BRO Academy

Manual

GIT

Version Control System

If you are working on a project with a team, you need to share code with your colleagues. Even if you are working alone, it is useful to keep a history of code changes so you can always understand when and what changes were made.

Often developers work in teams on the same project, which means several people can change one file simultaneously. To avoid confusion, version control systems are used, allowing you to store the project's change history and, if necessary, revert to a previous version. One of the most popular systems is Git.

A version control system (VCS) is like a "time machine" for programmers. When you work on code, VCS allows you to save different versions, go back in time, and see what changes were made. It helps scientists, programmers, and anyone creating something to track changes and manage the development process.

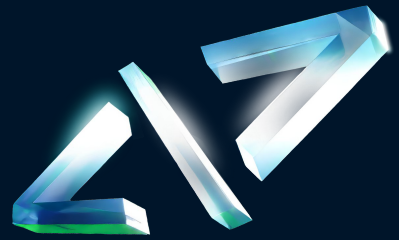
Imagine a gamer reaching the final level, losing, and returning to the beginning of the level—reaching the nearest checkpoint allowed by the developers. Without checkpoints, you would have to start the game from the beginning after each loss.

Or imagine writing a book. You make different versions—edit, add new pages, and correct mistakes. It would be convenient to save each version separately to always go back or see how your thoughts changed. In the programming world, something similar is used—version control systems.

Why use Git

- Storing the complete history of changes
- Describing the reasons for all changes made
- Rolling back changes if something goes wrong
- Finding the cause and person responsible for errors
- Team collaboration on a single project
- Ability to change code without interfering with colleagues' work

GIT



Git — is a distributed and decentralized version control system.

A decentralized system means that each developer has a personal project repository with a complete set of all versions. All necessary files for work are on the computer. Constant network connection is not required, so the system works quickly. For team development, repository synchronization is needed, as the project is one, and its state should be the same for everyone.

Initially, it was developed for Linux development.

Git is a tool that allows programmers to save, manage, and track changes in the code. It allows creating "snapshots" of the code (called commits) and tracking how the code changes over time.

At the same time, Git is a program that needs to be installed and connected to the project for version control management.

Parts of git

Repository

— a directory in the file system containing: configuration files, log files of operations performed on the repository, a file location index, and a storage containing the controlled files themselves.

Local Repository

— a repository located on the developer's local computer in a folder. This is where development and change fixation occur, which are sent to the remote repository.

Remote Repository

— a repository located on a remote server. This is a shared repository where all changes come and from which all updates are taken.

Our remote repositories for your projects will be on GitHub.

Parts of git

Fork

— a copy of the repository. It can also be considered an external branch for the current repository. A copy of your public repository on GitHub can be made by any user, after which they can send changes to your repository via a pull request.

Storage

— the content of the hidden .git folder. This folder contains all versions of the working area and service information. These versions are automatically named by the system, consisting of letters and numbers. For example, bea0f8e and d516600. Do not manipulate the .git folder manually. All work with the system is done through commands using special applications or the console.

Clone

— downloading the repository from a remote server to the local computer into a specific folder for further work with this folder as a repository.

Parts of git

Branch

— a parallel version of the repository. It is included in this repository but does not affect the main version, allowing free parallel work. When you make the necessary changes, you can merge them with the main version.

Master

— the main or primary branch of the repository.

Commit

– is like saving the current state of the project in a version control system. There is a special command—commit. It saves the new version of the project and adds it to the storage. The save file shows all changes that occurred in the working area, the author of the changes, and a brief comment describing the essence of the changes. Each commit stores the full state of the working area, its folders, and project files.

Parts of git

Pull

If working on a project in a team, you need to get the latest version of the project before starting to write code. For this, execute the pull command. This way, you get all the changes made since the last synchronization with the remote repository. Now they are in your local computer repository.

Push

To send colleagues the latest version of the project, execute the push command. If there have been no changes in the remote repository since the last synchronization, all saved changes will successfully upload to the cloud, and colleagues will get the latest version by executing the pull command. If there were changes, Git will ask you to pull the latest versions before sending them.

Pull Request

— a request to merge a fork of the repository with the main repository. The pull request can be accepted or rejected by you as the repository owner.

Parts of git

Merge

— merging changes from one repository branch with any branch of the same repository. Most often, merging changes from a repository branch with the main repository branch.

Code review

— the process of checking the code for compliance with certain requirements, tasks, and appearance.

How the project works:

The repository stores all versions of the project. When transferring this project to another person, they will see everything that has happened to it before.

Nothing is lost or deleted without a trace. When a file is deleted in the new version, a record is added indicating that the file was deleted.

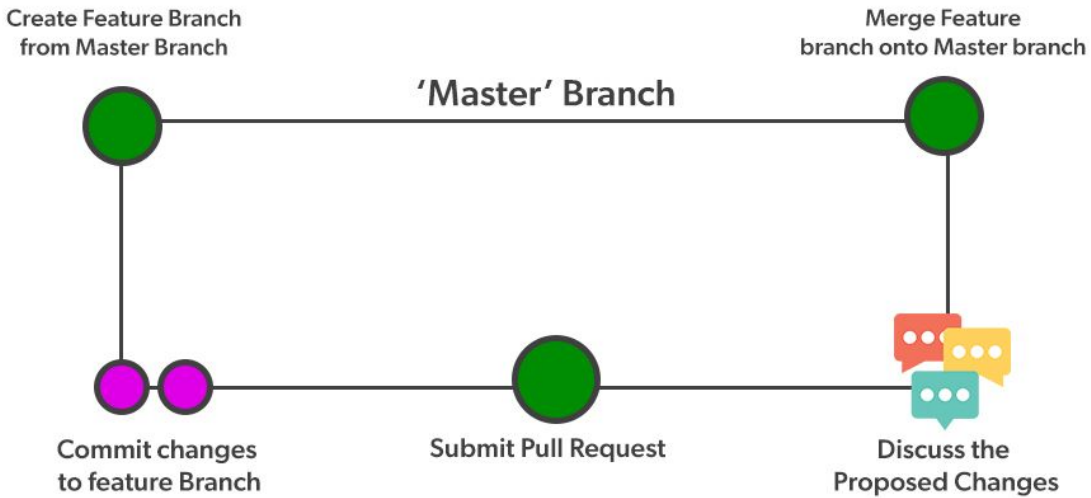
You can always return to any version of the project by loading it from storage into the working area.

How to synchronize repository data between developers?

Initially, Git repositories can synchronize from user to user by themselves. Additional programs are not needed for this. There are special console commands to transfer data from one repository to another.

This method is complex and rarely used. Developers usually synchronize the local repository with the remote one. A remote repository is the same repository, only its data is in the cloud.

Process of working with the repository



How to use Git with GitHub

01. Creating a repository

A repository is a place where you will store your code. On GitHub, you can create a repository representing your project.

02. Cloning a repository

When a repository is created on GitHub, you can **clone** it to your computer. It's like downloading a book from the internet to work on it locally.

03. Adding and changing code

You edit your code, and every time you make a change, you save it with a "**commit**." It's like taking a snapshot of your book's current state.

04. Sending changes to GitHub

After making changes to your code and committing them, you can **send** these changes to GitHub. It's like sharing your book with other readers.

05. Collaborative work

Now other people can see your code, comment on it, and even suggest changes. You can discuss changes and make them in your code.

06. Updating code

When someone makes changes to your code, you can **get** these changes on your computer. It's like downloading a new chapter of a book written by someone else.

GitHub Flow

GitHub Flow - is a way to work with Git and GitHub suitable for small and medium projects. It allows efficient collaboration on code and making changes. Here's how it works:

Creating a branch

When you start working on new code, you create a new branch in the repository. In this **branch**, you will make your changes.

Adding changes

You add, edit, and delete files in your branch. Each change is called a **commit.** You can make many commits until you finish your work.

Sending a pull request

When ready to share your changes, you create a **pull request**—like proposing changes to the main code branch. Others can review, discuss, and evaluate your changes.

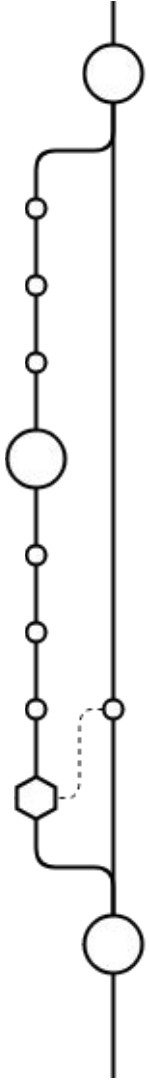
Discussing and making edits

During the pull request discussion, you may get comments and suggestions for improvement. You can **make edits** until everything is ready.

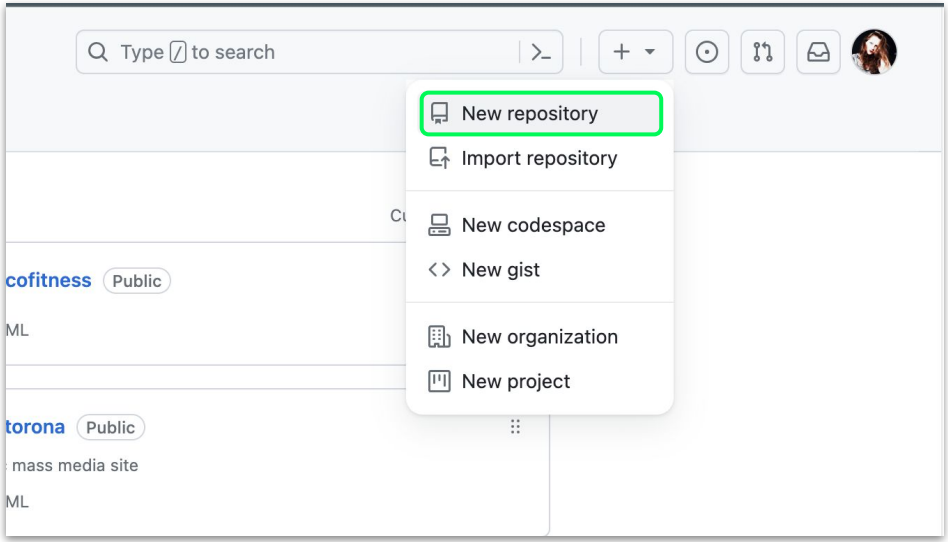
Merging changes

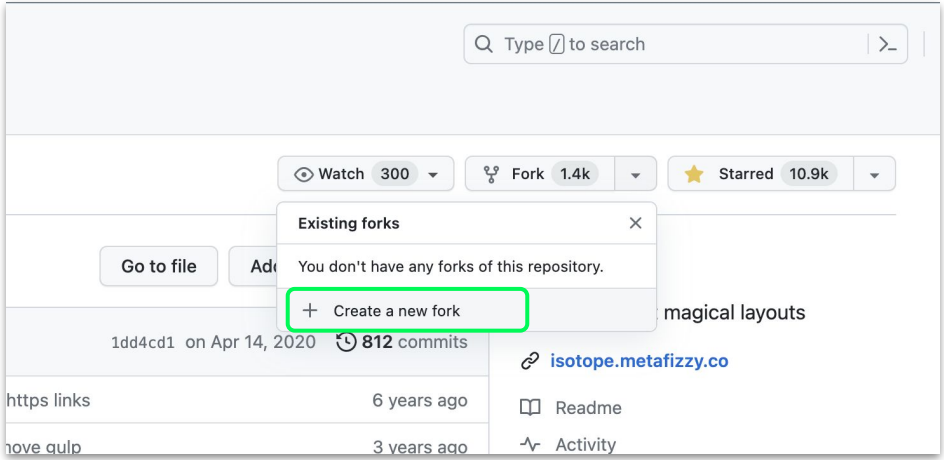
When your pull request is ready, you can **merge** your changes with the main code branch. This way, your code becomes part of the project.

Git workflow for a task:



1. Create a new branch for a current lesson.
2. Write some code.
3. Commit the changes, specifying what was done in a commit message.
4. Push the changes to the repository.
5. Create a pull request.
6. Change the status in Jira to "In review". Include a link to the open pull request in the task comment.
7. If the task is reopened, repeat the process from step 2.
   A new branch is not needed.
8. Merge PR after approve
9. Pull changes from remote master to local master





1

Creating a Repository

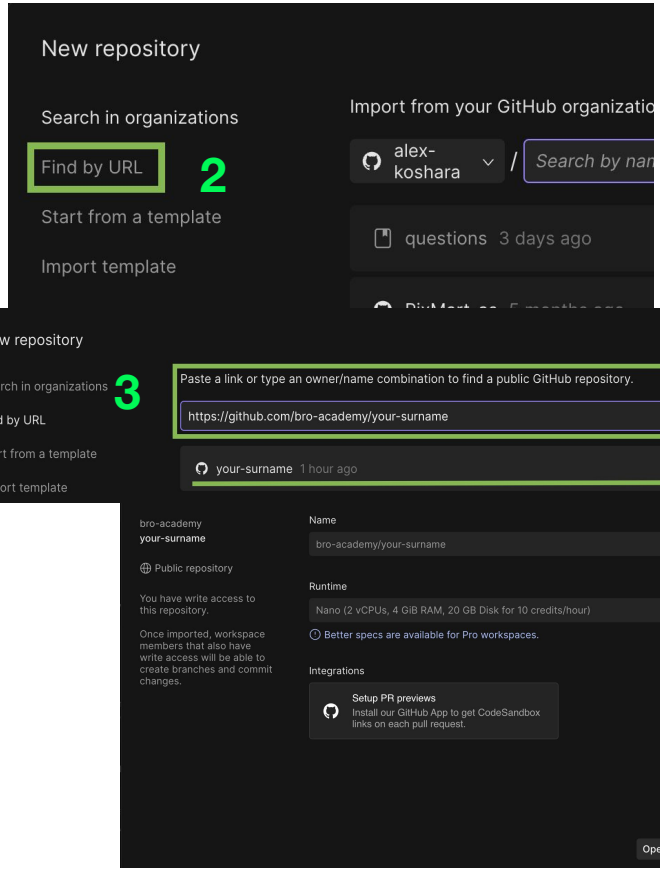
or

Forking an Existing Repository

CLONE IN CODESANDBOX.IO

Using **Codesandbox**

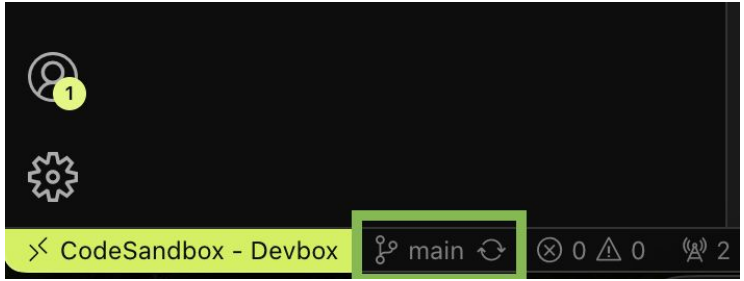
1. Click on **Import** button
2. CHOOSE **Find by URL**
3. Copy **URL** from your repository in **Github**
4. Click the yellow button **Import Repository**

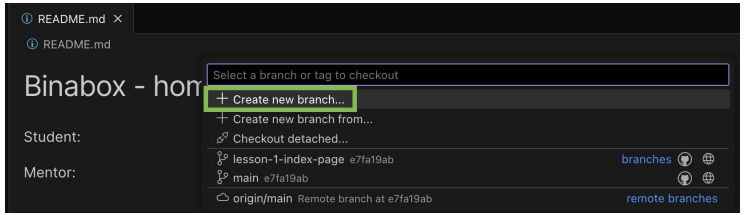


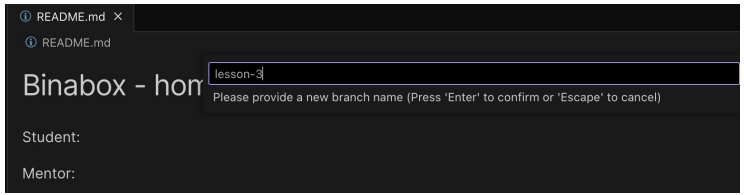
Creating branch

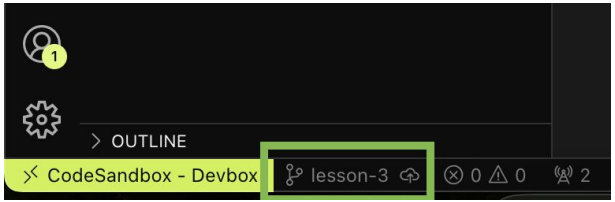
Using **Codesandbox**

1. Click on **main** button
2. Type branch name or click on **Create new branch** and type **lesson-***number-of lesson*
3. Press **Enter**
4. You can look **lesson-** branch in the left bottom corner after creating branch





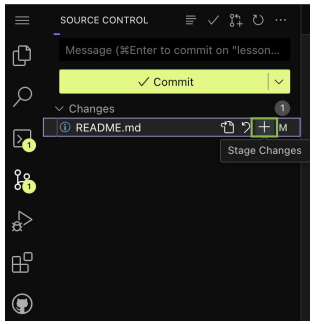


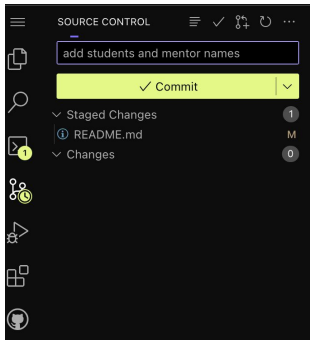


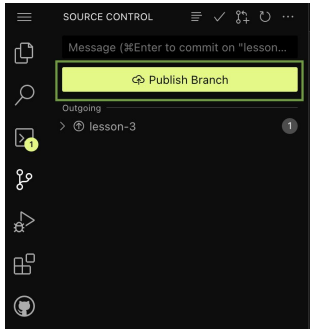
Modifying Code

1. In the left panel, select **Source Control** and open the first item – Source Control to see the changed files.
2. Select the changes to include in the commit.
   a. You can click the **+** button in the **Changes** section to select all changes.
   b. If you forget to select changes, a warning will appear, and you can safely click **Yes**.
3. Enter a message in the **Message** field (it should answer the question "What does this commit do?").
4. Click the **Commit** button.
5. Click the **Publish Branch** button to perform the push action.

Check your repository on **GitHub.com** to ensure all changes have been applied.

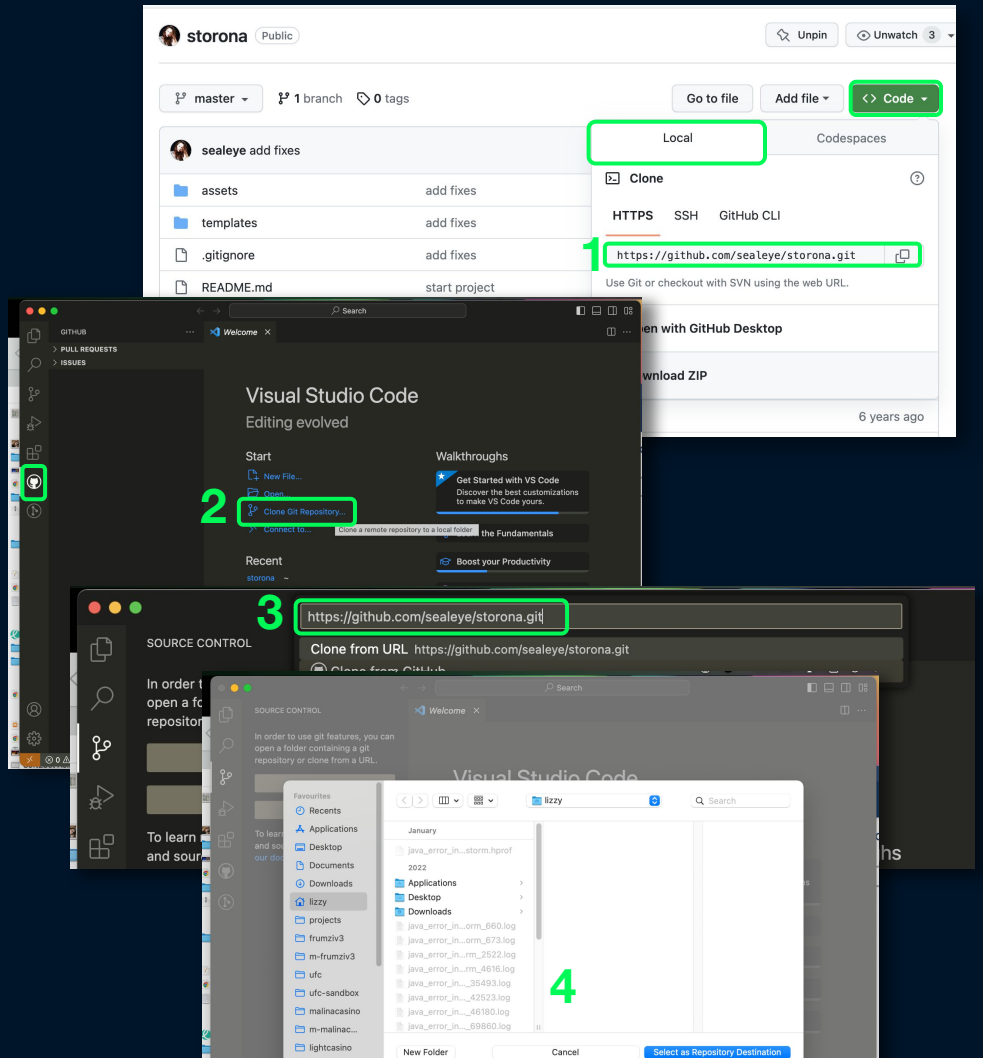






IF YOU’D LIKE TO USE

VS CODE



2.1

Clone

Using **VS Code**

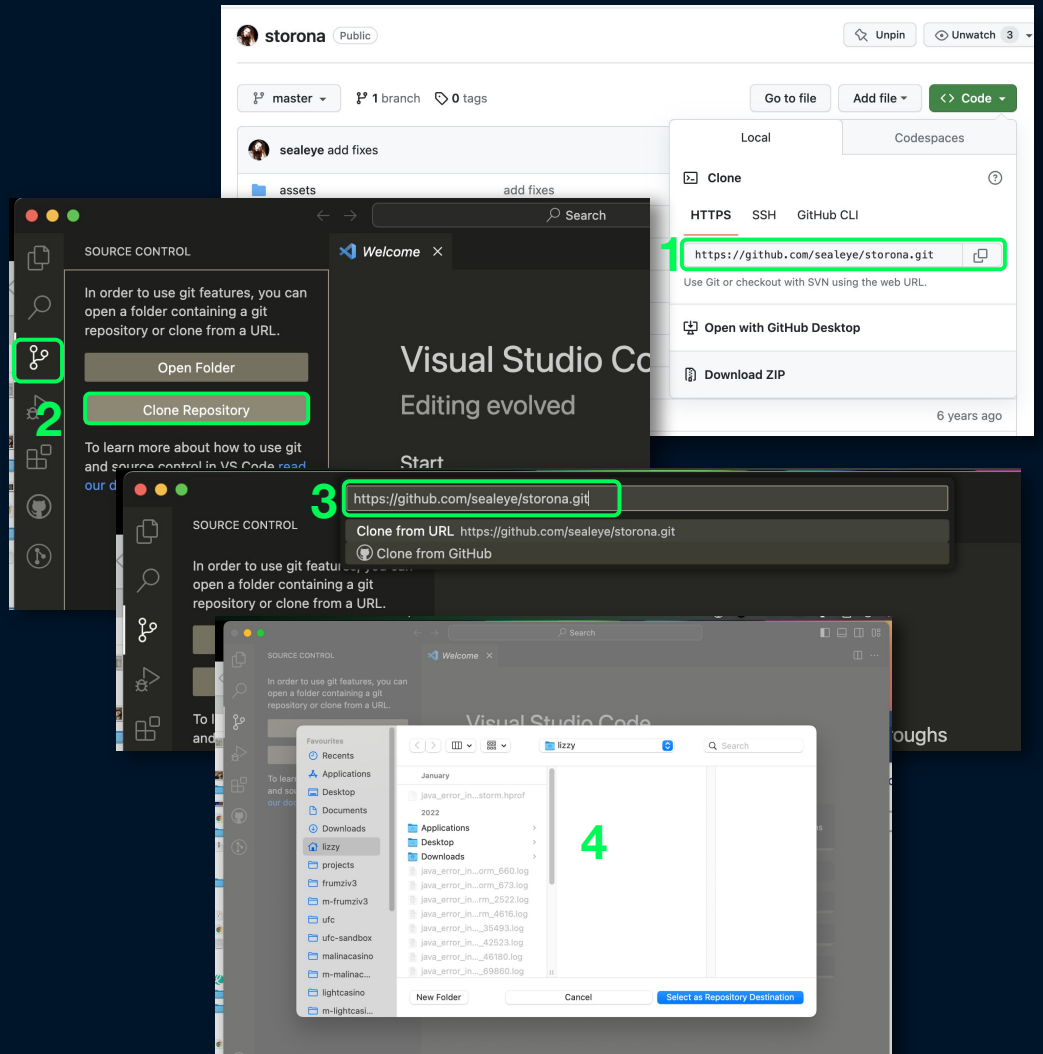
1. **Copy** the repository link.
2. On the main page of the code editor, select "**Clone Git Repository**".
3. **Paste** the link in the top field.
4. **Choose the folder** where the code will be stored on your computer.

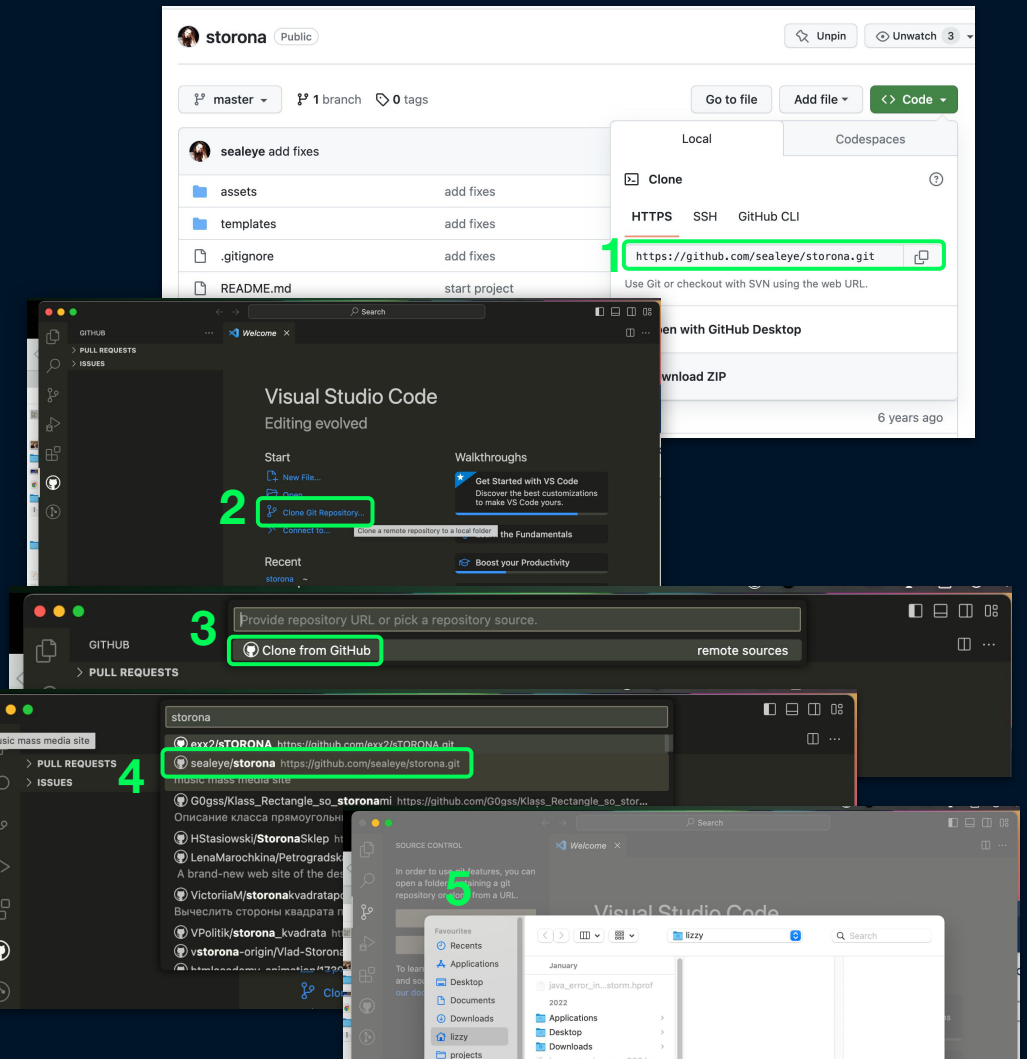
2.2

Clone

Using **VS Code**

1. **Copy** the repository link.
2. In VS Code, select "**Source Control**" on the right panel and click "**Clone Repository**".
3. **Paste** the link in the top field.
4. **Choose the folder** where the code will be stored on your computer.





2.3

Clone

Using **VS Code** and plugin **GitHub**

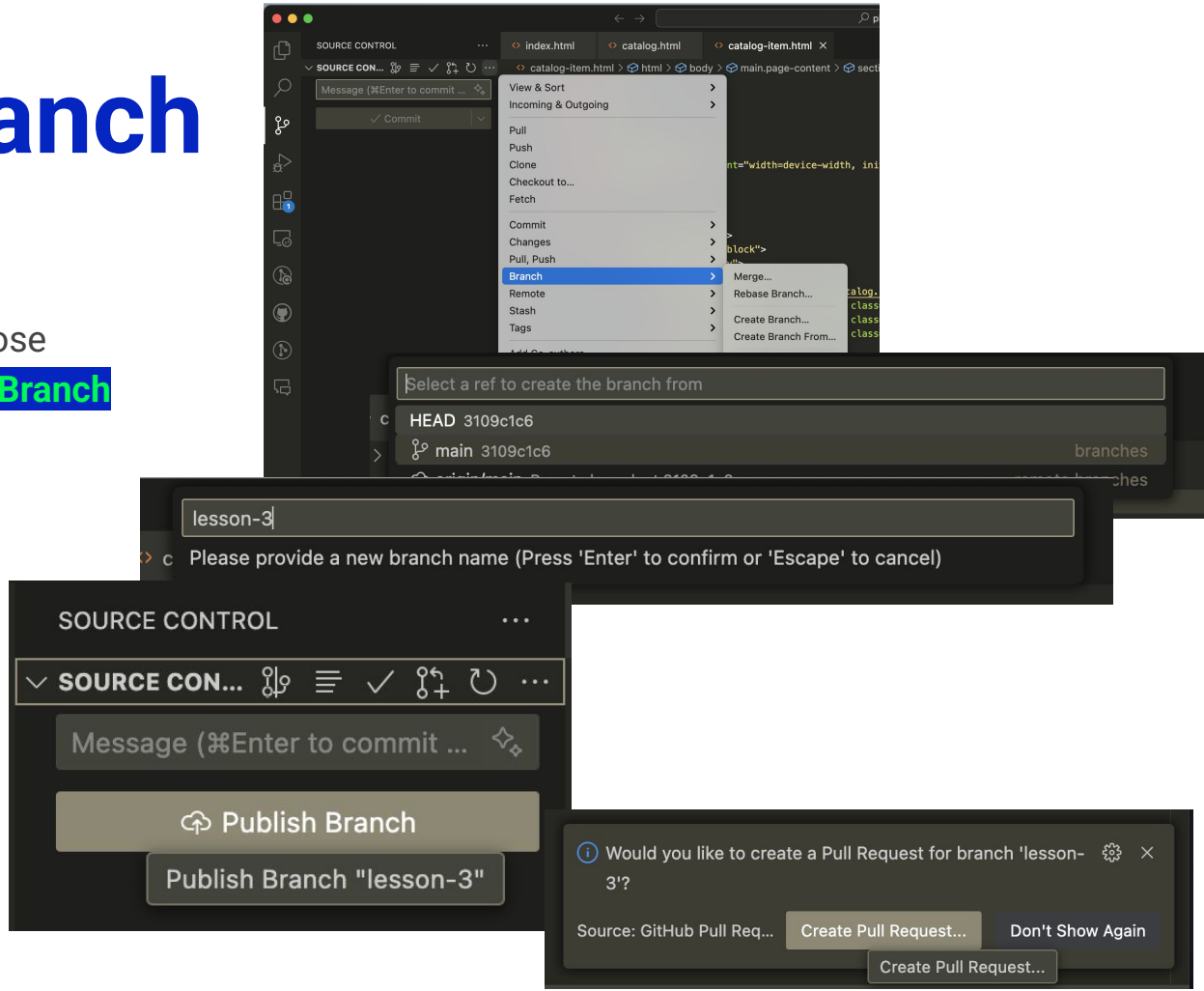
1. **Copy** the repository link.
2. On the main page of the code editor, select "**Clone Git Repository**".
3. Click "**Clone from GitHub**".
4. **Choose the project** to clone.
5. **Choose the folder** where the code will be stored on your computer.

3

Creating branch

Using **VS Code**

1. Click on **...** button and choose **Branch** option with **Create Branch From...**
2. Select **main** branch always
3. Type branch name **lesson-***number-of lesson*
4. Click **Publish Branch**
5. You can choose **Create Pull Request** button in the right bottom corner after publishing branch

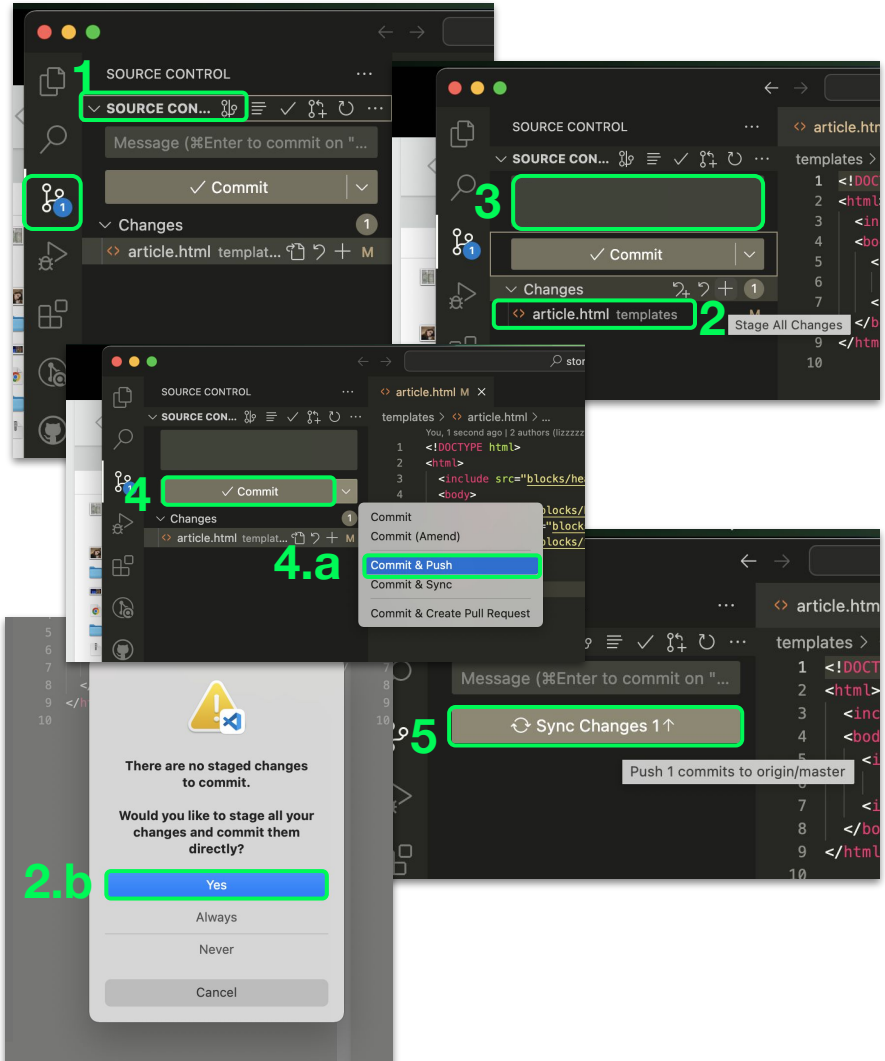


4

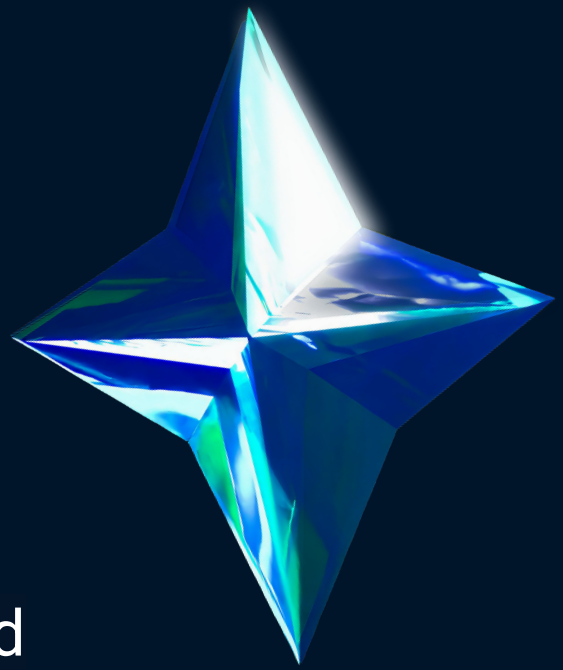
Modifying Code

1. In the right panel, select **Source Control** and open the first item – Source Control to see the changed files.
2. Select the changes to include in the commit.
   a. You can click the **+** button in the **Changes** section to select all changes.
   b. If you forget to select changes, a warning will appear, and you can safely click **Yes**.
3. Enter a message in the **Message** field (it should answer the question "What does this commit do?").
4. Click the **Commit** button.
   a. You can also select **Commit & Push**.
5. If you didn't select Commit & Push, click the **Sync Changes** button to perform the push action.

Check your repository on **GitHub.com** to ensure all changes have been applied.



TAKE CARE OF TEAMMATES



What or how to commit

- Pull before all new commits
- Coordinate with your co-workers
- Remember that the tools are line-based

Branch names

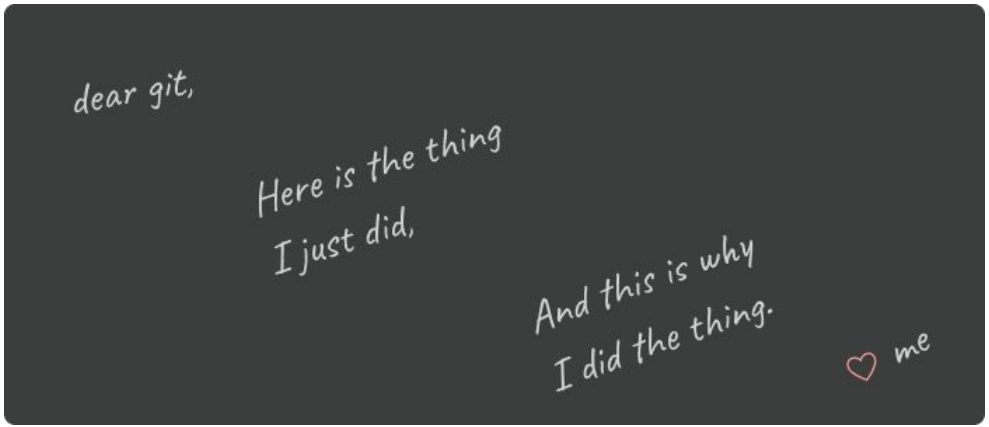
- Descriptive Names
  login, navbar-overflow
- Use Hyphens
  fix-login-issue
  fixLoginIssue or fix_login_issue
- Alphanumeric Lowercase Characters. Avoid punctuation, spaces, underscores, or any special characters whenever possible.
- Avoid Unnecessary Hyphens
  feat/new--login-
- Short and Effective

How to name commits

- Imperative Mood
  Fix bug #67
  Fixed bug #67

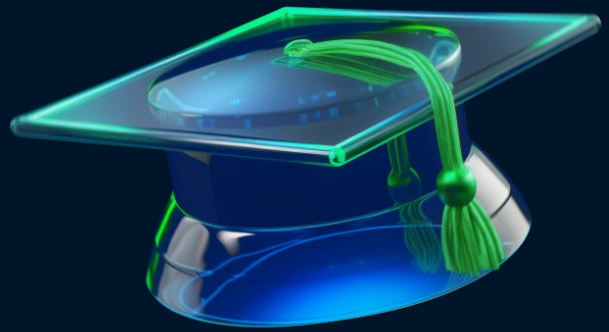
- Try to fit the subject line inside 50 characters.
  Avoid trailing period and unnecessary words/symbols.

- Capitalize the description



COMMIT RULES

- **Use a descriptive commit message**
  ✅ good: "fix validation in contact form", ❌ bad: "fix", "sdfsdf"
- **Use the imperative mood**
  ✅ good: "fix validation", ❌ bad: "fixed validation"
- **Make each commit a logical unit**
  don't commit more than described
- **Make atomic commits**
  logical units, the smallest possible size
- **Strive to use no more than 60 symbols**
  to be visible in GitHub UI



5.1

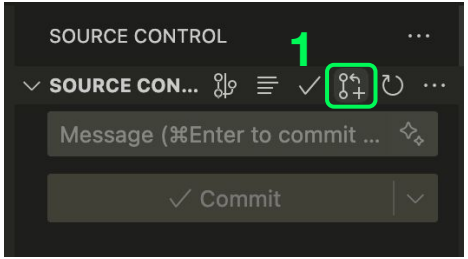
Creating pull request

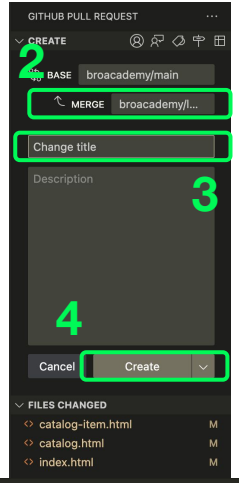
Using **VS code or VS code**

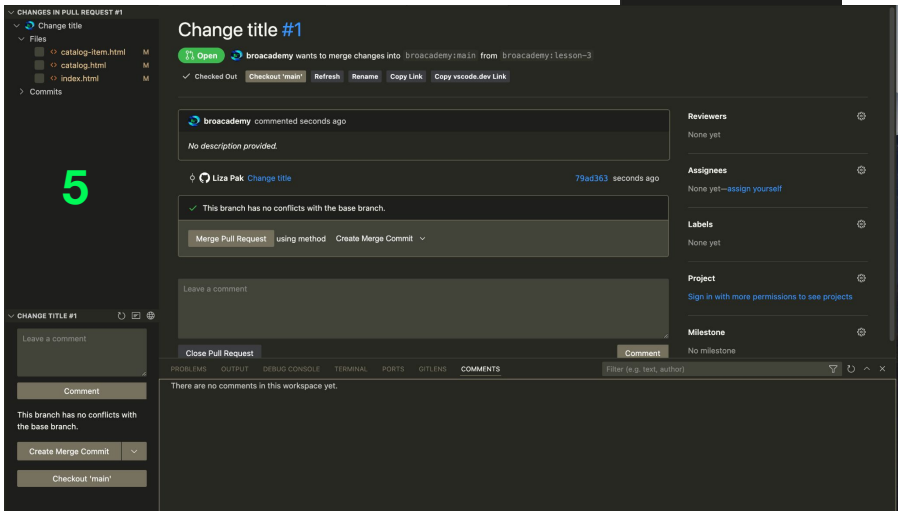
1. Open Source control tab and choose **Pull request** button

   If you don’t have changes, you can’t create PR

2. Check **branch name**
3. **Describe** the changes made in your commits
4. Click **Create**
5. See the result





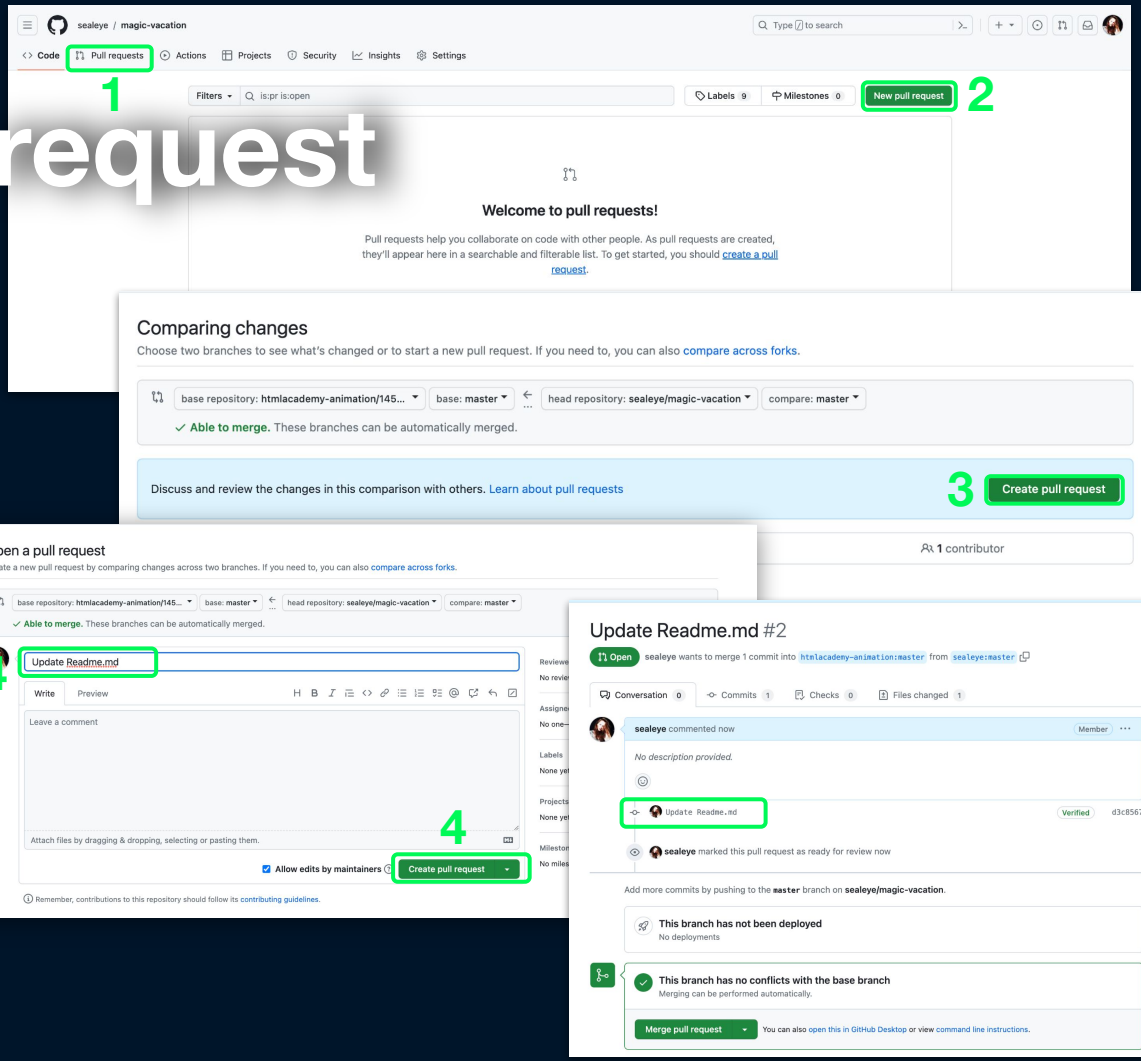


5.2

Creating pull request

Using **github.com**

1. Open the repository on **GitHub** and select the **Pull Requests** tab.
2. Click the **New pull request** button.
3. A panel with the commits to be included in the pull request will appear.
   Check branches names.
   Click the **Create pull request** button.
4. Describe the changes made in your commits and click **Create pull request**.
5. Review the created pull request.



README.md

– is a text file that introduces and explains a project. It contains information that is commonly required to understand what the project is about.

Why should I make it?

It's an easy way to answer questions that your audience will likely have regarding how to install and use your project and also how to collaborate with you.

Who should make it?

Anyone who is working on a programming project, especially if you want others to use it or contribute.

When should I make it?

Definitely before you show a project to other people or make it public. You might want to get into the habit of making it the first file you create in a new project.

Where should I put it?

In the top level directory of the project. This is where someone who is new to your project will start out. Code hosting services such as GitHub, Bitbucket, and GitLab will also look for your README and display it along with the list of files and directories in your project.

How should I make it?

While READMEs can be written in any text file format, the most common one that is used nowadays is Markdown.

Markdown rules

Markdown is a simple way to format text that looks great on any device.

It doesn't do anything fancy like change the font size, color, or type — just the essentials, using keyboard symbols you already know.

Italic	*Italic*
Bold	**Bold**
# Heading 1	**Heading 1**
## Heading 2	**Heading 2**
[Link](http://a.com)	Link
![Image](http://url/a.png)	
> Blockquote	▏Blockquote
* List * List * List	• List • List • List
`Inline code` with backticks	`Inline code` with backticks
`` # code block print '3 backticks or' print 'indent 4 spaces' ```	`# code block` `print '3 backticks or'` `print 'indent 4 spaces'`