# 1. Create a GitHub repository from the web interface

1. Make sure you login with your GitHub account.
2. On the left of your home screen you should be able to see a list of your repositories, which may be empty if this is your first time with GitHub. At the top of this list, you will find a green button labelled “New”. Click on this button to create a new repository.
3. In the next screen you will be asked to add information about your new repository, among which a name for your repository, an optional description, whether it will be public or private, if you want to initialize your repository with a README and if you want a .gitignore file and a license. We will see each of these in details.
   1. NAME: As the advice says, repository names should short and memorable. For our purposes, put “workshop-sandbox-web”. Do not worry that all of you will use the same name as your repository will be uniquely identified by your repository’s name AND your username. This means that you cannot use the same name for two of your repositories.
   2. Description: As we have said, this is optional. If you define a README page for your repository, the description is not really necessary.
   3. Public vs Private: A public responsible can be searched for by users in GiHub and be found base on keywords found in the name, the README page or the description. If you want to protect your code, your data or your documents in the repository, it’s better to make it private. In this case, only users who have been given access to the repository can see it, clone it, commit to it or fork it (we will define these terms later). GitHub used to have restrictions on how many collaborators you could have in private repositories, but these were lifted recently. For the time being, make your repository public.
   4. README: This creates a “first page” for your repository. Here, you can specify the description of your project, probably include a user manual, how people are supposed to treat your repository (how you allow them to extend it or contribute to it) and other useful information. If you select to initialize the repository with a README, GitHub will create a branch for you, which will help you enormously with your first interactions with your repository. So, we will tick this option for our repository.
   5. .gitignore: This is a file that informs GitHub (git in particular) that there are some types of files that are not necessary and should not be committed to the repository every time you submit a change. These are often files that are automatically generated and change every time we do something with the code. The most common example is binaries as the product of code compilations. In some cases, every time you save or you run your code, these files will be produced over and over again. In this case, it does not make much sense to commit these changes in these files. So, you tell GitHub to simply ignore these files every time you commit (so you don’t have to do it manually every time). But how do you know what files you can ignore? Thankfully for us GitHub provides some default gitignore files, based on the language or the technology you’ll use in your project. In our case, we will predominantly use R, so we specify this as our gitignore file.
   6. license: This is an important decision, especially concerning projects that involve privacy or copyright. Unfortunately, we will probably not have the time to cover in this workshop (hopefully in a future one). Thankfully for us, GitHub provides a good guide on how to pick a license and what all the licenses include (<https://choosealicense.com/>). For our project, we will go “simple and permissive” and go with the MIT license which “lets people do almost anything they want with your project, like making and distributing closed source versions”.
4. Once everything is set up, go ahead and click the green button “Create repository” at the bottom of this screen.
5. Congratulations! You have created your first repository! Doesn’t it look cool??

# 2. Create a GitHub repository from the app

1. Make sure you have downloaded, installed and set up the GitHub Desktop app.
2. Select File->New repository…
3. You will be asked to provide the information for the new repository, including name, description, README, .gitignore, and license, as previously. GitHub Desktop will ask you for one extra field, which is the local path. This is the directory where all repositories (new and cloned) will be stored in your computer. When installed, GitHub Deskop will create a folder by default. You can leave it as is for now.
4. For the name use “workshop-sandbox-app” and the rest of the parameters as in the previous step.
5. As we are informed by the GitHub Desktop, the repository is only available in our local machine (as opposed to what happens when we created the repository online). Therefore, we have to publish the repository to GitHub, which can be done by the blue button in the application interface.
6. Once we do this, we are asked for a little more information. The name and the description are usually inherited by the repository we just created. If you want the repository to be private, you can keep the respective box checked. For this example, you can uncheck it. You can leave the Organisation as “None”, as we have not created any organisations yet. Make sure from the top of this dialog box to select “GitHub.com” since we do not have an upgraded account for GitHub Enterprise Server.
7. Once everything is set, hit “Publish repository” once again.
8. From here we can go to our repository by navigating to github.com or by hitting the “View on GitHub” button from the app’s interface.
9. As you can see, the two processes produce almost identical results.

# 3. Clone a repository

1. The simplest way to clone a repository is through the GitHub Desktop app. Now, concerning how you start the process, you still have two options: from the web or from the app.
   1. From the web: In the web interface of the repository, you can click the green button labelled “Clone or download”. This will give you a number of options, including to use Git through the command line, or download the contents of the repository as a compressed file. For our case, we will select the “Open in Desktop” option. This option assumes you have already installed and set up the GitHub Desktop app. If you haven’t, you will be prompted to do so.
   2. From the app: Select File->Clone repository… Initially, you will be presented with a list of repositories to clone from your GitHub account. Since we do not own the repository we want to clone, we will not find in this list. So, from the top select the “URL” tab. There you can either use the URL which is provided by the web interface when you hit the green button “Clone or download”. Alternatively, you can simply specify the username/name of the repository (remember: this combination uniquely identifies a repository). For our case, put “{username}/workshop-sandbox-web”, where you can replace {username} with your actual username in GitHub. You can leave the default local path as is.
2. Whichever method you pick, once you hit clone in the app, the repository will open in the app. Also, if you navigate to the default location you will find that a directory with the name of the repository, and its contents, has been created there.
3. From then on, we can start making changes or creating new content (using RStudio for example) for our cloned repository.

# 4. Version Control: My first commit

## 4a. Version control using R studio

1. Open RStudio
2. Go to File->New Project…->Version Control->Git
3. Go to your repository on github.com and click on the green button labelled “Clone or download”.
4. Grab the repository URL and paste on the field in the dialog box in RStudio.
5. In RStudio open dialog box. As the project directory name use the repository name and leave subdirectory path as the default GitHub directory.
6. Take the R script you have written and move it to the directory created for the repository in the default GitHub directory.
7. After finishing your edits, return to RStudio and go to Tools->Version Control->Commit and select the R script you just added. At this point, you will be asked to also provide a short description of the changes you have made. In the version control vernacular, this is known as a “commit message” and it has been made mandatory by such systems as GitHub. The commit message is vital as it informs developers what has happened and it helps them to track the history of the project.
8. After you complete your commit message you can select “Push Branches”. This action in GitHub is known as “Push to the stream”, where the stream is the repository. This is because Git uses a “two-phase commit”. In the first phase, we commit changes in our local repository. At this point the history is tracked in our local repositories, but changes are not visible to other collaborators yet. To make them visible, we need to push our local commit to the stream (usually called the “master branch”).
9. Now your changes have been uploaded on GitHub.

## 4b. Version control using GitHub Desktop App

1. Take the R script you have written and move it to the directory created for the repository in the default GitHub directory.
2. Open your GitHub Desktop App and make sure you are in the workshop repository. To check that, in the top left corner, make sure that the tab “Current repository” points to the proper repository.
3. If you have copied the file, you will see that there are changes to check in the list on the left of the interface.
4. On the right, you can see the changes on the file you have selected from the left, in terms of lines added or deleted. A modified line is counted as one line deleted and another line added.
5. At the bottom left, there are two fields, one for the commit message and another for the description. Being mandatory, the Desktop app automatically generates a commit message based on the context of the action, in this case “Create file”. If you wish, you can change the default message to something more descriptive. For our purpose, we can leave it as is.
6. You can also add a more extended description, which is optional.
7. Once everything is done, click “Commit to master”. “master” is the main branch. If we had different branches, we would have to select the desired branch from the second tab at the top of the interface.
8. After committing, we should also push to the stream, by clicking on the “Push origin” button.
9. Now your changes have been uploaded on GitHub.

# Day 2

# Create and use branches and forks

1. Branches can be used to isolate development work without affecting other branches in the repository. Each repository has one default branch and can have multiple other branches. You can merge a branch into another branch using a pull request. You can use branches to: Develop features, fix bugs, Safely experiment with new ideas, etc…
   1. You cannot create branches directly through RStudio. To create a new branch, navigate to the main page of the repository, click the drop-down branch selection menu at the top of the file list that says **branch: master**
   2. Type a unique name for your new branch, in our case, type “features”, then select the **Create branch: features** from ‘master’ box or hit “Enter” on your keyboard. This will copy all the files and commits from the base branch (called master) to the new created branch and set it as actual branch on your interface. The drop-down branch selection indicates on which branch you currently are.
   3. Further, If you want to create a branch from any other branch, make that the branch you are actually working on is that base branch. Let’s say you want to create a branch ‘fix plots’ from the “features” branch and that you are actually on the master branch, you will have to switch to the ‘features’ by selecting it, then write ‘fix plots’ and finally select the **Create branch: ‘fix plots’** from ‘features’ box.
   4. In RStudio, to switch between branches, first pull the remote repository to gather all new changes (including the branch you just created), then select the top right drop down menu that has the name of your current branch (master) and select the branch “features”. Now every commits and push you make will update just the branch ‘features’. When your work is done, you can then create a pull request to merge your branch with the base branch (the master branch or any other branch).
2. A fork is a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project. Most commonly, forks are used to either propose changes to someone else's project or to use someone else's project as a starting point for your own idea. Forking a repository is a simple two-step process. Let’s for the massaraevi/GitHub\_workshop2020 repository
   1. Navigate to the <https://github.com/massaraevi/GitHub_workshop2020> repository. In the top-right corner of the page, click **Fork**.
   2. A copy of the repository will be created on your GitHub account, and you can edit it as if you created it yourself. If you just want to contribute to the project, you can create a pull request to the original project for your contribution to be integrated in the original project.

# 2. Proposing changes with pull requests

1. After you add changes to a branch or fork, you can open a pull request to ask your collaborators or the repository administrator to review your changes before merging them into the project. When thinking about branches, remember that the base branch is **where** changes should be applied, the head branch contains **what** you would like to be applied. We are going to add some modifications to the features branch, then create a pull request to the master branch.
2. Add some modifications to the readme file of the ‘features’ branch in RStudio
   1. In RStudio, switch to the feature branch, then add the line “this line for pull request 1” to the readme.md
   2. Commit and push your changes (pull, stage, commit, push)
3. On GitHub, navigate to the main page of the repository, then in the "Branch" menu, choose the branch that contains your commits. In our case, it’s the ‘features’ branch that we’re going to merge with the master branch
   1. To the right of the Branch menu, click **New pull request**.
   2. Use the base branch dropdown menu to select the branch you'd like to merge your changes into (in our case, select the ‘master’ branch).
   3. Type a title (“pull request 1”) and description (“Added a line on readme for pr 1”) for your pull request
   4. click **Create Pull Request.** The administrator will then review and accept or reject your pull request

# 3. Merge and conflict resolution

1. After a pull request has been opened, you (as the repository administrator) can review and discuss the set of proposed changes, then merge the changes to the specified branch.
   1. On GitHub, you can see the changes made by the pull request creator, suggest modifications, add comments, mark a file as viewed and submit your review
2. By default, any pull request can be merged at any time, unless the head branch is in conflict with the base branch. Let’s merge pull request 1 to the master branch.
   1. Navigate to the main page of the repository and select the “Pull Requests Tab”. By default, it will display the opened pull request waiting to be merged.
   2. Select “pull request 1” to open it. You will se the details about the pull request such as description and the commits included in that pull request.
   3. To merge the pull request, you can click the green drop down button “Merge pull request and select the merge method you want.
      1. the default **Merge pull request** option on a pull request on GitHub, all commits from the feature branch are added to the base branch in a merge commit
      2. When you select the **Squash and merge** option on a pull request on GitHub, the pull request's commits are squashed into a single commit. Instead of seeing all of a contributor's individual commits from a topic branch, the commits are combined into one commit and merged into the branch.
      3. When you select the **Rebase and merge** option on a pull request on GitHub, all commits from the topic branch (or head branch) are added onto the base branch individually without a merge commit.
      4. Using, Git, it is possible to select only certain commits to merge to the base branch. This is called cherry-pick. Unfortunately, GitHub doesn’t have any feature for this, and it should be done using the Git command line or a Git GUI.
   4. Let’s select the default Merge pull request option, then select confirm merge to merge and automatically close the pull request.
3. Merges (from pull request on GitHub or from simple pull in RStudio) can create conflicts. They occur when GitHub can’t make a decision about what should be on a particular line of a project file modified by different collaborators (or when you work on the same file from different pc) and needs a human (you) to decide. Let’s create and resolve a merge conflict.
   1. go on GitHub, switch to features branch and change the line 2 of readme.md file to “Changed line number 2 from GitHub”, then commit with commit name “readme GitHub update”
   2. Now let’s go to RStudio, make sure we are on features branch and replace the line 2 of readme file with “Changed line number 2 from RStudio”, then save the file. Now let’s start the commit pipeline by first pulling from remote repository
   3. We get and error! There is a merge conflict. So, we are not allowed to pull, it failed. GitHub is protecting us because if we did successfully pull, our work on RStudio would be overwritten by whatever we had written on GitHub. So, GitHub is going to make us decide. GitHub says, either commit this work first, or “stash it” (It means saving a copy of the README in another folder somewhere outside of this GitHub repository).
   4. Let’s follow the advice and commit.
   5. Now let’s pull again, we still get an error. Actually, we’re just moving along this same problem that we know that we’ve created: We have both added new information to the same line. This kind of conflicts often occur when different collaborators are working on the same line in the same file. Let’s close that window and inspect. Notice that in the git tab, readme.md is tagged orange U; this means that there is an unresolved conflict, and it is not staged with a check anymore because modifications have occurred to the file since it has been staged.
   6. Let’s look at the README file. We got a preview in the diff pane that there is some new text going on in our README file showing the local and remote changes.
   7. So, to resolve this merge conflict, we have to choose, and delete everything except the line we want. So, we will delete the <<<<<<HEAD, =====, >>>>long commit identifier
   8. Now let’s commit and push. It works.
4. Pulling, syncing and committing often will help reduce the number of conflicts.