Collaborative Coding

ALexader Perera Lluna/ Mónica Rojas Martínez
December 15th, 2023

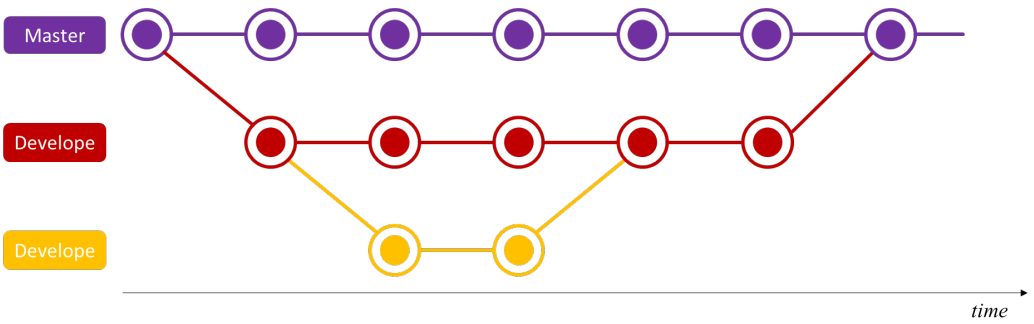
Contents

1. Introduction
2. Creating an account
3. Creating a collaborative story
4. Report

1. Introduction

GitHub is a cloud repository service that helps developers store, track, and manage changes to their code. It uses an open-source version control system called Git, which allows for easy branching and merging of different versions. In Git, each developer has a copy of the entire project that can be independently managed.

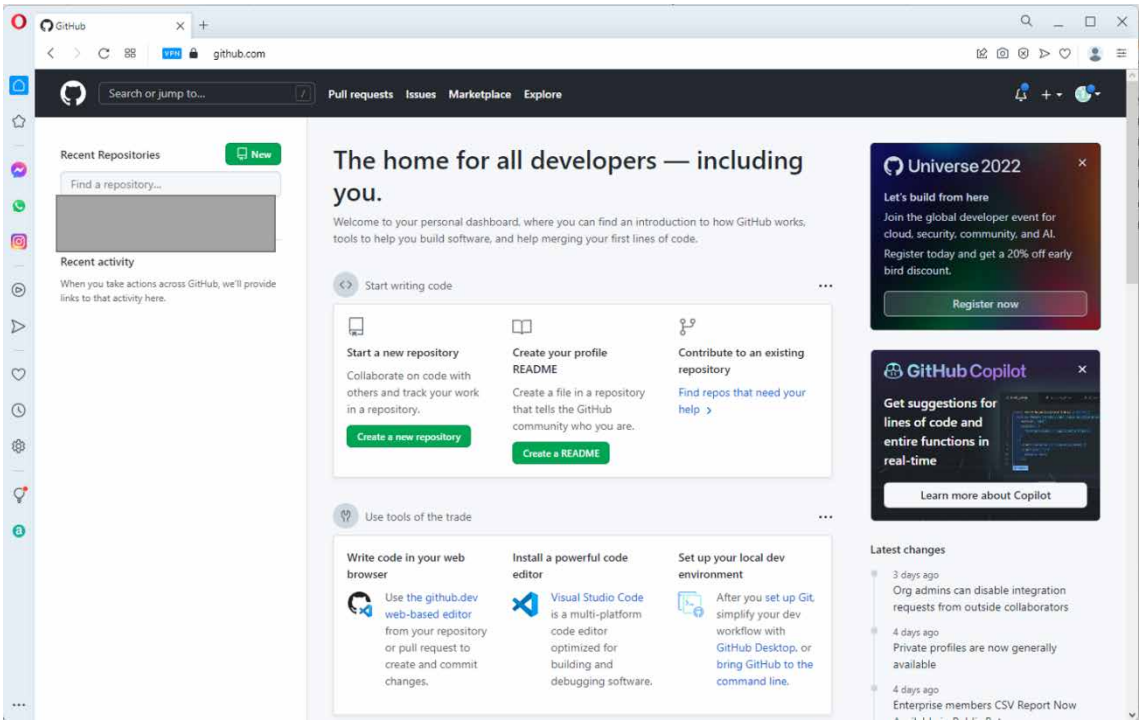
The project is divided into independent branches (figure 1) emerging from the main branch (or master, in purple). Each branch (red, yellow) is created by a developer as a copy of the project at a given time. In this way, the developer is able to add new features or fix bugs without changing the master project. After testing and approving the changes to the code, the "new branch" can then be merged to the master project and continue its track. Additional branches can also be created from secondary ones, and each branch can eventually become a new project by its own.



2. Creating and setting your account

- Go to https://github.com/
- Select “Sign Up” and follow the instructions

Once you sign-in you should be re-directed to your personal account like this:



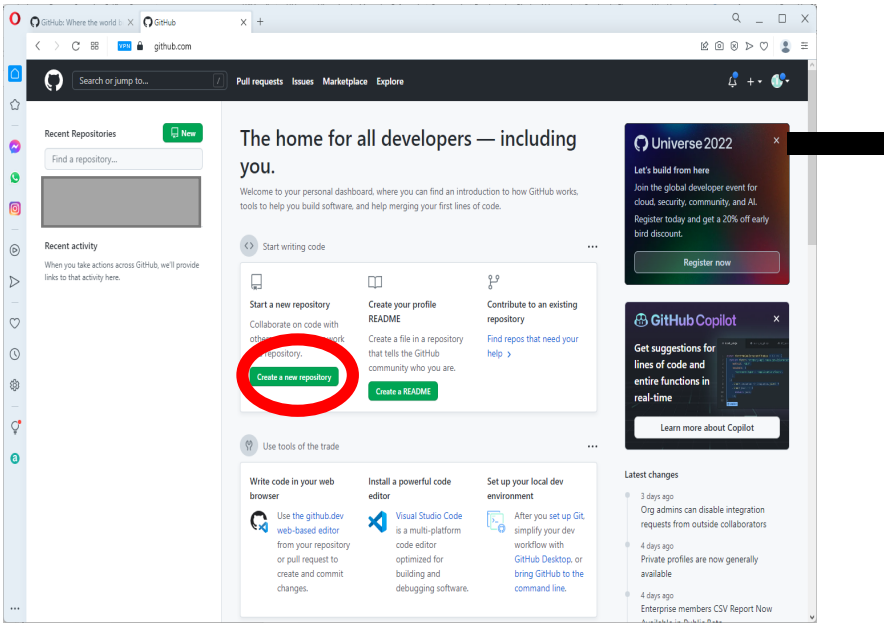
- Install the GitHub client following this link. It will automatically choose the version according to your OS.
- Log in from the client.

3. Example: *Collaborative story*

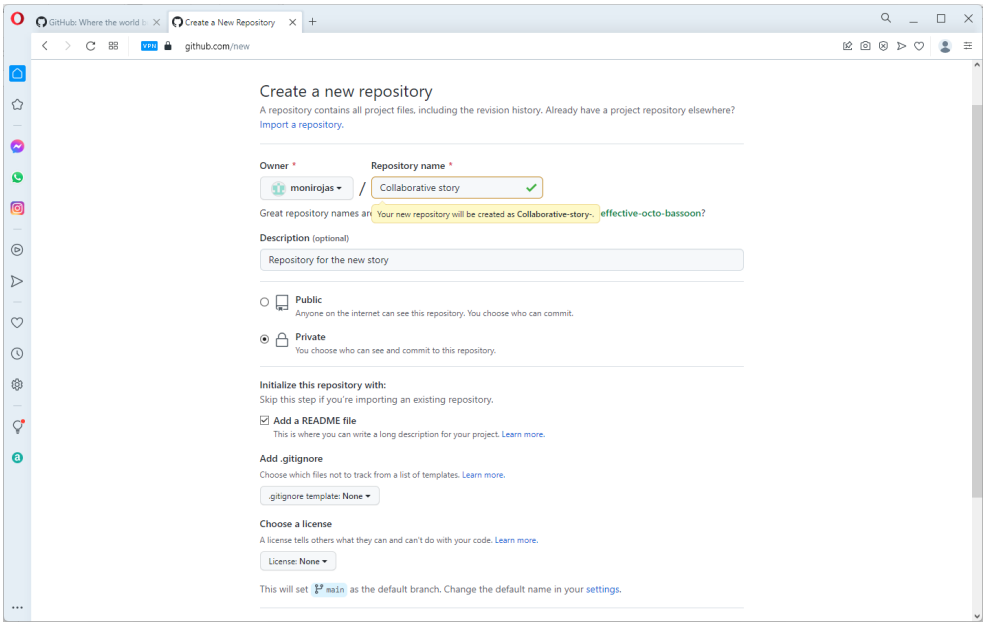
Below, you will find an illustration of how to create a collaborative story using GitHub. Please note that this serves as an example to guide you through the tasks assigned for this week. The purpose is to familiarize yourself with the GitHub workflow. You will subsequently apply a similar approach to complete this week's assignment.

Basics

- Go to the GitHub web page and sign in using your personal settings.
- Create a new repository by clicking in “Create a new repository”



- Add a meaningful name to your repository. You can choose your repository to be private or public (for everyone in the internet). Choose the “private” option (later on you will see how to share it with your collegues). You should also add a Description and activate the “Add a README file option”.
  Add an appropriate description of the project to the README file. By doing so, you will be able to keep things organized and easy to follow. The repository in this example is named “Collaborative story”

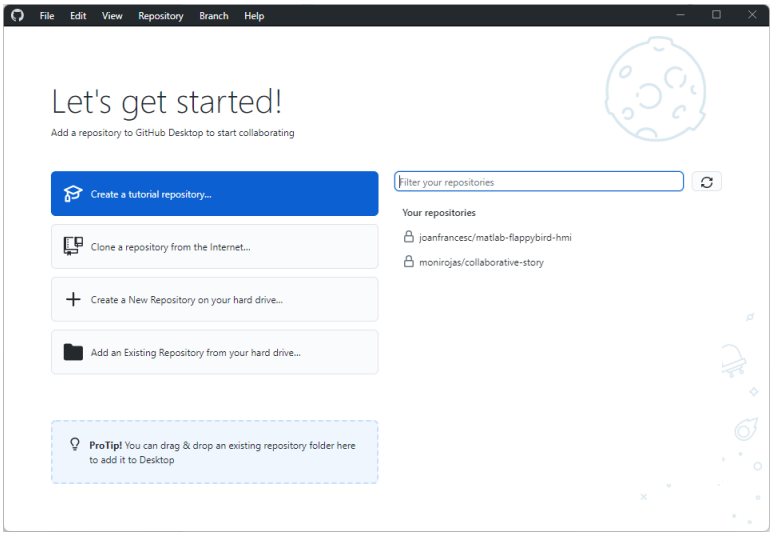


- Once you are done, click in “Create repository” at the bottom of the page. Note that by doing so you will set the current repository as the main branch of your project

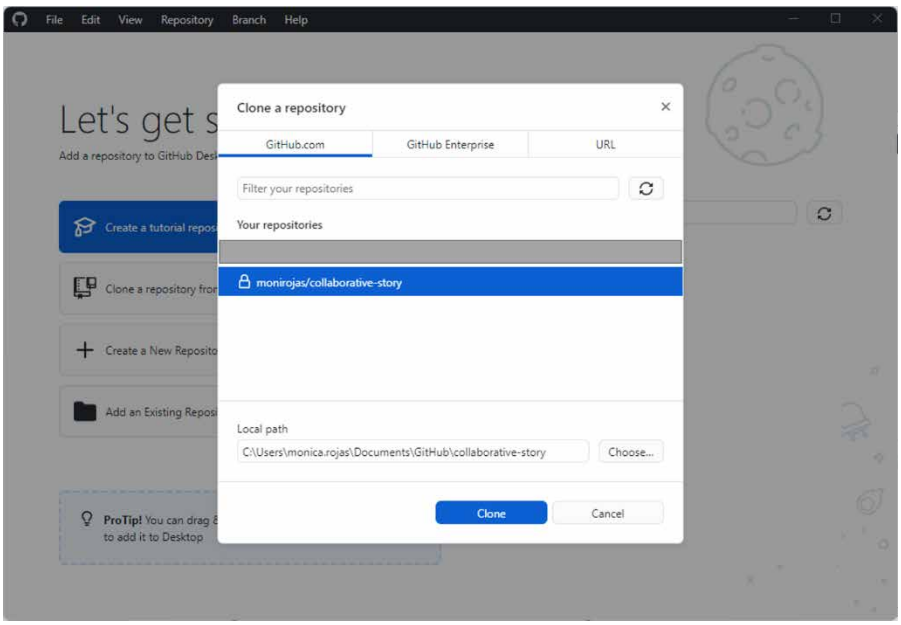
Now you can add files and modify scripts using the web application. However, you can interact with GitHub using the desktop application and work with the files stored in your

local machine. You can also copy a repository on the cloud to your machine or copy files from your machine to the GitHub cloud.

- Open the GitHub Desktop application and sign in. It will redirect you to the web page where you can sign-in using your credentials. Once you are done you will be shown a window like this in the Desktop application



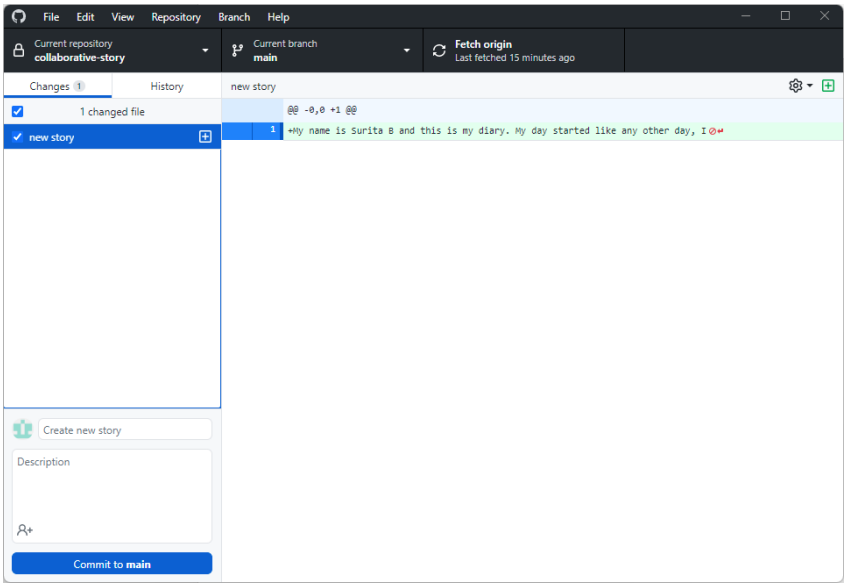
- Click in "Clone a repository from the internet", select the repository you just created in the web application and click on "clone". Your repository will be saved in the location chosen as local path at the bottom:



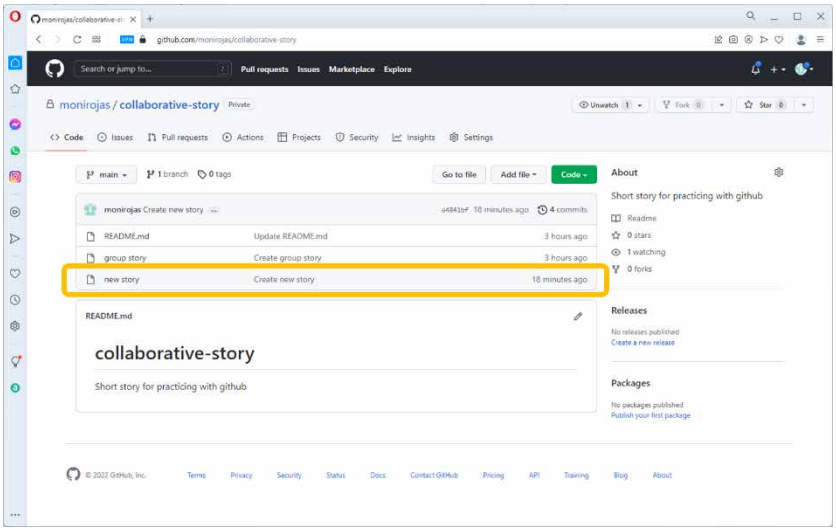
- You can then go to your local path of GitHub in your computer and check that it has been successfully created. Then you can create new files in this directory. In the example, I created a new file named *new story*.txt and added the following sentence:

  "My name is Surita B and this is my diary. My day started like any other day, I…"

- If then we check the desktop application. Something like this will appear:



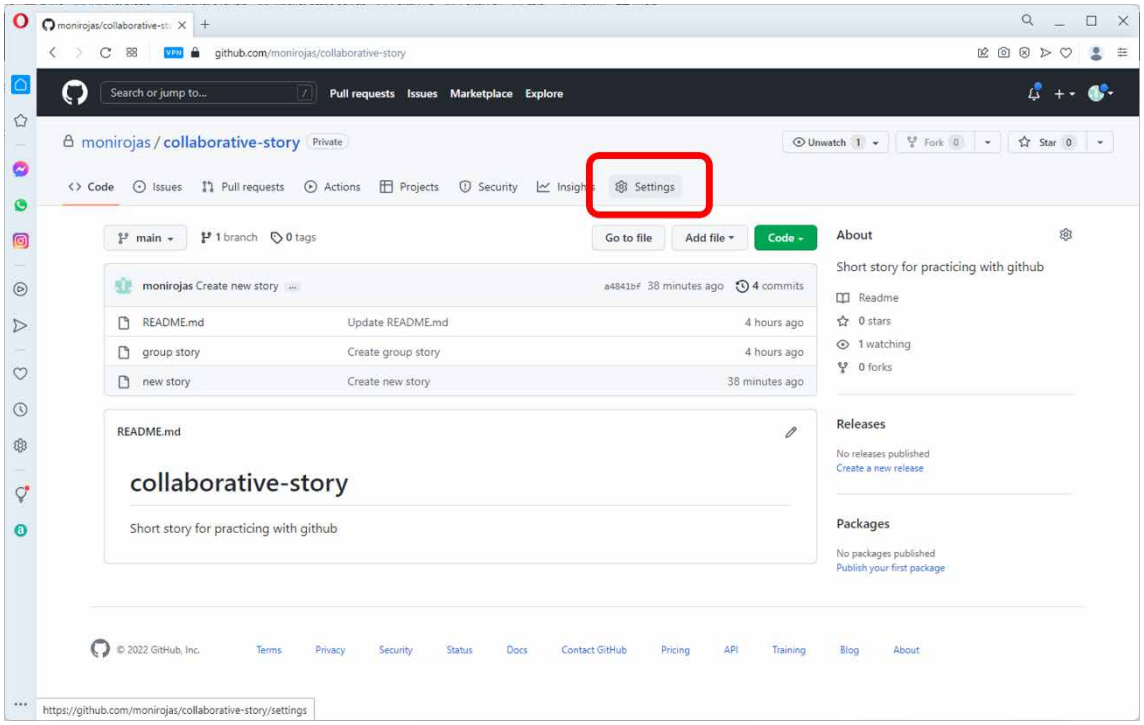
- However, this file is not saved to your repository in the cloud just yet. To do so you have to commit the changes. First, add a description (at the left side of the last window), it will l help you and others to track changes. Make sure that your current repository is the one you intend to commit the changes to and then click on *commit*. Then click on *Push origin* to upload changes to the cloud and check it in the web site. The file you created in your local machine and committed with the desktop application should now be available in your repository on the cloud.



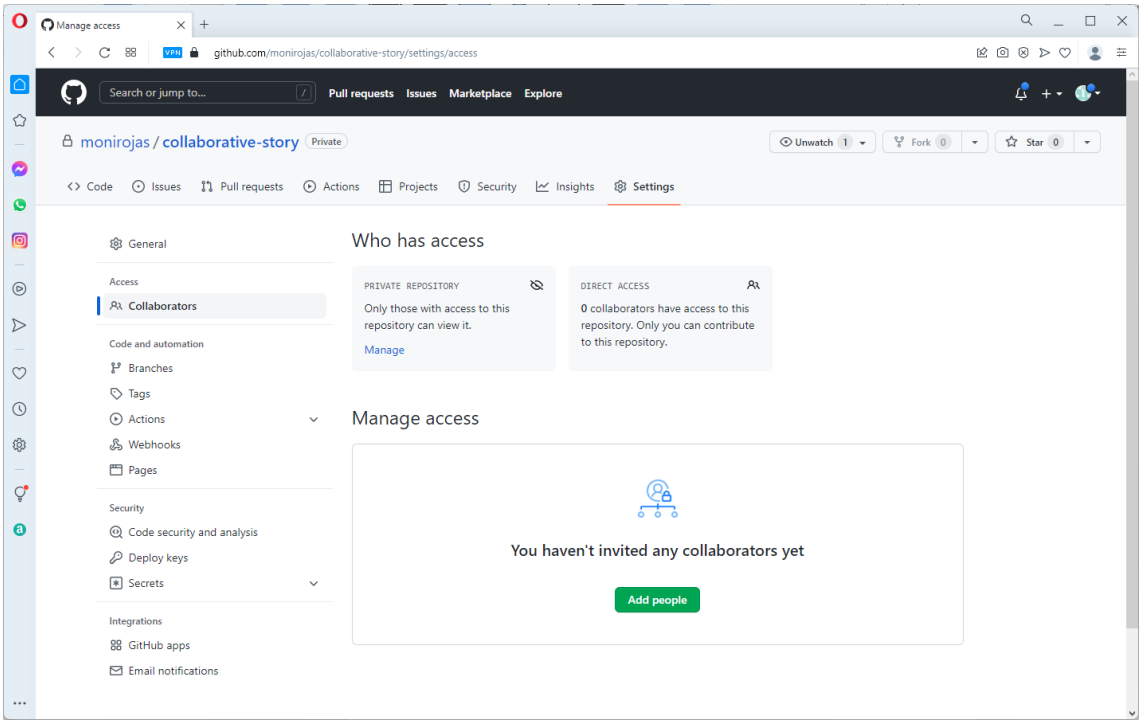
Example of a project

The objective of this task is to collectively create code to address a challenge while practicing the use of Git and GitHub tools. For every project we must choose a project leader that will be in charge of inviting collaborators using their account names

In the web site, click on the repository you will like to share and the click on *settings*

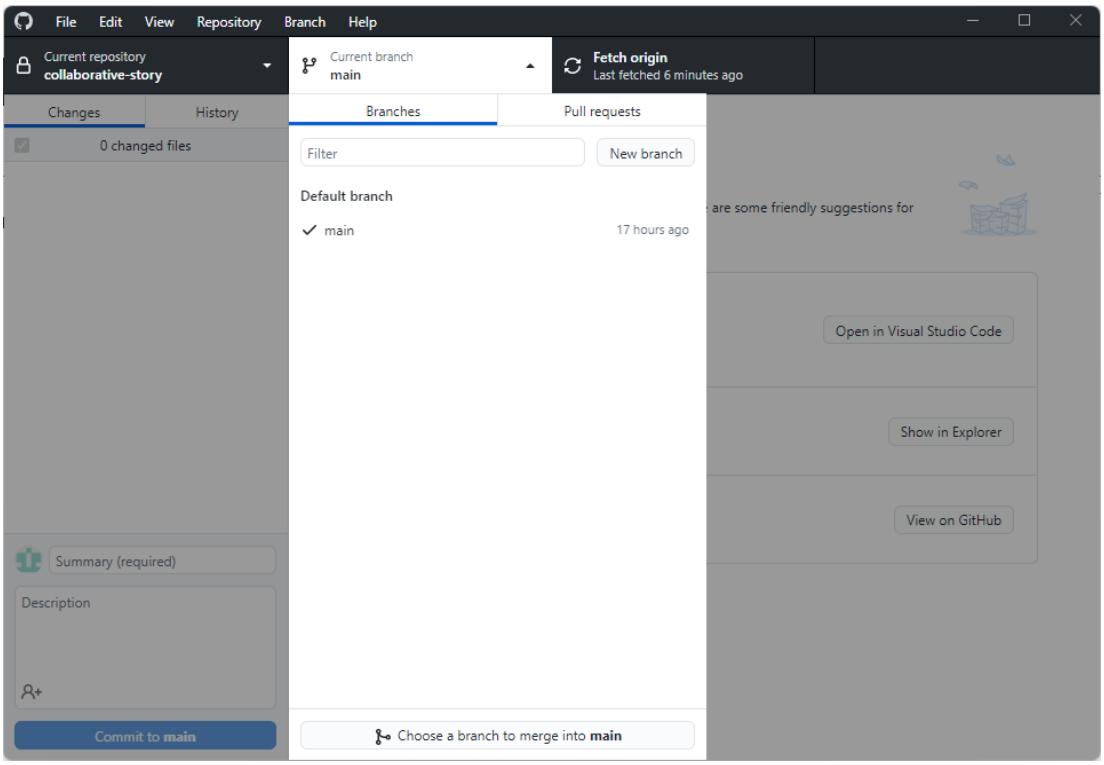


- Click on Collaborators in the left panel and then click on *Add People*. Look for the collaborators you wish to add by their usernames in GitHub and then add them to the repository one by one. An invitation will be sent to the member of the group by email and they should accept the invitation to join the repository.

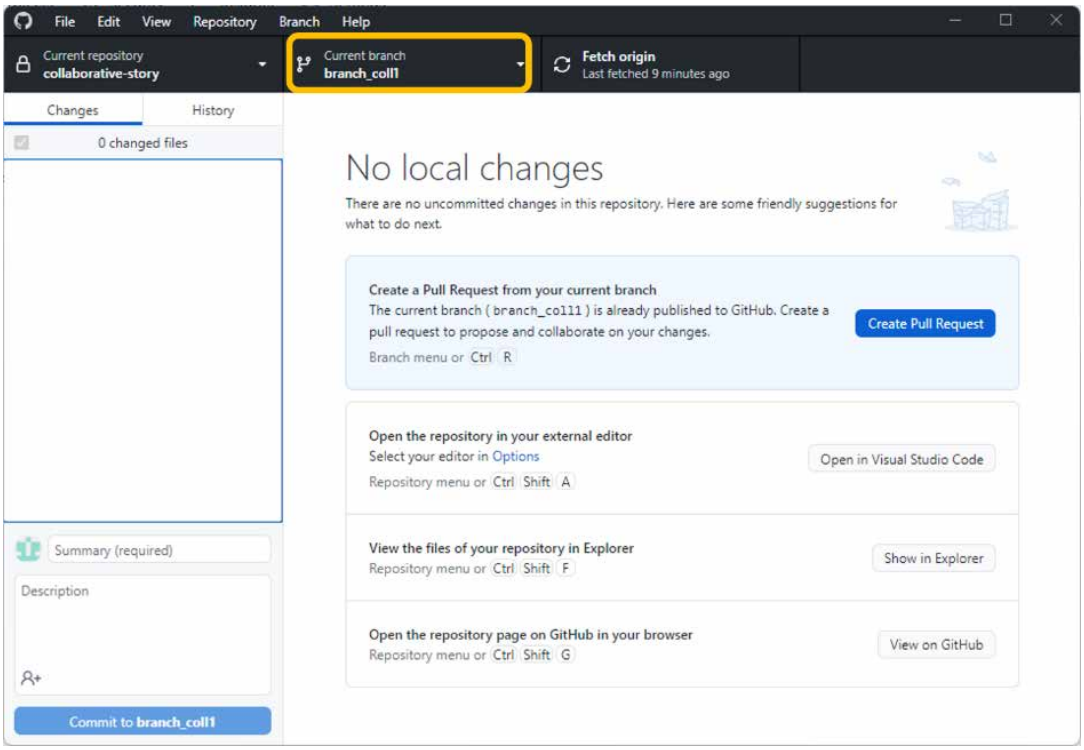


- Each collaborator should now create a new branch (including the leader). This branch will be own by each collaborator and the modifications they do to the repository will

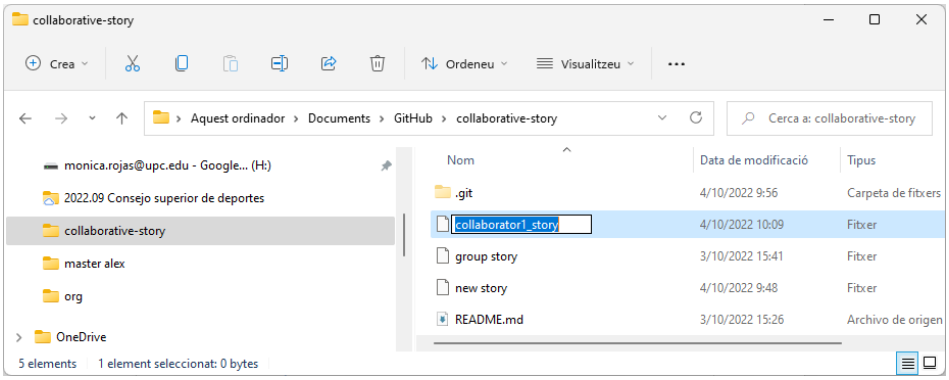
only be reflected on their paths. To create a new path, go to the Desktop application and click on *main*. You will open a dropdown like the following:



Then select New branch and give it a meaningful name. In this practice, you can name the new branch with your OWN name to recognize it easily. Click on Create branch and then on *Publish branch* to upload the changes to the repository. You should have a new branch like this:



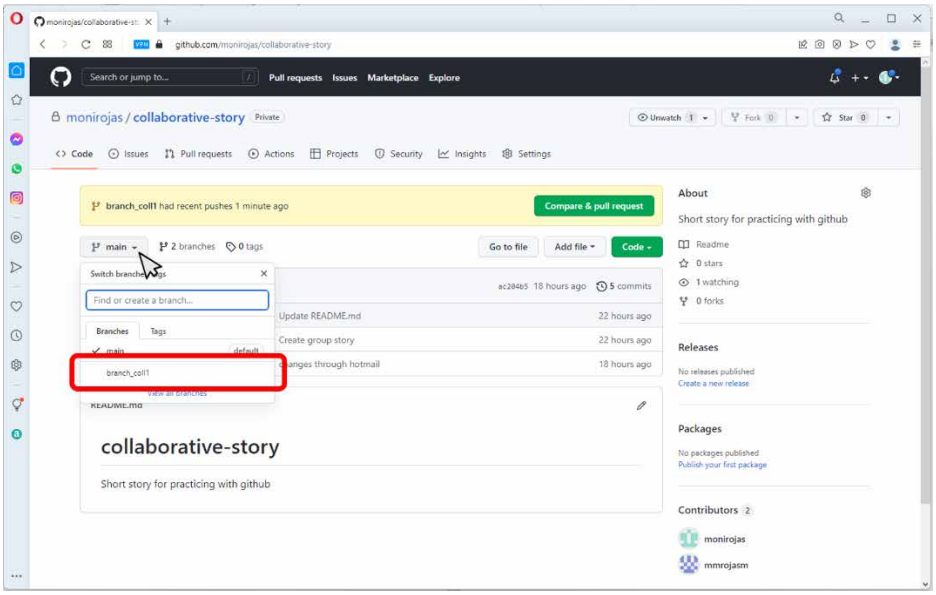
- Now you can create a new file directly in the location of your GitHub files in your computer. An example is presented as:



Once created, you can check if it is reflected on your Desktop application.

To upload changes edited in your local machine, click on *Commit to branch*_xxx and then click on *Publish origin.*

If you check on the GitHub server (i.e. on the website). You should be able to see the new branch and the file you created



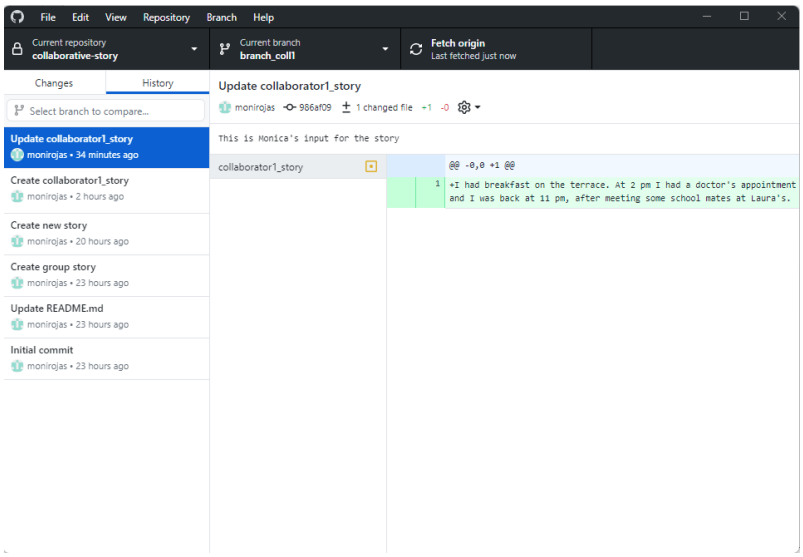
- Once on your branch you can edit files either in the desktop app or in the website. For example, if we edit the file *collaborator1_story* using the website you have to click on the pencil at the right. Just remember that for coding it is simpler to use an IDE in your local machine (like Jupyter notebook). In the example we are adding the sentence: *"I had breakfast on the terrace. At 2 pm I had a doctor's appointment and I was back at 11 pm, after meeting some school mates at Laura's."*
  When you edit something, remember to add a description of your work in the field intended for it. In addition, remember to commit changes to your branch, not to the main, until you are sure your work is correct and done. If you chose to edit a file in your local machine, you need to click on *Publish* origin after committing changes so they can be reflected on the server. You will create files like this when you create a function outside the main.

Every time you commit, you should a have a banner like this one in the web site.

If you click on *Compare & pull request. You should be able to see your changes.* Something like this:

Observe that the changes are highlighted in green. When you erase code, changes will be highlighted in red. You can also perform this operation using the Desktop app. However, if you have chosen to edit the file directly in the web site, you'll have to Pull the changes before you are able to see them in your local machine.

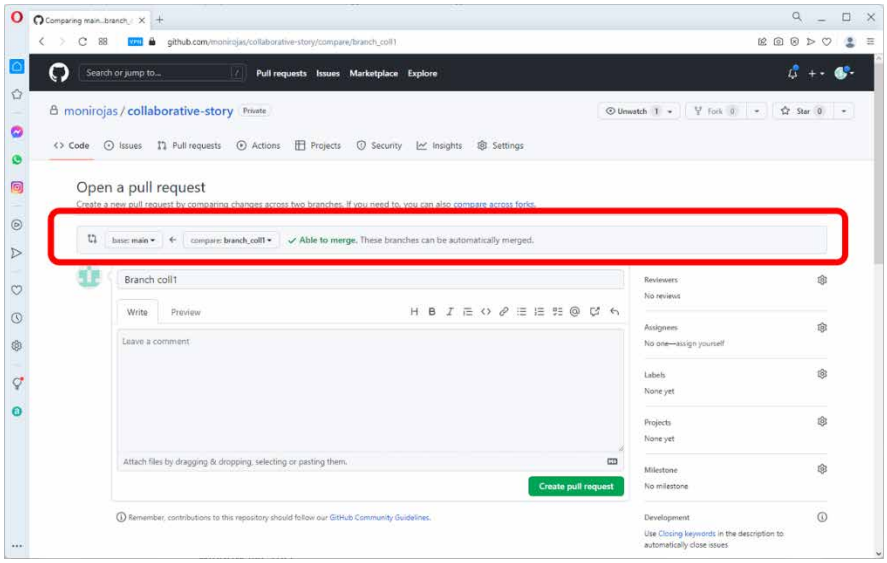
After pulling you will be able to see the changes you introduced on the History menu,



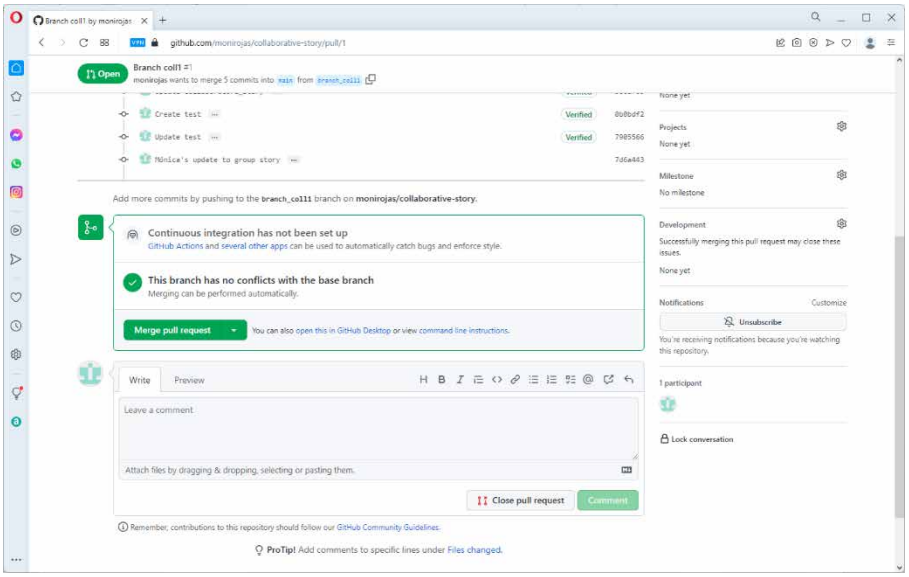
- Now let's take a look on how to add changes to an existing file shared by all collaborators, for example, the main story which is called *group_story* in this example. Since you cloned all the contents from the main branch when you created your own branch, you already have a copy of the main story. Remember: the version you have in your branch is only available for you and any change you do will not be reflected until you *commit* and *merge* your changes. Let's add the following sentence (either in your local file or in the web site):

"I had breakfast on the terrace. At 2 pm I had a doctor's appointment and I was back at 11 pm, after meeting some schoolmates at Joan's. It was supposed to be at Laura's but they changed their mind at the last minute.
I missed the bus so I had to walk 4 blocks to the nearest metro station. I had lunch with David at 12 and headed to the beach at 6 pm after finishing my work day."

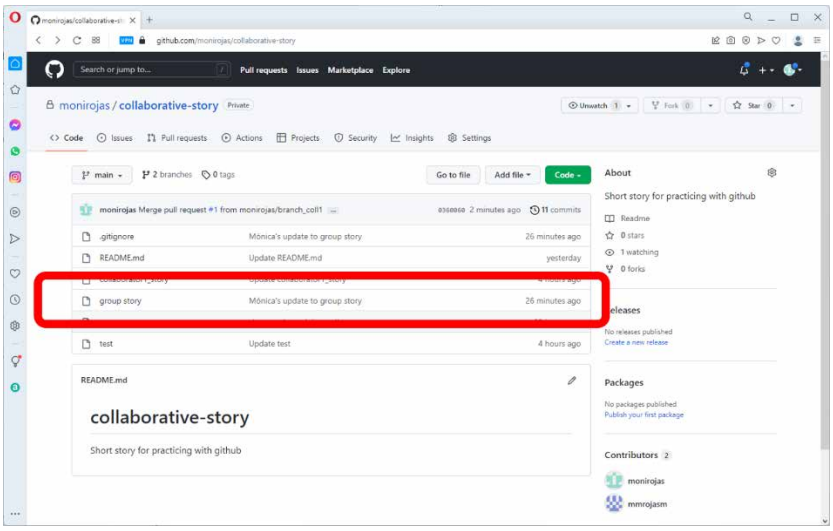
- Up to here you have the changes in your own branch and now you need to merge them with the main branch. To do so, you have to commit changes in your branch (*collaborator1_branch*) using the Desktop App. At this point you will need to add a *Resume*. If you do not do it, the app will not allow you to commit the changes. Then you have to click on *Push origin* to upload changes to the server.
- You still need to Merge your changes with the main branch. This is done by clicking on *Create Pull Request.* It will take you directly to the web page where it will show you a window like this:



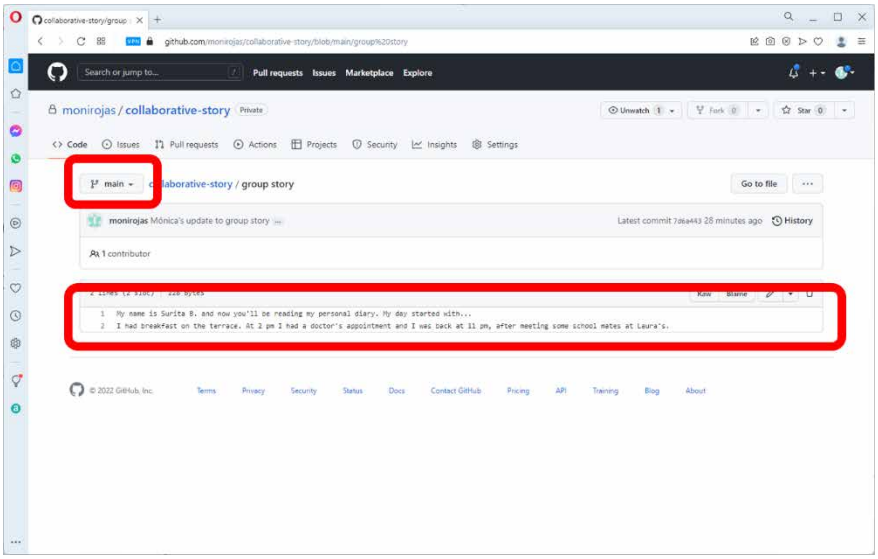
Note that there is an arrow pointing from *branch_coll1* to the *main* inside the red box. That means that you will be merging your changes with the main branch (you can do the same procedure with other branches by changing the destination). There is also a note in green indicating that you can merge the changes. To continue, click on *Create pull request.* You will be taken to a new window (in the next figure). To complete the action click *on Merge pull request*



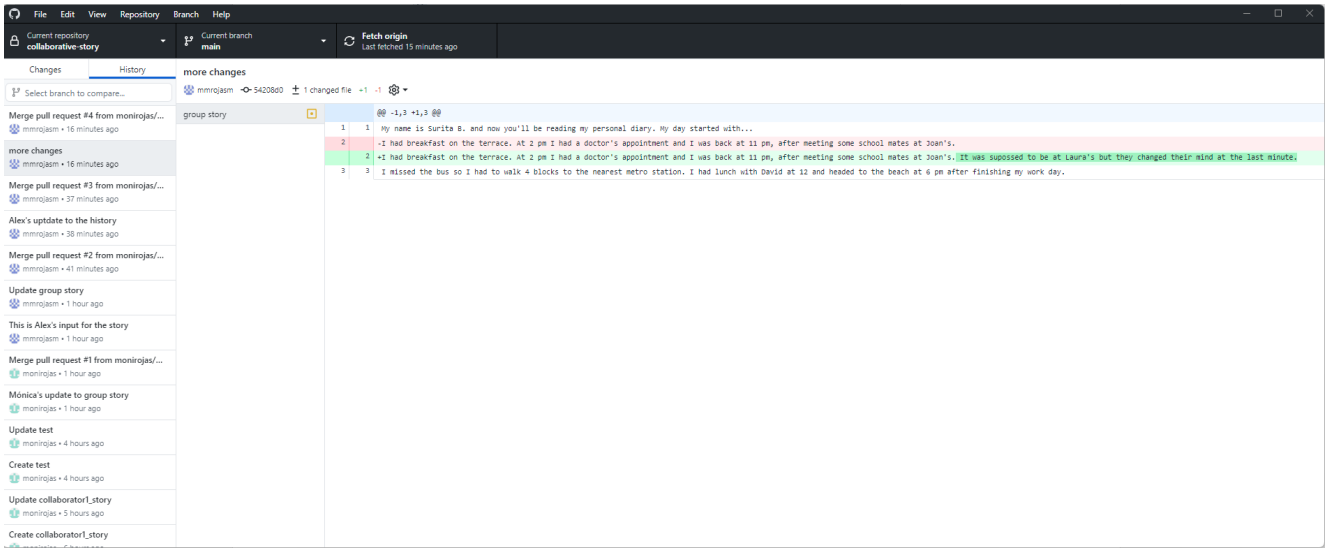
Now, if you check the main branch, there is a new record on group_story:



And if you click on the file to see its contents, it will look like this (note that changes are now reflected in the main):



The following figure shows the history of changes as seen in the Desktop app. Something similar will be obtained from the web site but the information is presented more clearly presented in the desktop application. On the left hand you will see the changes introduced by two different collaborators (icons in green and blue) and on the left, the contents of the file *group story* allocated in the main branch. Lines highlighted in red are previous versions, that is, content as appeared before. In green you will see the result of the last merge.



4. Exercises

For this week's assignment, you will be working in groups of 3 so you need to identify and get in contact with your partners. The groups are already created in the virtual campus and is configured as to allow you to send messages between group members.

You need to create a private repository in GitHub. Select one team member to be the leader. The leader will be in charge of creating the main branch and allocate the necessary files on it. He will also invite other members to the project. We encourage you to do this task together as well as to plan the tasks focusing on specific dates so you complete the work on time.

Your tasks for this assignment involve writing sections of code and/or developing various functions in Python. To accomplish this, you'll need to make modifications to the main file, named as *collaborative_coding.ipynb*. Write the functions and incorporate them into the directory where the main file is situated.

Important: all functions have to be created in separate notebooks and called from the main. Keep in mind that you may need to import some modules inside the function so it can properly work (even when the modules are imported in the main).

Before start working on the solution, each collaborator must create a branch to work on. The responsibilities for each collaborator are specified below. **Compliance with the specified task distribution is mandatory**; kindly refrain from redistributing tasks and adhere to the assigned responsibilities. It is important to note that the **tasks are not sequential**, so you and your colleagues should organize and plan your work accordingly.

Collaborator 1

- Function *renamevars* (df, dict_names)
  Rename variables on a given dataframe *df.* The function returns a *df* who's columns names are as detailed in *dict_names*
  Inputs:
  df: a given data frame
  dict_names: a dictionary for mapping the actual names of the columns in the dataframe (each key of the dictionary) to a given new simpler name (i.e. the values of the dictionary). You can follow the example in the notebook *numpy_pandas_sklearn.ipynb* for the Lab 4.
  Output:
  renamed_df: returns the input dataframe but with the columns renamed as in *dict_names*
  Please work with a version of the dataset that uses simple names for the rest of the analysis
- Summarize the data after cleaning (that is, after removing some correlations) when the data is ready. Annotate your observations. For example, how many observations do you have? Are there apparent differences between controls and patients? Is the variability comparable? If you check the minimum and maximum values are there outliers? If so, what will you do with them?
- Using the function *group_and_average,* create a dataframe for aggregating each variable of the cleaned_dataframe across trials for each subject. The resulting dataframe should consist on 32 observations, one for each subject.

Collaborator 2

- Function *scat_plt* (var1, var2, groups)
  Given the variables *var1* and *var2*, creates a scatterplot of the two variables, displaying the information given on *groups* using different colors (or symbols). That is, observations belonging to a given group 1 will be displayed on a given color, observations belonging to group 2 will be displayed with a different color and so on. *var1* is displayed in the *x-axis* and *var2* in the *y-axis.* The obtained plot should contain a legend displaying the information regarding to *groups*

Inputs:
var1 and *var2*: Two given variables of the same length
groups: A variable the same size as var1 and var2 where the information regarding to group belonging is contained.

- Based on the variable *'name'* write some code in the main that creates two new columns in the dataframe: one displaying the subject id (S1, S2, S3…) and another displaying the trial (t1, t2, t3, …, t6 or t7 in some cases). Remover the column name from the dataframe.
- Function *normalize (df, op)*
  Given a dataframe *df*, normalizes all variables according to the options in *op*. *op* can only take two values, 0 for normalizing the variables based on the z-score and 1 for normalizing the variables based on the min_max approach. The function returns a dataframe consisting on normalized variables.
  Be aware of avoiding normalizing variables that are supposed to be categorical, even if the type of such variables is not specifically categorical (that is, a variable can be of type numerical even when it represents categories)
  Inputs:
  df: A given dataframe
  op: numeric variable (either 0 or 1)
  Output
  norm_df: normalized dataframe

Collaborator 3

- Look for correlations between the variables related to the fundamental frequency ('MDVP:Fo(Hz)', 'MDVP:Fhi(Hz)' and MDVP:Flo(Hz) in the original dataframe) using the *scat_plot* function. Use the scat_plot function created by your collaborators for this purpose. If the variables are correlated keep only a subset that are representative (one or some of them). Discard the others by removing them from the dataframe. Do the same for the variables related to Jitter ('MDVP:Jitter(Abs)', 'MDVP:RAP', 'MDVP:PPQ', 'Jitter:DDP') and Shimmer ('MDVP:Shimmer', 'MDVP:Shimmer(dB)', 'Shimmer:APQ3', 'Shimmer:APQ5', 'MDVP:APQ', 'Shimmer:DDA',). Name this dataframe as *cleaned_df* and work with it from here on.
- Function *group_and_average(df, gv)*
  This function averages all variables on a given dataframe by aggregating them according to the variable *gv*. You can use the group_by operation from pandas. The function returns the averaged and aggregated dataframe.
  Inputs:
  df: a dataframe
  gv: the name of the grouping variable
  Outputs:
  av_df: averaged df

- Implement the instructions in the main code to classify the DataFrame data into patients or controls using k-nearest neighbors with k=3. Compare the outcomes across three scenarios: 1) utilizing cleaned and aggregated data, 2) cleaned, aggregated, and z-score

normalized data, and lastly, 3 the same, but normalized using the min-max option. Are there differences?

5. Evaluation

For the evaluation you have to share the repository you created to construct the project (one per group). The project has to have at least 4 branches, one for the main and, one for each collaborator. The history of changes as reflected in GIT will be reviewed for tracking the work of each team member. Please share the final repository with the GitHub account *monirojas* for the evaluation.