. laptop). To this end, we will use GitKraken, a Graphical User Interface for Git (which can be easily downloaded for free on Windows, Mac or Linux).

For those who might have any prior experience using Git in the terminal, *matching terminal commands* will be stated as well. In the second section "How to GitHub?", we will see how you can back up the code you've written with GitKraken on your **LOCAL** machine to GitHub, a **REMOTE** platform in the cloud for which you need an internet connection. Then, we will see how you can use GitHub to collaborate with others on the same codebase.

Using Git(Kraken) to keep track of your code history

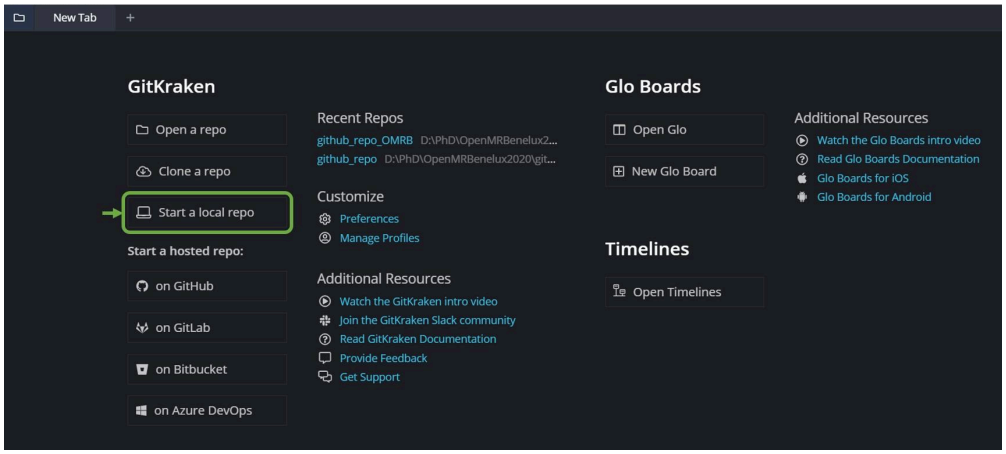
Setup: Open GitKraken and log in with your GitHub account

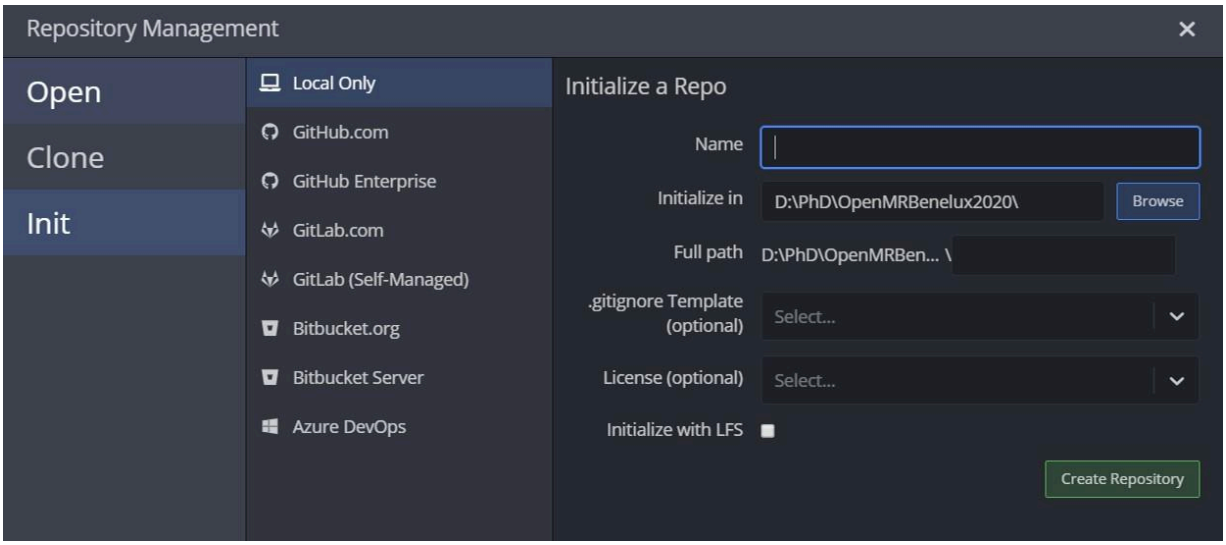




- After downloading GitKraken, open the GUI and sign in with your GitHub account by clicking on "Sign in with GitHub". If you have not created a GitHub account yet, please do so first. As a student, you can apply for the GitHub Student Developer Pack, which includes offers and benefits from GitHub partners.
- Thereafter, a new tab will be opened on your browser, where you will be asked to authorize GitKraken access to GitHub. Click on "Continue authorization".

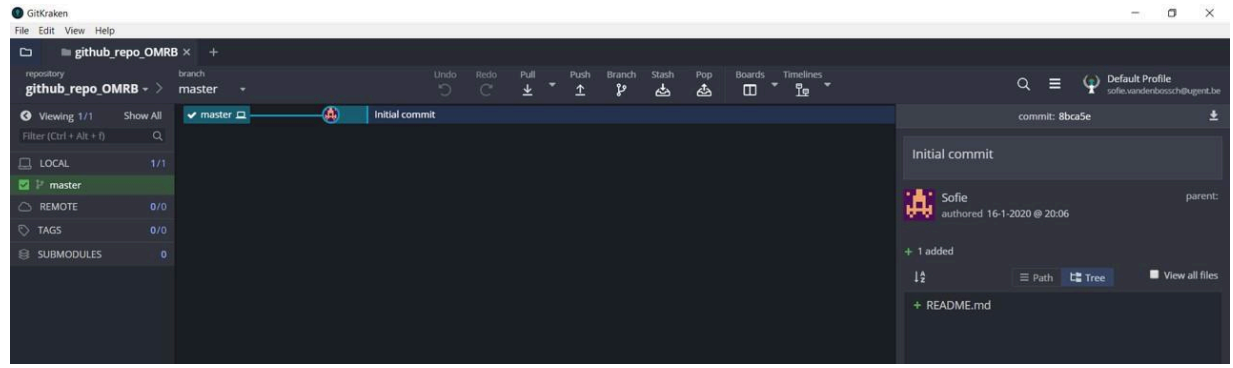
Create a new local repository [*git init*]





- Create a new (local) project folder, i.e. a Git repo(sitory) that will be tracked by Git and contain all your project files (command: *git init*) by clicking on “Start a local repo” (or go to ***File*** → ***Init Repo***.
- INPUT: fill out all the non-optional fields:
    - ***name***: name of the repository (e.g. github_repo_OMRB)
    - ***initialize in***: path to the directory where you want to save the repository

  The optional fields (i.e. .gitignore Template, License, LFS) fall outside of the scope of the current tutorial and thus will be left empty. After all the non-optional fields are filled in, click on “Create Repository”.



OUTPUT: A new initialized Git project at the specified repository path containing a blank README.md file (see: bottom right corner of the above figure). You can also (obviously) browse this newly created folder with your file explorer.

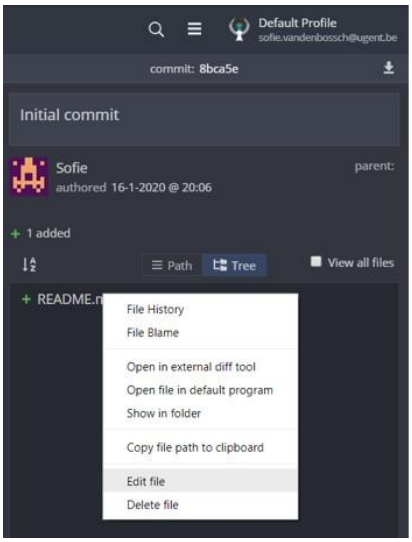
Note 1: You can initialize a repository in a folder that already contains some file(s).
Note 2: Only the files that are in the folder where you have initialized the repository will be tracked by git.

Extra: hidden .git folder

Add a description to your project

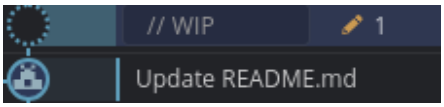
In order to create clear project documentation, a default README.md file is added; written using Markdown (a simple markup language). First, we will add a project description:

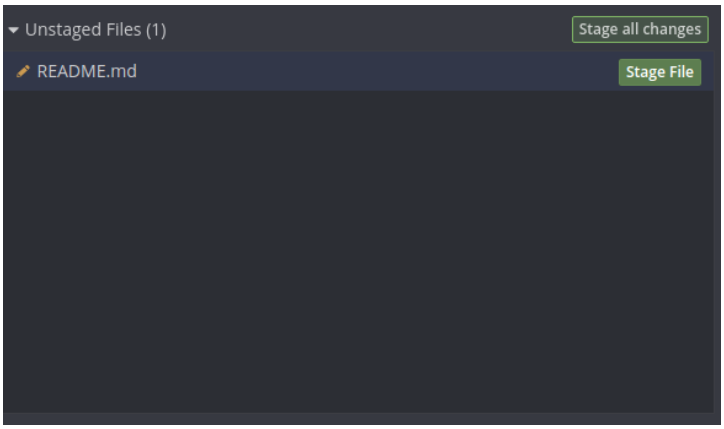
1. Open the file using Atom or your favorite text editor.
2. Add some content to the file. It could be:
   - # Project description
   - a paragraph of filler text using lorem ipsum.



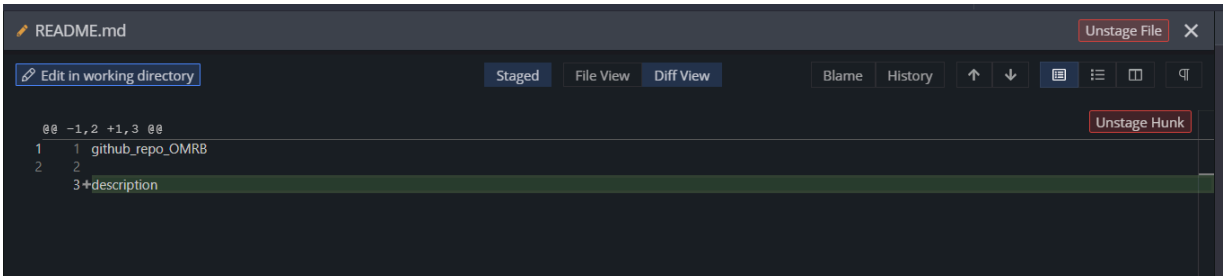
Extra: Note you can also modify the file in GitKraken by right-clicking the README.md file and selecting "Edit file". The file will be opened into edit mode. The "**editable**" tag in the upper left corner, denotes that you can edit the current file. The blue dot in the upper-right corner indicates unsaved changes. In order to save them, hit *Ctrl/Cmd + S*.

Staging the changes [*git add*]





- You have made some changes to a file. GitKraken will let you know by showing a new line at the top of the central window with "**//WIP**" (Work In Progress) followed by a pen and a number. The number indicates how many files have been changed.
- The right hand column of GitKraken lists the files that have been changed in the box "**Unstaged files**".



You can click on each of those files to open the "**Diff view**" that will show you the lines in this file that have been changed

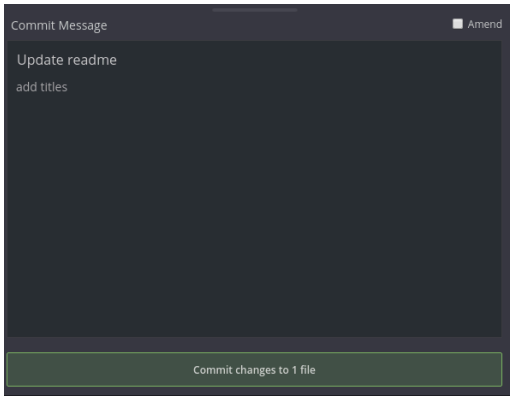
Now we want to tell Git to keep track of this file. The first thing to do is to "**stage**" the file before we tell Git to take a snapshot of it.

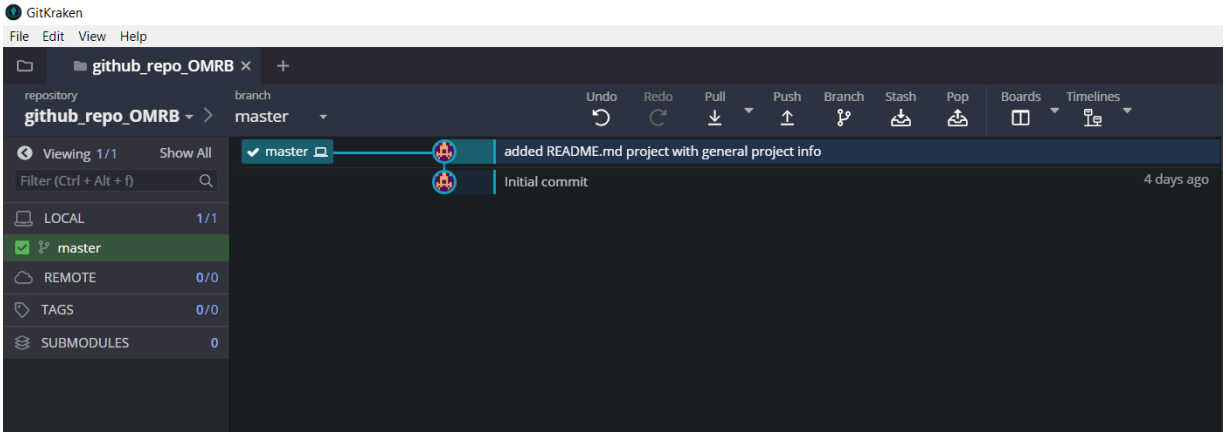
- Stage the file by right-clicking on it and selecting "**Stage**" or by clicking the "**Stage file**" button that appears when you hover on the right of it.
- If you want to stage all of the files that have been changed you can simply click on "**Stage all changes**". This will move the files in the box "**Staged files**" below.

Extra: if you are using the command line, the command *git status* would give information about what file has been changed, added, deleted, or staged. The command *git diff* would give you something a bit like the "**Diff view"** telling about which specific lines in a file have been changed.

Committing the changes [*git commit*]

Now we want to take a snapshot of our project that includes all the changes we have just staged. For Git that means we want to create what's called a "**commit**". A bit like a checkpoint in the history of your project.
Each commit has to have a commit message that is very short summary of what was added to the project in this commit. Once your project history grows, the list of commit messages will create a log that will help you figure out what happened when: so make sure the message header you write is as short, clear and unambiguous as possible.
You also have the possibility to make more extensive commit message with a header and a body, if you want to get into details or give reasons why some changes were made.





1. In GitKraken, once you have staged all the files you want to commit, write your commit message in the bottom right section of the main window in the "**Summary**" (for the message header) and "**Description**" (for a more lengthy version).
2. Click on the "**Commit changes to *?* files**"

Once you have done that you should see that a new commit has been at the top of your project history in the central window of GitKraken with the commit message next to it.

The holy "trinity": edit → stage → commit

A lot of using Git is basically going through cycles of:

1. editing some files
2. staging them
3. committing them

Feel free to make some more changes, add new files, and commit them to practice this. Also try to remove some things you added to see how they show up on the *Diff view***.** And if you remove a file you also have to commit the deleted file to tell Git about it.

Remember that GitKraken will keep track of changed, added, and deleted files by showing it at the top row of the central window.

If you create new files you will see "**+**" followed by the number of files created. If you delete new files you will see "**-**" followed by the number of files deleted. And if you want more details, you can always click on that top row to see the list of unstaged files.

Extra: *git log* would be the command to use in a terminal to get an overview of the commit history and also to search for specific commits by date, by submitter, by commit message...

Adding new code: Branching [*git branch*; *git checkout*]

Now we are going to develop our project by adding a new file to it. A good practice when you want to start developing something new in a project (e.g. additional code…) is to do this in what is called a "**branch**". It allows you to implement changes without affecting previously committed files.

Branches are like different versions of the same project that might start from the same point but can develop in very different directions. It is a bit like if at one point you created a copy of the folder containing your code and you started doing different things in the 2 folders. Working with branches in Git does the same but allows more flexibility and provides you with other advantages.

Until now you have been working on the default branch called the "**master**" branch: in a newly initialized repository, the only branch created is the master branch. You can see all the branches of your project listed under LOCAL in the left hand column of GitKraken.

1. Create a new branch by right-clicking on the master branch and selecting "Create branch here" [*git branch*].
2. A panel will open in the central window where you have to give a name for this new branch. Let's call it "Dev" for development.

You will have a new "Dev" branch listed under LOCAL. Next to it you will also see a tick showing on which branch you are. To reuse the analogy of the several directories above, the tick shows you in which directory you are.

In Git, when you change branches, you are "**checking out**" a branch: a bit like checking out a book in a library. In GitKraken you can checkout a branch by double clicking in the left column [*git checkout*].

1. Check out the Dev branch if it is not already the case.
2. Add a new file to your project. For example, add a "test.md" file using Atom or another text editor. You can also add an image, some MATLAB or Python code file… Remember that you need to add this file IN the folder where you initialized the repository for Git to see this file.
3. Stage and commit the file(s).
4. Now open a file explorer window (or finder or nautilus depending on the operating system your are using) and go to the directory where your project is.
5. Now checkout the master branch in GitKraken and see what happens to the file you have just created: the content of the folder changes depending on which branch is checked out
6. Checkout the Dev branch again and see if the file reappears.

Adding new code: Merging [*git merge*]

Now you have 2 different branches. Your Dev branch is ahead of the master branch because it has some commit that does not exist in the master branch. But how do you bring the changes made on your Dev branch into your master branch? That's when "merging" branches comes into play. Merging takes the commits on two different branches and combines them.

Once again, when coming back to our 2 folders example from above, it would be like taking the new content from one folder and dropping it into the other folder. We are going to merge the new changes from Dev into the master branch.

1. Checkout the master branch.
2. Right-click on the Dev branch and select "**merge Dev into master**"

This will create a new commit on master that represents where the merge has happened.

Extra: You might sometimes run into what is called a "**merge conflict**". This happens when you are trying to merge 2 branches where the same line in the same file has been modified. A bit like when you are copying the content of a folder into another folder and your operating system finds 2 files with the same name. In that case, it will ask you which one you would like to keep. A merge conflict is the same with Git, except that Git is smarter and will ask you which lines of the 2 different files you would like to keep.
GitKraken has a useful visual interface that can help you solve those problems. Let us know if this happens and we will walk you through it.

How to GitHub?

In this second section, we will first see how to use GitHub to back up and keep track of the changes you have been making to your code on a remote place in the cloud. Then we will see how you can use GitHub to collaborate with others on the same codebase.

Using GitHub to backup your code

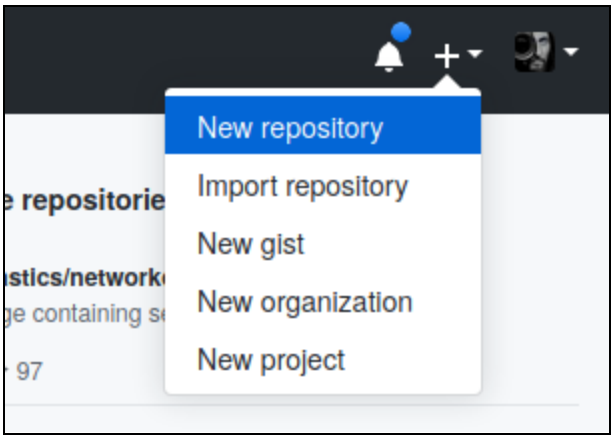
Put your files and your changes online

All the files and the history of the changes you made are for now on your local machine (e.g. laptop) only. They are stored on what is called a local repository.
We now want to put them online so that they are:

- backed up;
- can be easily accessed and/or modified by you from another computer or someone else.

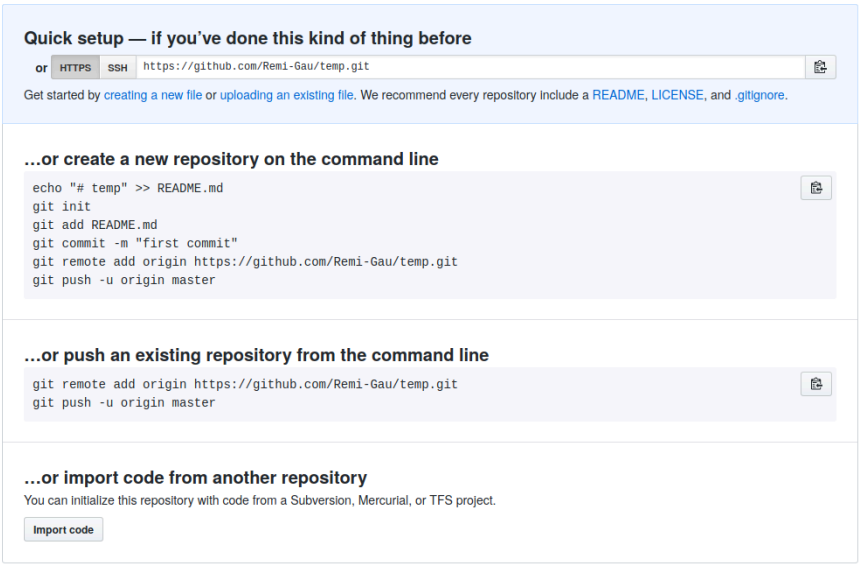
To do this, we need to first create a remote repository to host them.

1. Login onto your GitHub account and create a repository by clicking on the "**+**" sign at the top right corner of the screen:



2. On the next screen you should fill in the name of the repository:
   Keep it fairly short and self-explanatory: trust us, this will help when you start having many repositories. The name does not have to match with the name of the folder where your files are stored on your local machine.

3. Decide whether you want this remote repository to be public or private. Choose public. Public repositories can be viewed by anyone on the internet where as private repositories can only be accessed to the owner of the repository and the people who have been added as collaborators. Since 2019, you can have as many private repositories as you want without a special GitHub account (Pro or Student). However, GitKraken will not let you work on private repositories unless you have the Pro version of the software.
4. We want to create an empty repository so don't ask to create a README file (as we already have one in our local repository) or a .gitignore or license file.
5. Click on "***Create repository***"; you have now created the empty, remote repository and GitHub should open a help page giving you some instructions on what to do next.



In our case the next things we need to do are:

- To "link" this remote repository to your local one;
- Move all the files and their history into the remote repository.

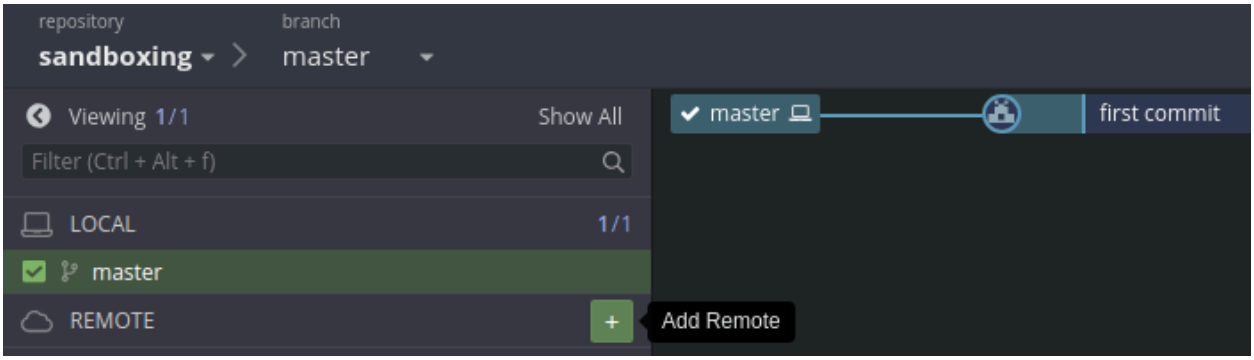
To do this we will need the URL of this remote repository that is displayed on the help page that appears on GitHub when you create an empty repository. This URL should have the following format: https://github.com/**your_username**/**repo_name**.git

If you are using the command line, this help page will actually also tell you which commands to type to do the next two steps. It is a very useful reminder.

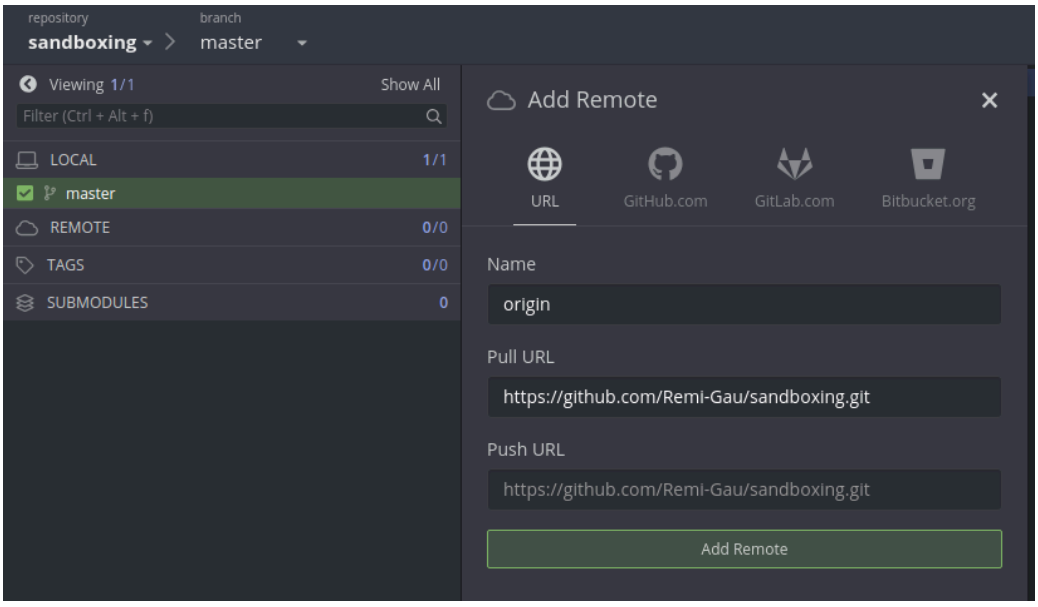
Linking the local to the remote repo [*git remote add*]

Before you can tell Git to save the content of your local repo online, your first need to tell it that a place exists where to save things. In Git lingo that means you need to add a remote to your local repository. How to do so?

1. To add a remote in GitKraken you need to click on the "**+**" sign that will appear when you hover over the "**Remote**" on the left-hand column.



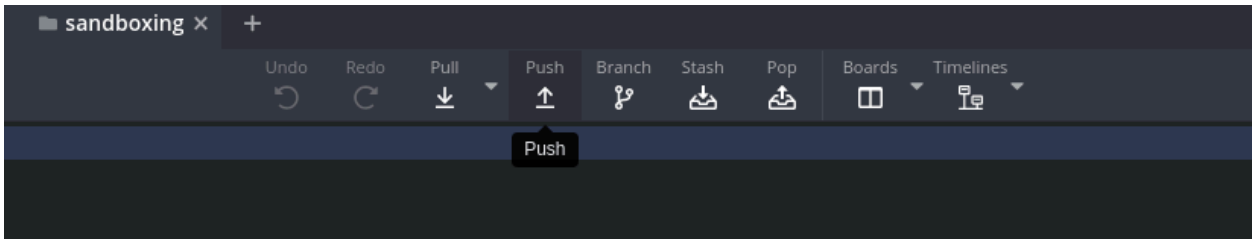
2. This will open the "**add remote**" panel where you can paste the URL in the "pull URL" box.
3. Give a name to the remote you are adding. A common name you will often find is "**origin**", but feel free to call it something that makes sense to you.
4. Click "**Add remote**". The new remote should appear in the left column.



Saving the state of repository [*git push*]

We now want to save the changes we did. In the Git world, we say that we want to push our local changes to the remote repository.

1. You should first choose the branch, you want to save. So checkout the master branch by double-clicking on it in the left-hand panel.
2. To push changes in GitKraken, you simply need to click on the Push button at the top of the window.



3. The first time when you click on it, GitKraken will ask you where you want to push those changes. It will ask for:
    a. the name of the remote;
    b. the name you want to give to the branch where you will push.

GitKraken will guess that you want to push your changes on the only remote you have ("**origin**") and will give the remote branch the same name as the branch you have currently checked out (this is actually a better practice unless you want to make your life extra complicated).
The remote branch will be added under the remote repository in the left-hand column.
You can now say that your local branch is tracking your remote branch.

4. Go back on GitHub and refresh the page of your repository. It should now show a content that mirrors the content of the master branch on your local repository.
5. You can now check out the other branches you have in your local repository and push them to your remote repository.
6. If you commit new changes to one of your local branches, you can then simply push your changes to your remote by clicking push.

GitKraken will show you where each branch is in the commit graph, so you can figure out if your local branch is ahead of your remote branch: meaning that you have local commits that you have not pushed on your remote. If this is the case, you will also see a number with an arrow pointing up in front of the branch name under the "**Local**" section: this tells you the number of commits you have not yet pushed.

Using GitHub to collaborate with others

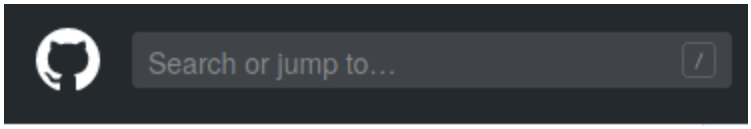
In this workflow, you want to use the codebase developed by someone else and possibly expand on it, improve or fix something. So you will copy someone's repository on your GitHub account and then on your computer. You will make some changes to the code, push those onto your remote repository and then ask the owner of the original repository to integrate your improvements into their codebase.

Making a copy of someone's repository: forking

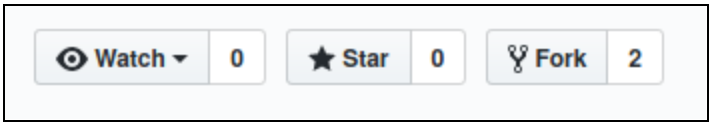
The first step is to create a copy on your GitHub account of the repository that you are interested in and that is currently located on someone else's account. This is called "forking" or creating a "fork" of the repository.

In this workshop, you can fork the repository created by your neighbour in the previous step. Otherwise you can look at the usernames of the people present at the workshop here and fork one of their repositories. How to proceed?

1. You can search users in GitHub by using the search bar at the top left of the screen.



2. Once you have found the user, you can select the repository you want to fork. We will call it the "**upstream**" repository.
3. You can then fork the upstream repository by clicking the "**Fork**" button at the top right.



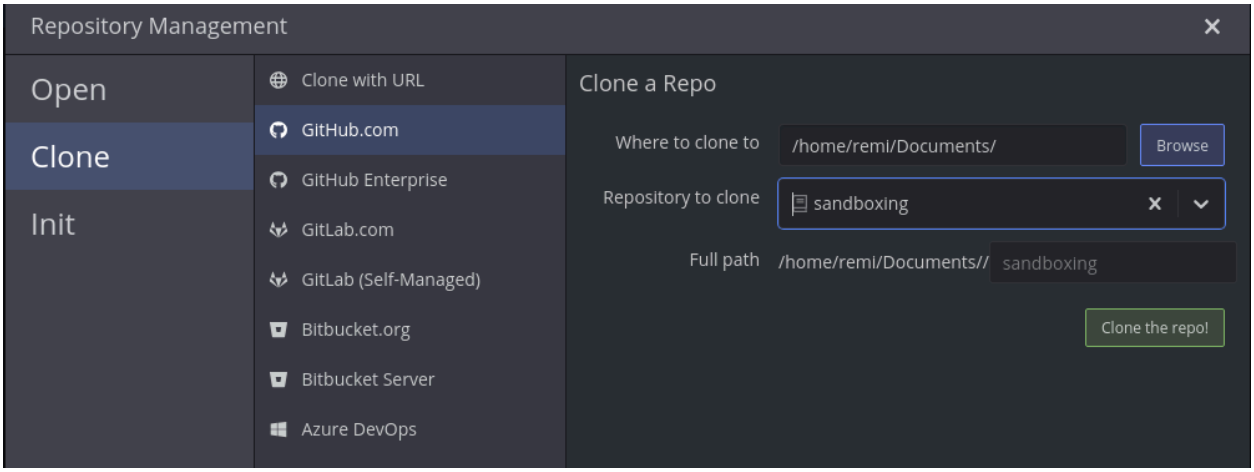
Once this is done you will find a copy of the upstream repository you forked on your account. This way you can make all the changes you want to this code without affecting the content upstream material.

Copying a repository from your account to your computer [*git clone*]

So now we have a remote repository on your GitHub account that does not have a local counterpart on your machine. So we need to make a local copy: in the Git world this is called "**cloning**".

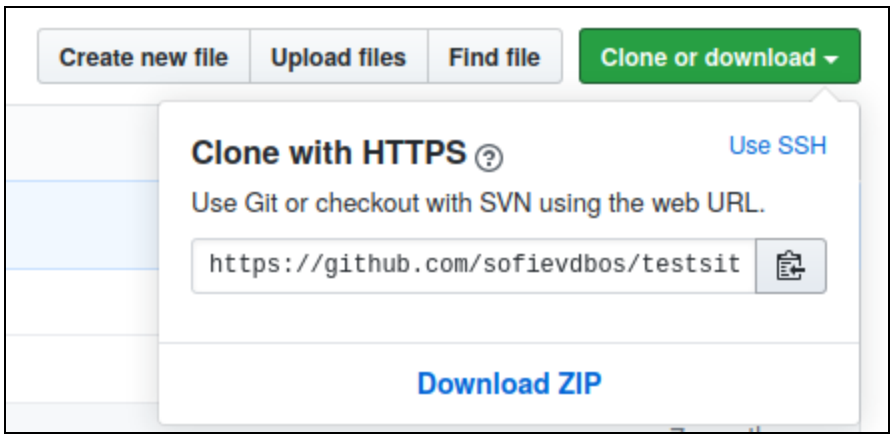
1. To clone a repo with GitKraken go to ***File*** → ***Clone Repo***.

2. Then in the next window, you can select the "**GitHub.com**" panel.
3. You then need to mention where you want to clone the repository on your machine.
4. Finally you need to say which remote repository you want to clone.
5. Once you are done, click "**Clone the repo!**"

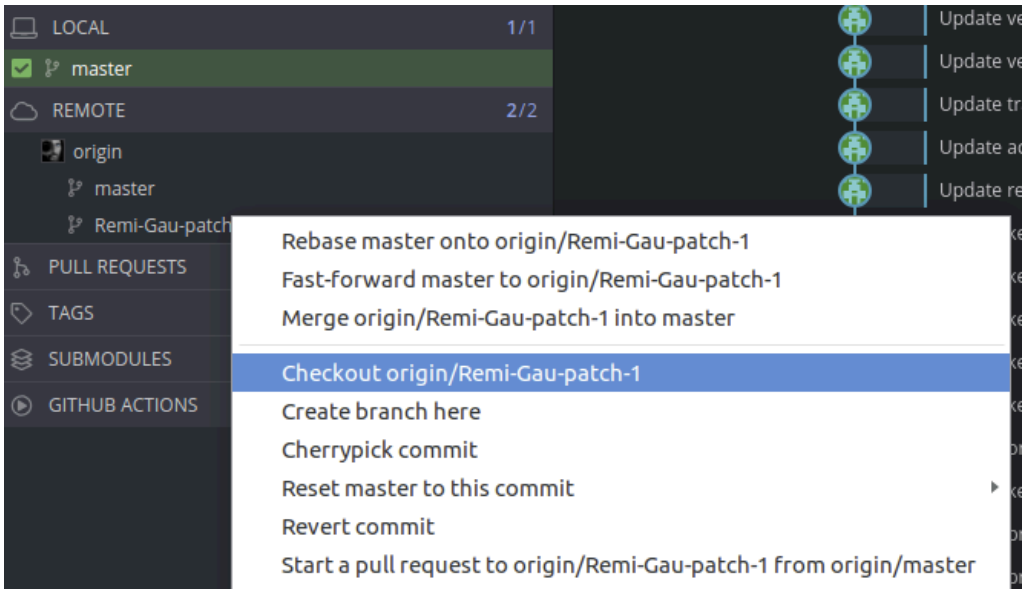


This will create a repository in the folder you chose for "**Where to clone to**" with the same name as the remote repo.

If you are using the command line or in case you need the URL of a repository to clone it, you can find it by clicking the "**Clone or download**" button on the page of the repo.



By default, only the master branch of the remote repository is cloned, but you can easily create a local branch to track a remote branch by right clicking on the remote branch you are interested in the left-hand panel and then select "***checkout remote_name/branch_name"***.



Making some changes to the code

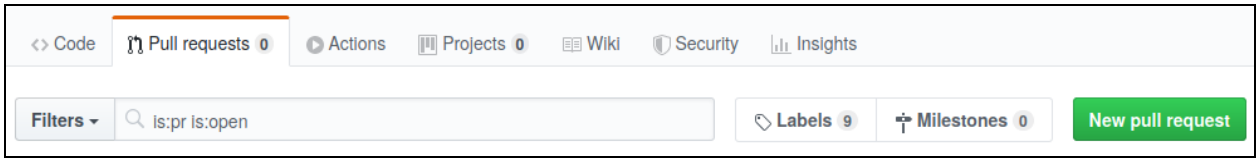
So you want to make some changes to the code. For example, you could simply add a line to the README file of the repository. So to do this, we will reuse what we have seen so far in this tutorial.

1. Create a branch where you will be making those changes;
2. Amend the README file: add whatever you feel like, as this is just for practice;
3. Stage and commit those changes;
4. Push the committed changes to your remote repository.

Ask the owner of the upstream repository to merge your changes into his/her codebase: pull request

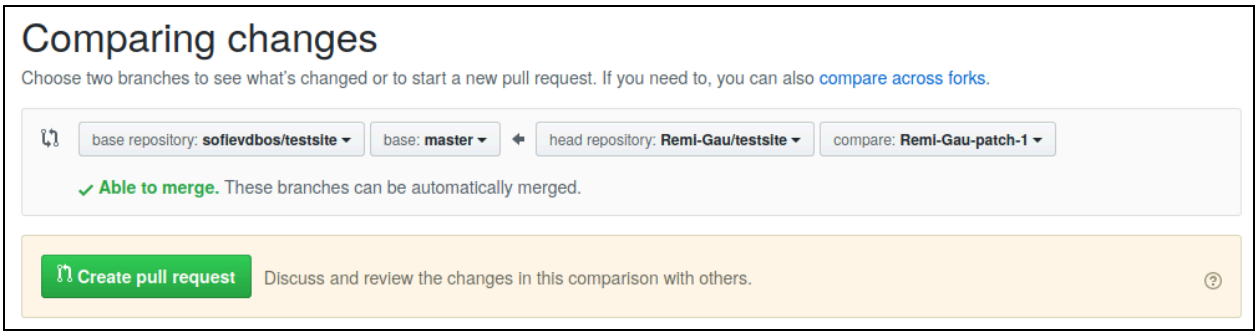
The next step is to get the changes from your GitHub repo to the upstream repo. You will then request the owner of the upstream to pull from your repo to his/hers.
In the Git world "**Pull**" is the opposite of "**Push**": the latter sends updates from repository A to repository B (as we were doing when we were pushing our commits from the local repo to the remote one), while the former gets updates from repository B to repository A.

1. To open a pull request go to your GitHub account on the repository where you have just pushed the latest commit.
2. Click on the "**Pull request**" tab and then on "**New pull request**".

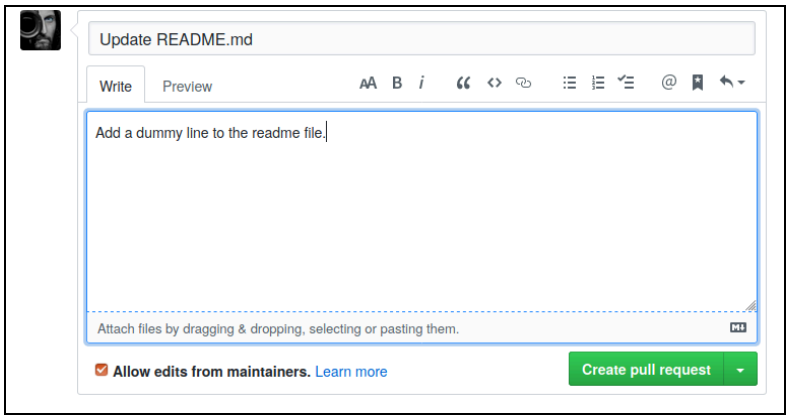


3. In the next window, you have to decide from which branch (called "**compare**" in the figure below) on your repository (called "**head repository**" in the figure below) should be merged into which branch (called "**base**" in the figure below) of the owner of the upstream repository (called "**base repository**" in the figure below).

   In most cases, you want to do a pull request on the default branch of the repository (that very often is the master branch), so GitHub will select that default branch but this might not always be the case.
   Similarly be careful which branch on your repository you select. But if you make a mistake don't worry, you can always close an erroneous pull request and create a new one.



4. When you are done, click on "**Create pull request**".
5. This will open a new section asking you to give a name to the pull request and also to describe the content of the pull request.



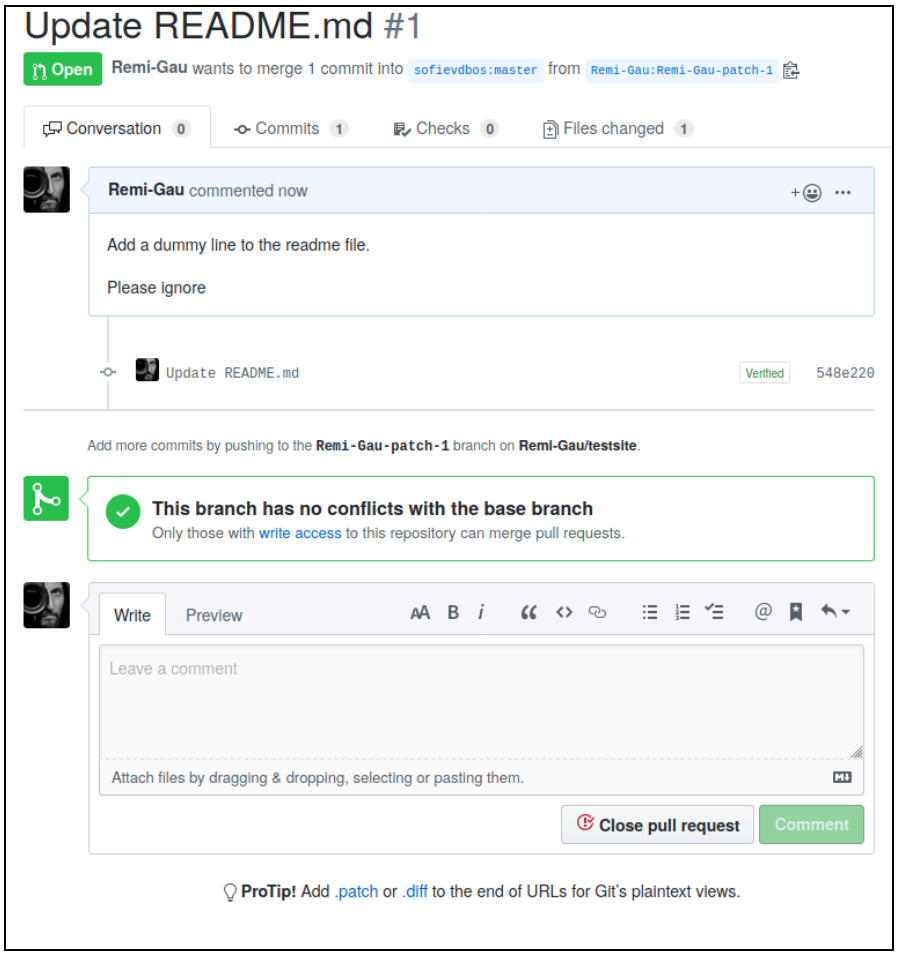
6. When you are done, click on “**Create pull request**”.

Reviewing and accepting the pull request

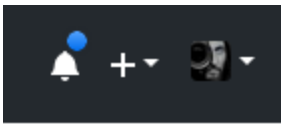
Once a pull request is created, there will be a page that presents several tabs. The main ones you should worry about for now are:

- The “**Conversation**” tab that shows all the commits that are contained in that pull request and that will be merged into the target branch, and provides a forum where the different persons involved in the project can discuss what is OK and what not with the pull request.
- The “**Files changed**” tab that will show all the changes made.

The “**Files changed**” tab becomes very useful to do a code review and make specific comments, suggestions, or ask for changes on the content of the pull request.

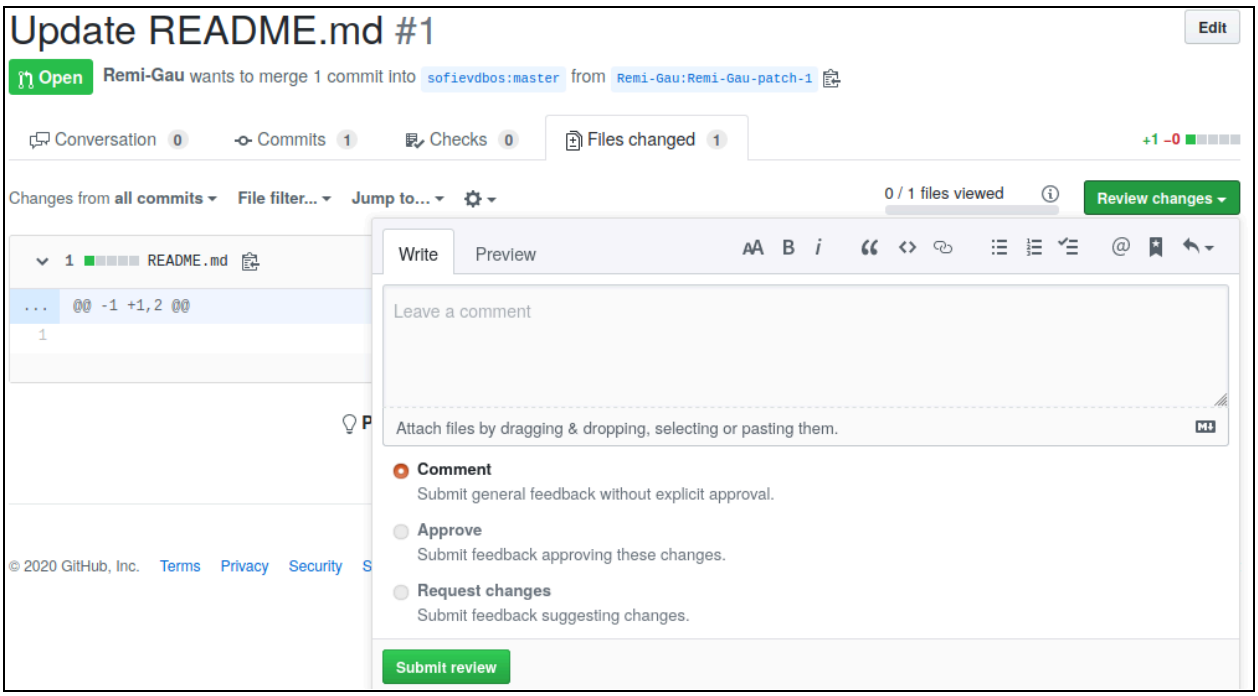


Hopefully while you were busy making a pull request on someone's repo, someone else was doing the same on yours. So you should get a GitHub notification about this. Check the bell symbol at the top-right of the GitHub page:



You can now review the pull request that was made on one of your repositories:

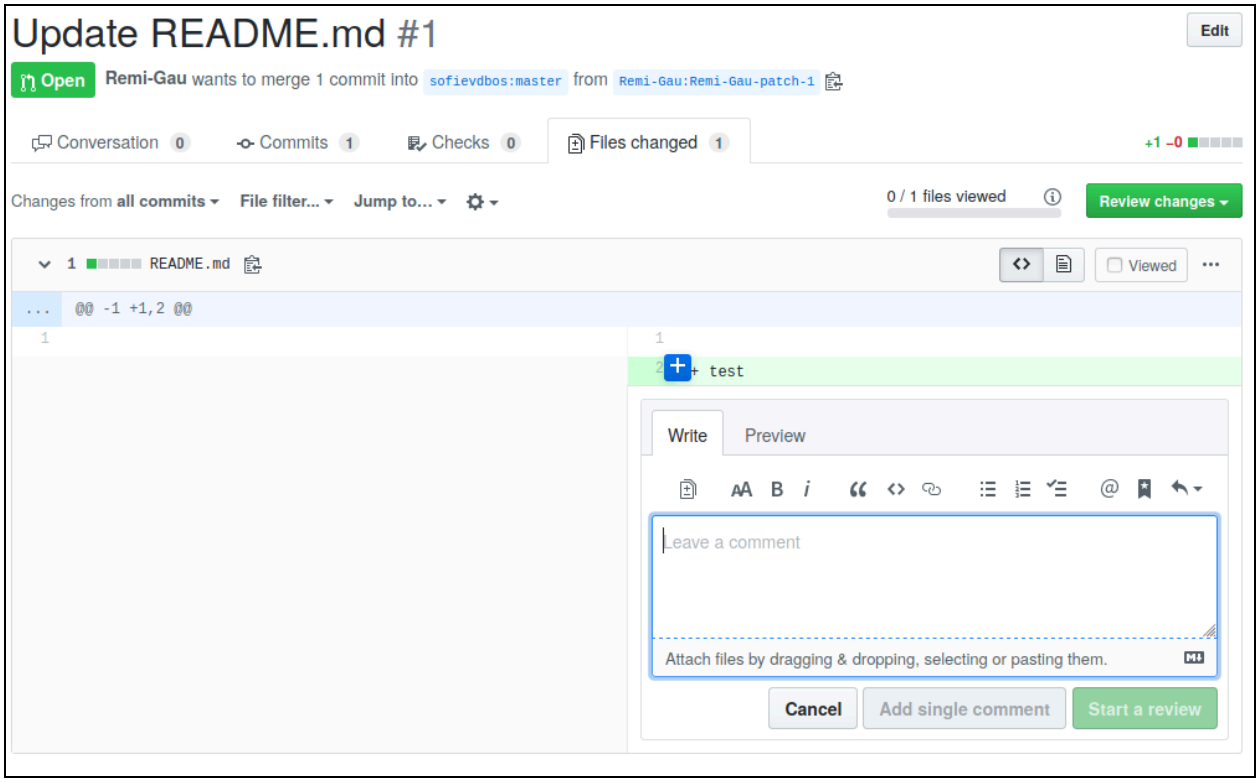
1. You can either make a general comment, approve or request changes.



   If you request changes, the person who made the pull request would simply have to:
   - Make the requested changes on the branch that he/she used to for this pull request;
   - Commit them on his/her local repository;
   - Then push them on his/her remote repository.

   The pull request will then be automatically updated.

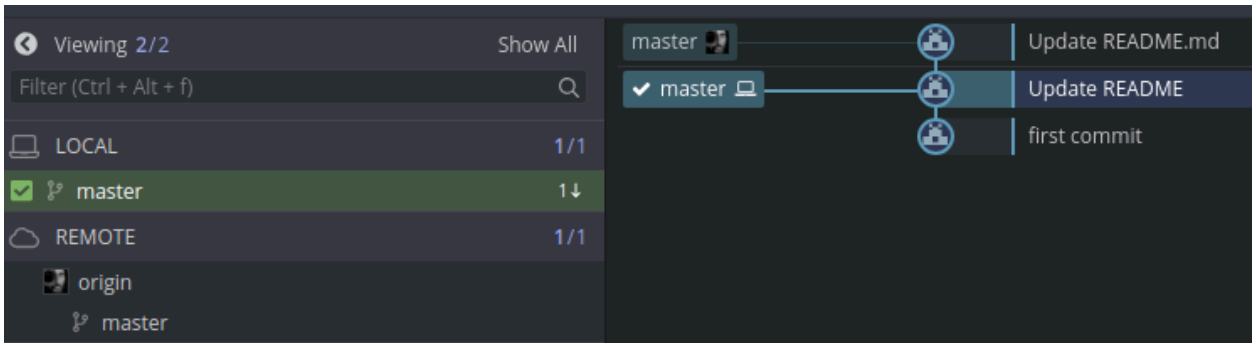
2. You can also make a specific comment by clicking on a certain line of a given file.



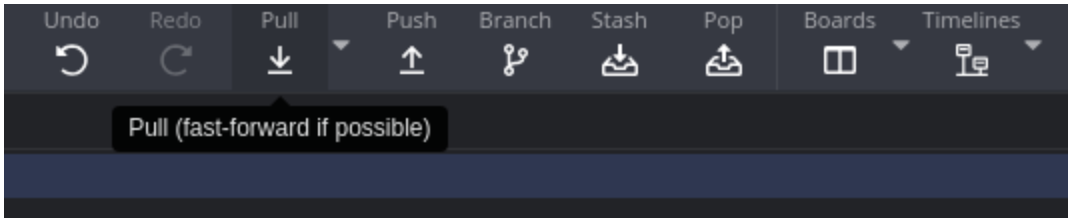
3. For now, simply approve the pull request.
4. Then go into the **Conversation** tab and click on “Merge pull request” to integrate all the suggested changes into your master branch.

Updating your local repo after the pull request [*git pull*]

You now have some new commits that exist on the master branch of your remote repo but not on your local repository. So your local branch is lagging behind the remote one. GitKraken will notify you with an arrow pointing down and a number in front of the master branch name in the left-hand column: this indicates how far behind (in number of commits) your local branch is.



To get the latest commits that are on your remote master branch, you simply need to check out the master branch (by double clicking on it) and to press the "**Pull**" button at the top of the GitKraken window.



Once this is done your will local branch repository will be up to date with your remote branch.

Using GitKraken to make a pull request and do a code review

You can also directly open a pull request from GitKraken, by right-clicking on the branch from which you want to make a pull request and then click on "**start a pull request to X from Y**".

If you actually want to review someone's pull request on your computer to nt only view the code but also to test it on your data for example, you can do that by adding this person remote repository to the list of remote just like you did before. Then you need to checkout the branch they send their pull request from by double clicking on it.

What now?

You have finished before the end of the workshop or you want to practice some more before starting applying those skills to your code or you want to get more technical. We have some suggestions:

Make a pull request on the repository for this workshop

You can help us improve the README file of this repository that contains a raw version of this Google Doc. You can sort of view suggestions for edits on a Google Doc as a sort of pull request but using a different system. So you can now, for example, make a pull request:

- to implement some comments or edits that you, or someone else, made to this Google Doc during the workshop;
- to train yourself at some markdown formatting.

Create your own website using GitHub pages

If you are starting and get confused ask us for help...
https://github.com/academicpages/academicpages.github.io

Familiarize yourself with the command line version of Git

If you are planning to get more comfortable with the command line version of git, we strongly recommend the version control chapter of the awesome Turing Way Handbook.

Glossary

Some of those were taken from the excellent turing way handbook project. :D

History	Gives you an overview of committed changes (e.g. to your code) over time (i.e. version control).
Initialize	Creating a new, empty Git(Kraken) repository (command: *git init*).
Local (repository)	Accessible through your local machine (e.g. laptop; for which you don't need an internet connection).
Merge	Merging takes the commits on two different branches and combines them. (reference: https://support.gitkraken.com/working-with-repositories/branching-and-merging/)
Push	Sending changes to a remote repo. The remote repository is updated with the changes pushed and now mirrors the local repo.
Remote (repository)	Accessible through the cloud (e.g. hosted on GitHub; for which you need an internet connection).
Repository	Refers to a project folder that is being tracked by Git and containing project files. Also called 'repo' for short, they can be local as well as hosted on GitHub (reference: https://the-turing-way.netlify.com/version_control/version_control.html)

FAQ

1. **Why can't I use GitKraken to work with private repositories?**
   - According to GitKraken's website, private repos are not available in the free tier of GitKraken. You need the Individual or Pro plan. https://superuser.com/questions/1500265/cant-use-gitkraken-on-private-repositories
   - **WORKAROUND FOR STUDENTS**: GitKraken Pro accounts are free for students through the GitHub Student Developer Pack. Apply for a GitHub Student Developer Pack by following the steps provided at: https://help.github.com/en/github/teaching-and-learning-with-github-education/applying-for-a-student-developer-pack

2. **Why can't I use GitKraken to work with GitHub Enterprise?**
   → Same as above (*I can't use GitKraken on private repositories?*).