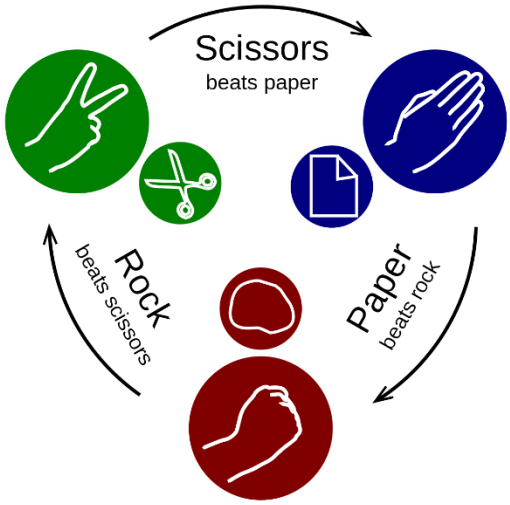
Practical Project: Rock – Paper – Scissors

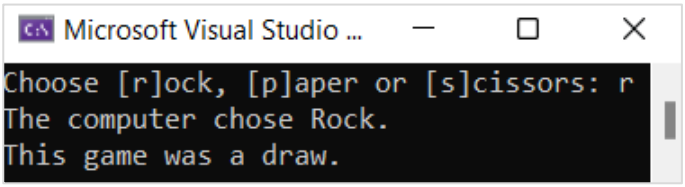
This is additional practical project and **it is not mandatory and it is not included in the final score**. The main purpose is to use gained knowledge in different type of problems and to improve your portfolio and GitHub skills.

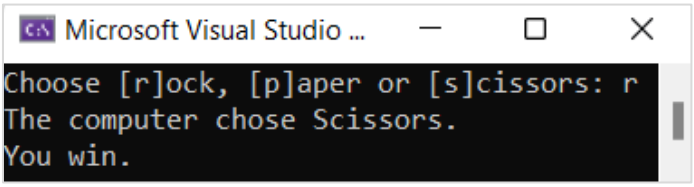


Rock - Paper - Scissors is a simple **two player game**, where you and your opponent (the computer) simultaneously choose one of the following three options: "**rock**", "**paper**" or "**scissors**". The rules are as follows:

- **Rock beats scissors** (the scissors get broken by the rock)
- **Scissors beats paper** (the paper get cut by the scissors)
- **Paper beats rock** (the paper covers the rock)

The **winner** is the player whose choice beats the choice of his opponent. If both players choose the same option (e.g. "paper"), the game outcome is "**draw**":



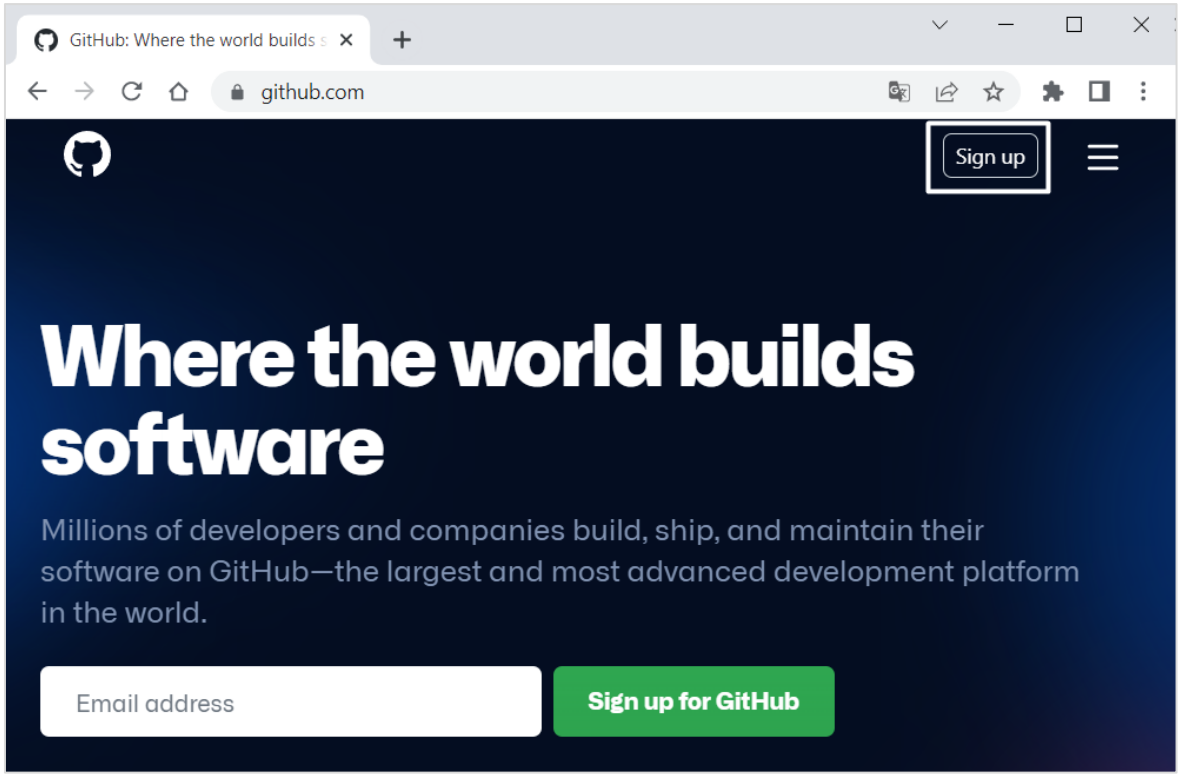


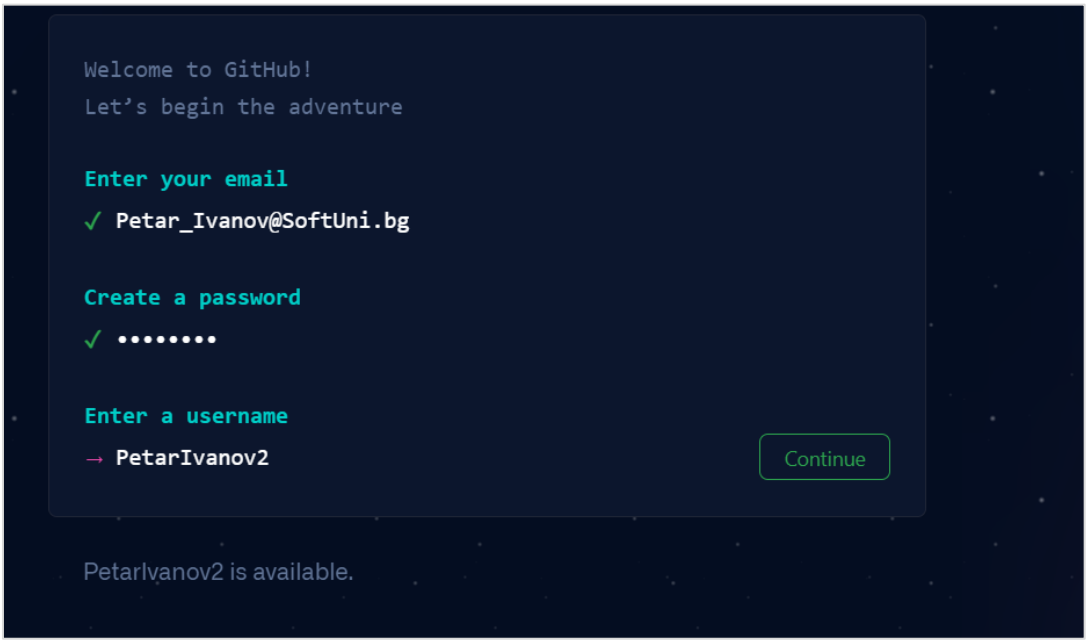
1. Create a GitHub Profile and Repo

Everyone should have a GitHub developer profile. First, we should **create our profile in GitHub**.

Register a GitHub Profile

Register for a free **developer account at GitHub** here: http://github.com with an email and a username:

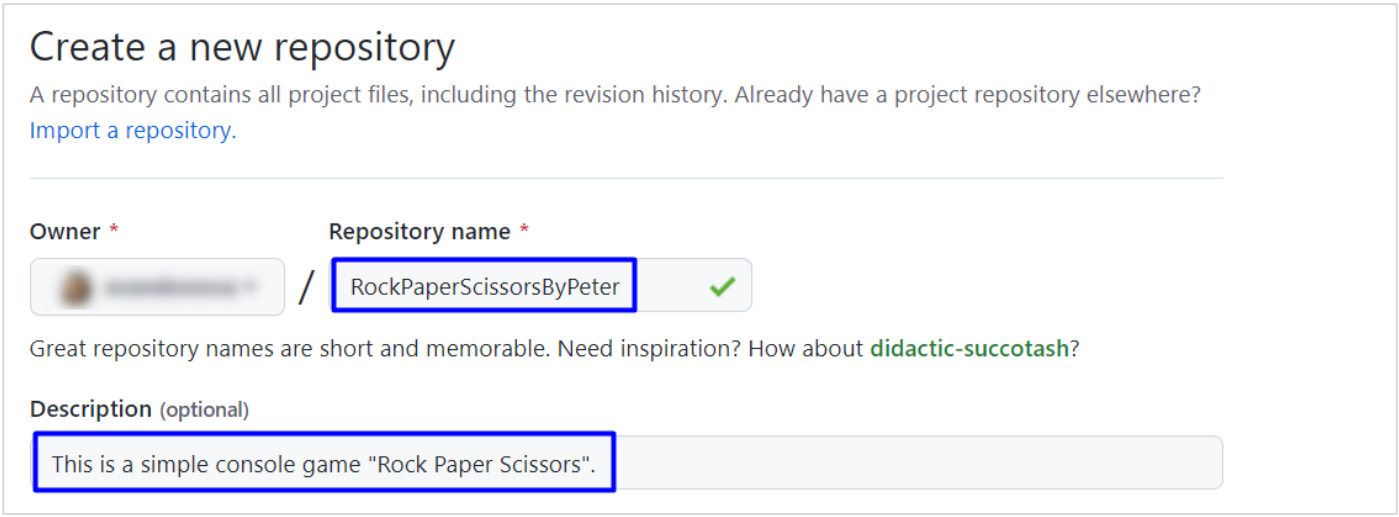




When you are ready, it is time to **create your first repository**. A **repository** contains **all of your project's files** and each file's revision history. You can discuss and manage your project's work within the repository.

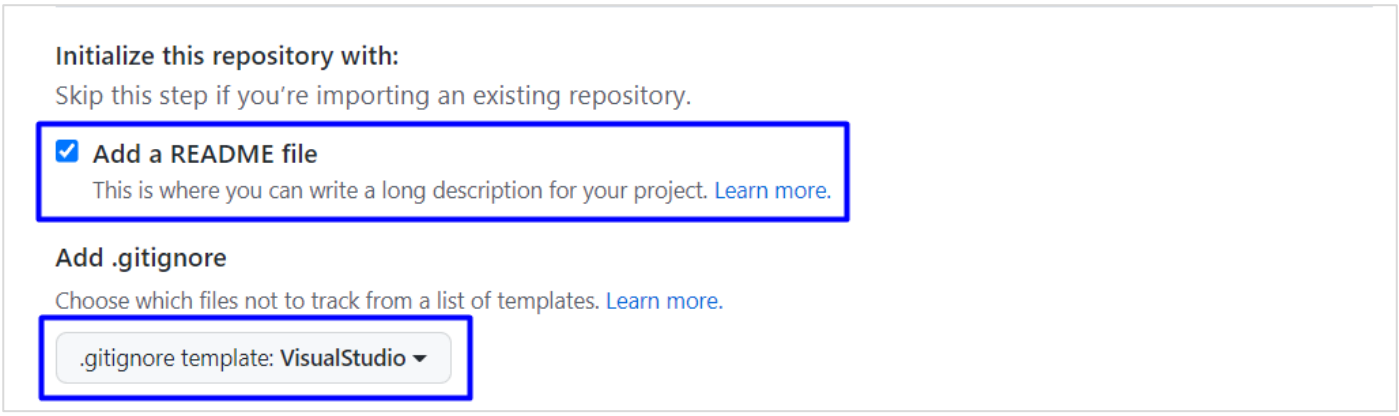
Create a GitHub Repo

Create a **new repository** from: https://github.com/new. Choose a **meaningful name**, e. g. "**`RockPaperScissorsByUsername`**" add a **short description** and make your repo **public**:



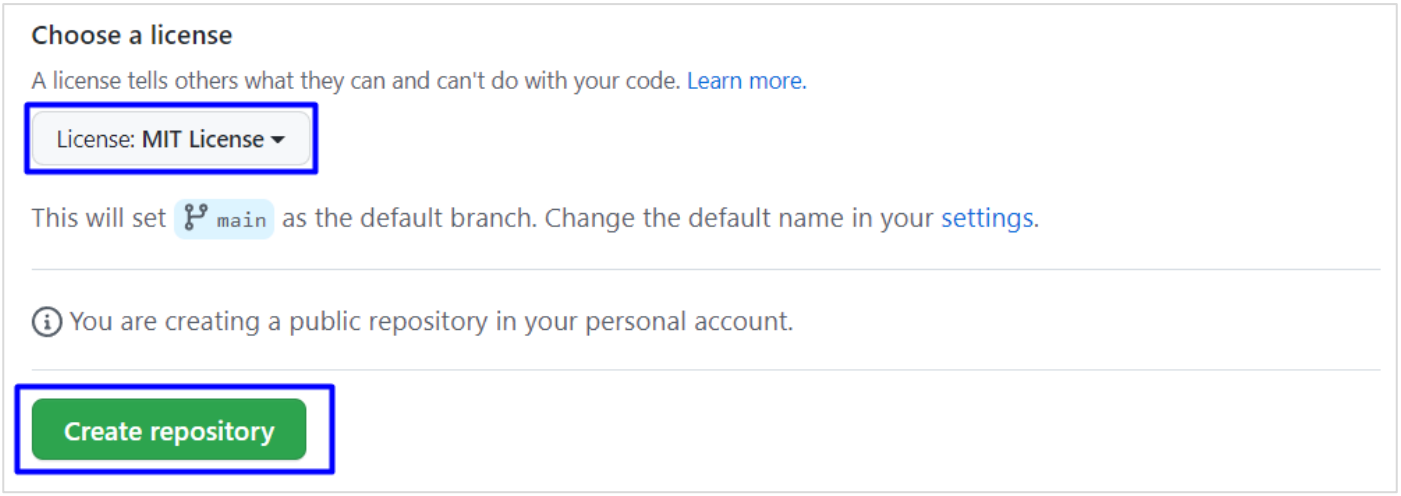
⚠	Please choose **your own original and unique name** for your project! Your GitHub profile should be **unique**, not the same as your colleagues'. You can follow this tutorial, but you can also **make changes** and **implement your project differently** from your colleagues.

Also, **add a `README.md`** file and **`.gitignore` for Visual Studio**, as shown below:

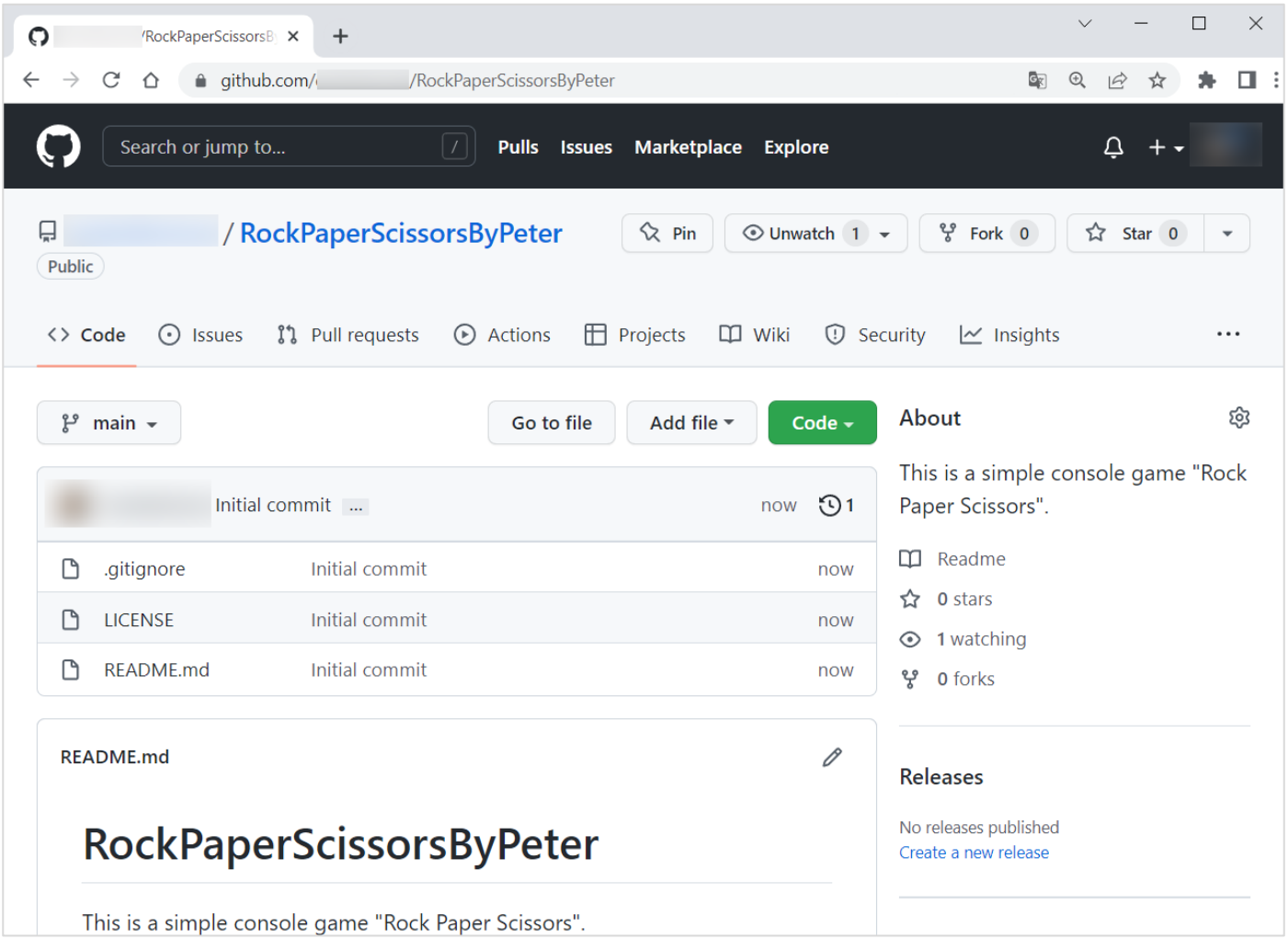


In Git projects the **`.gitignore` file** specifies which files from your repo are not part of the source code and should be ignored (not uploaded in the GitHub repo). Typically in GitHub, we upload in the repo **only the source code** and we don't upload the compiled binaries and temp files.

Finally, **change the license** to "**`MIT`**" (which is the most widely used open source license) or another license of choice, and click on the **`[Create]` button** to **create your repository**:



Now your **repository is created** and looks like this:



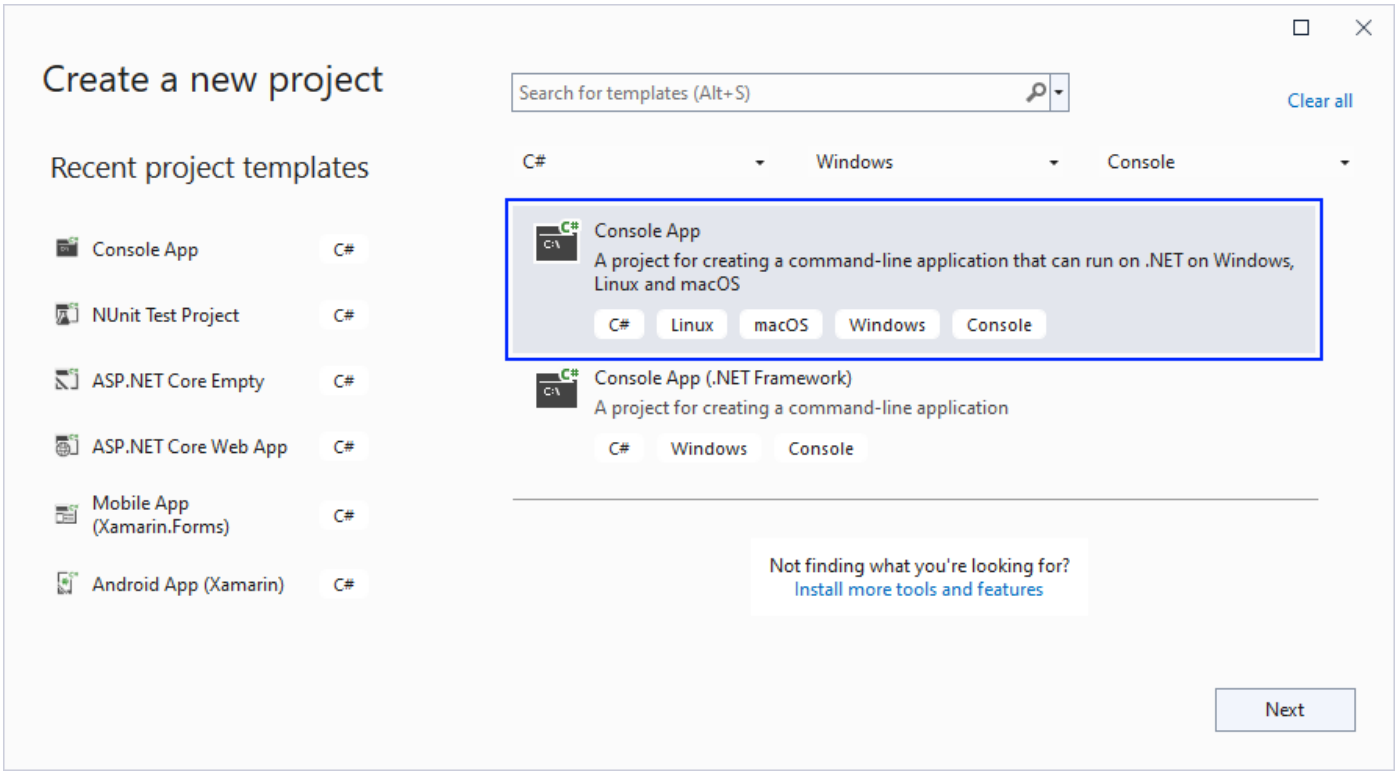
Now let's see how to **write the code** of our game.

2. Write the Game's Code

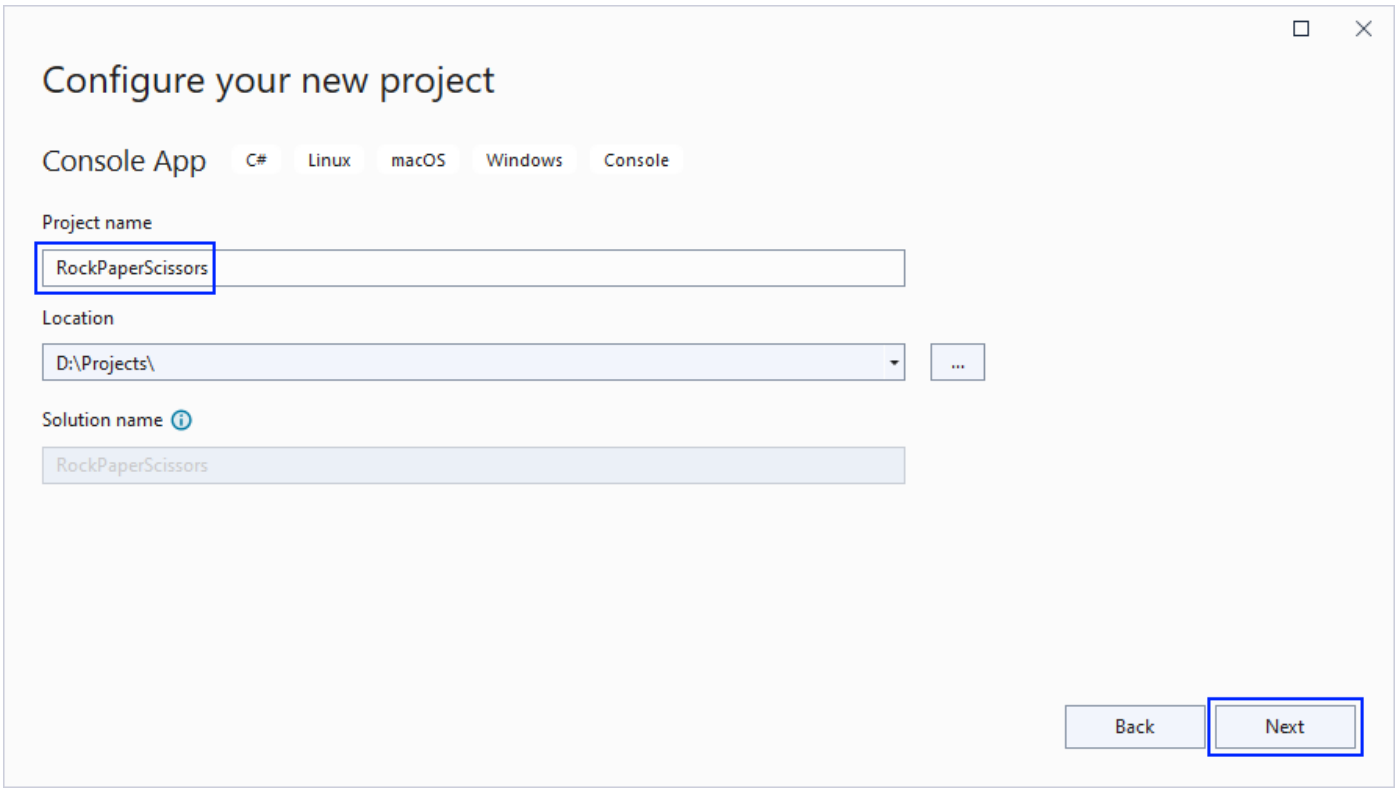
Let's create the game and play with it.

Create a Visual Studio Project

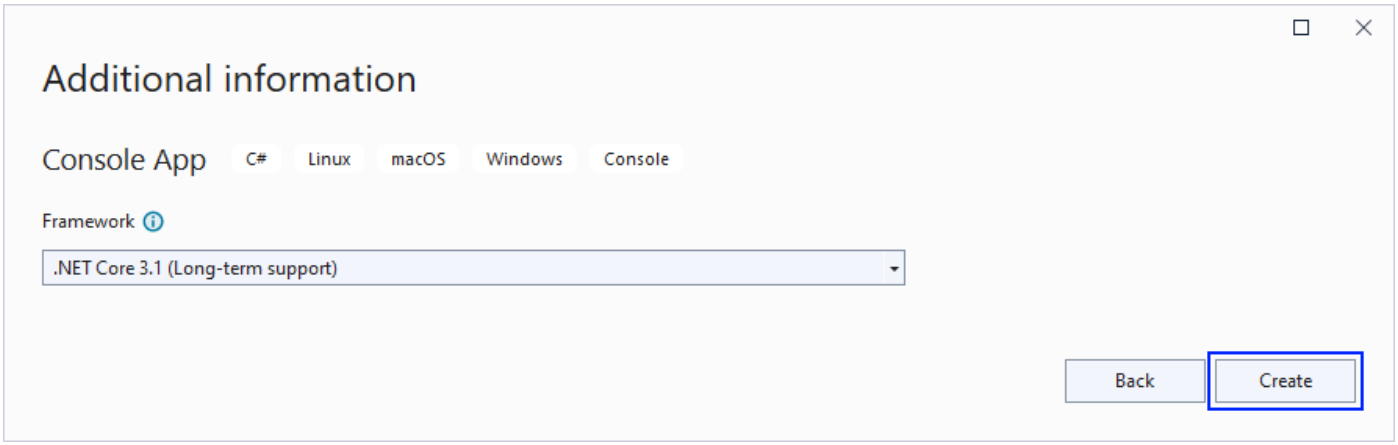
First, we should **start Visual Studio** and **create a new C# console application**:



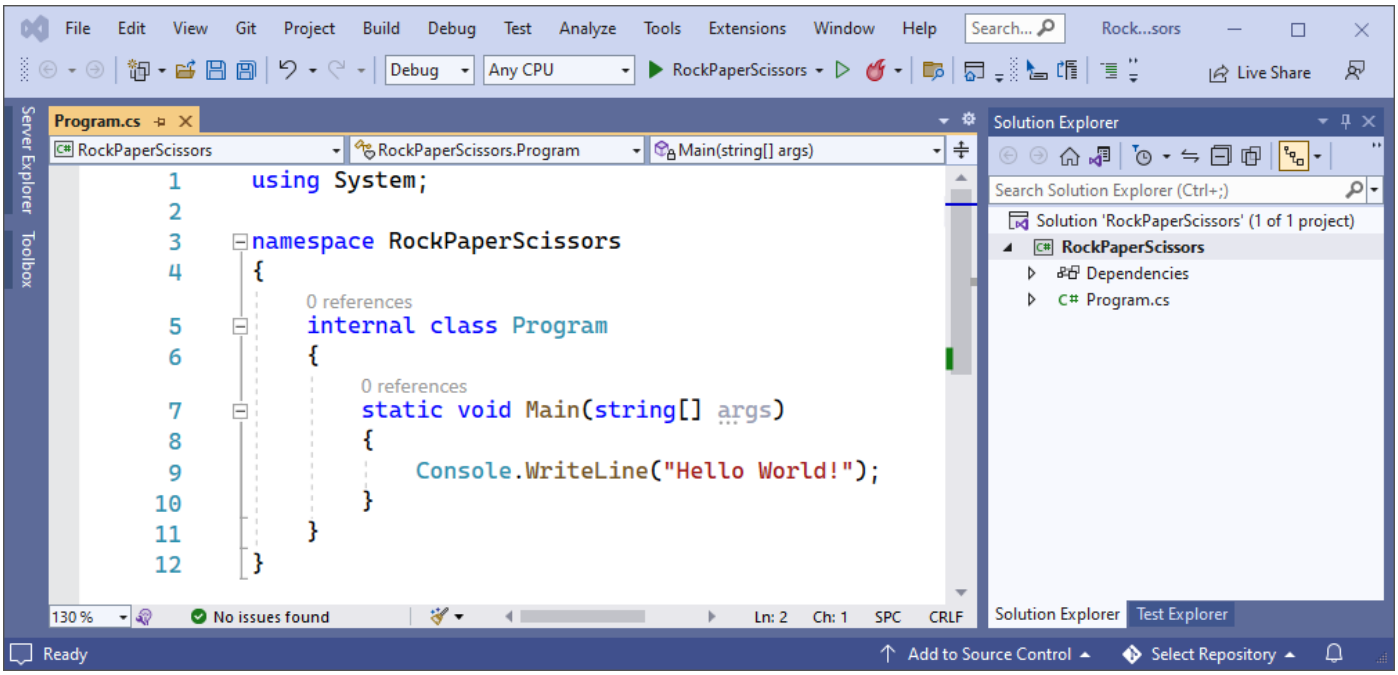
Then, **choose an appropriate name** and a **place to save the project**. You should also **check** the [**`Place solution and project in the same directory`**] box, so that we do not have an additional folder for our files. Then, click on [**`Next`**]:



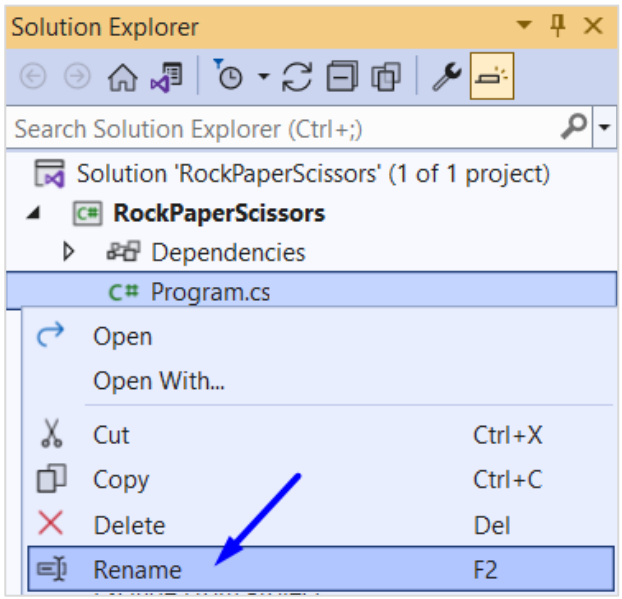
On the next screen, choose [**`.NET 3.1 (Long-term support)`**] and click [**Create**]:

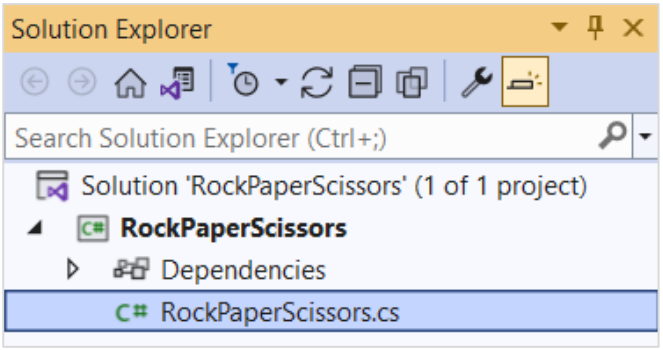


Our **project should be created** and should look like this:



Before we continue, let's change the name of our main class – **`Program.cs`** to something more **meaningful**. Do it like this:





Implement the Game Logic

Read Player's Move

Now let's start working on our code.

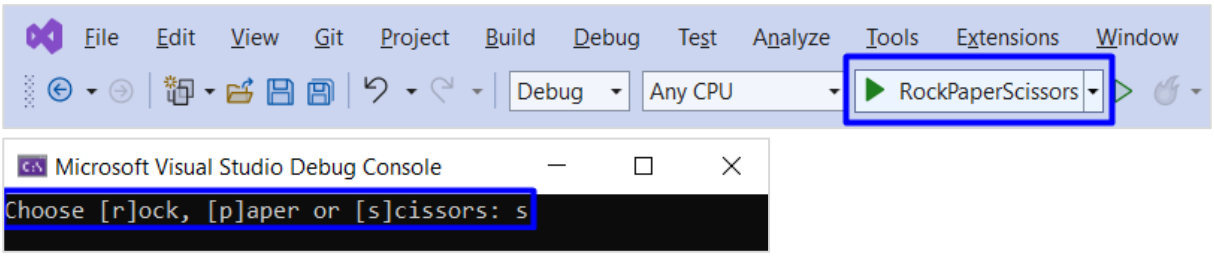
Create **three constants** for our "**Rock**", "**Paper**" and "**Scissors**", which we will use later. **Constants** are values which **do not change** for the life of the program. They should look like this:

```
const string Rock = "Rock";
const string Paper = "Paper";
const string Scissors = "Scissors";
```

Next, write on the console what options **("rock", "paper", "scissors"**) the player can choose from and read his **input data**. You already know how to do this:

```
("Choose [r]ock, [p]aper or [s]cissors: ");
string playerMove =
```

Now let's run the **app** in the console and check whether our current code **works** properly:



We can see that we have our text **written** on the console and we can also **write**.

Match Player's Move with Possible Options

Now it is time to turn the user input into one of our **player's move options**. To do this, create an **`if-else`** statement with the **possible moves** and change the variable value with our **`constants`**.

First, if the user has entered **"r"** or **"rock"**, then they chose **"Rock"**. Write it like this:

```
if (playerMove == "r" || playerMove == "rock")
{
    playerMove = Rock;
}
```

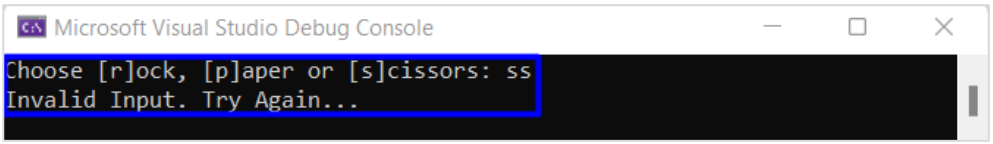
And if they entered **"p"** or **"s"**, then they choose **"paper"** or **"scissors"** accordingly. Write the **`else-if`** statements by yourself:

```
else if (playerMove == "p" || playerMove == "paper")
{
    playerMove = Paper;
}
else if (playerMove == "s" || playerMove == "scissors")
{
    playerMove = Scissors;
}
```

Now we should cover the case, in which the user enters an **invalid value**. To do this, use **`else`** and **print** a message on the console and **stop the program execution**:

```
else
{
    Console.WriteLine("Invalid Input. Try Again...");
    return;
}
```

Now let's **run** the app in the **console** and check whether our current code **works properly**, at the moment we have **logic** only for the **incorrect input** so the results should be as follow:



Choose Computer's Move

Then, **create a variable of type `Random`** that will help us **choose a random number** using the **`Next()`** method. We will use this **number** so that the computer can randomly select from "**`rock`**", "**`paper`**" or "**`scissors`**":

```
Random random = new Random();
int computerRandomNumber = random.Next(1, 4);
```

A little more information about `Random`:

.NET Core provides thousands of ready-to-use classes that are packaged into namespaces like the already known **`System`**. The **`System`** namespace contains fundamental classes, one of which is the **`Random`** class. It provides functionality to generate random numbers in C#. We will learn more about the **`Random`** class in the Objects and Classes lesson, but let's take a quick overall view of this class.

The line with code below creates a new object, which is an instance of the Random class. In this object will store the randomly generated number that we have to guess.

```
Random randomNumber = new Random();
```

The following code returns a random number, using the **`Next()`** method. This is a method, provided by the **`Random`** class. By writing "**`1, 4`**" in the brackets, we indicate to the method that we want our randomly generated number to be in the range between **1** and **3**. You should note that the lower bound is inclusive and the upper bound is exclusive, that's why we have **4** as the second parameter of the **`Next()`** method.

```
int computerRandomNumber = random.Next(1, 4);
```

You can read more about the **`Random`** class here https://docs.microsoft.com/en-us/dotnet/api/system.random.next?view=net-6.0.

We will need a **variable of type `string`** to keep our **computer's move**:

```
string computerMove = "";
```

Choose the computer's **random move**, to make this happen use the **conditional statements `switch-case`** or **`else-if`**. Also check the **input of the player**, e. g.:

```
switch (computerRandomNumber)
{
    case 1:
        computerMove = Rock;
        break;
    case 2:
        computerMove = Paper;
        break;
    case 3:
        computerMove = Scissors;
        break;
}
```

Think about how you can complete these **conditional statements**.

Check and Write the Result

Write to the console what is the **random** selection of the computer. e. g. "**The computer chose {computerMove}.**". Now we need to **compare** the choice of the **player** and the **computer**, again using **conditional statements**.

```
Console.WriteLine($"The computer chose {computerMove}.");

if ((playerMove == Rock && computerMove == Scissors) ||
    (playerMove == Paper && computerMove == Rock) ||
    (playerMove == Scissors && computerMove == Paper))
{
    Console.WriteLine("You win.");
}
```

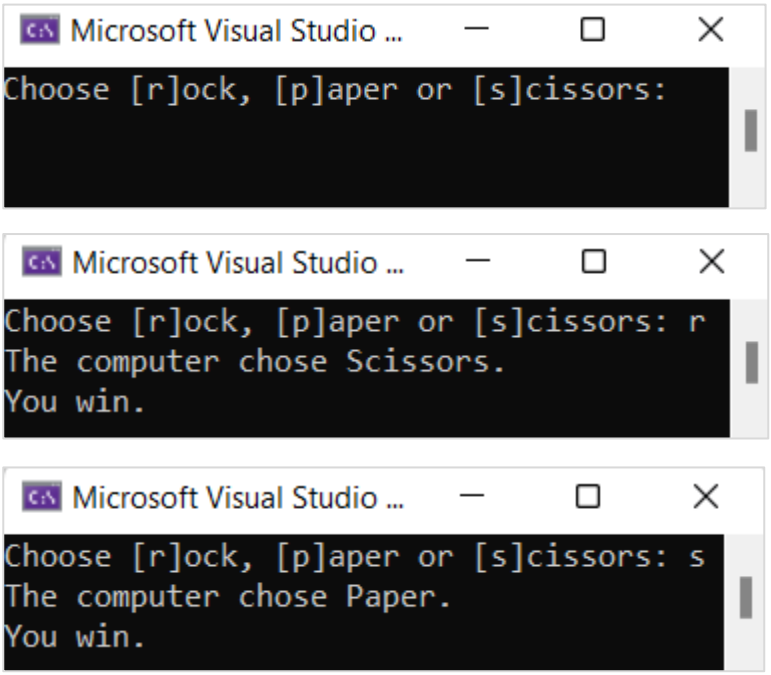
You can use this table for the **possible moves**:

You	Computer	Outcome
rock	rock	Draw
rock	paper	You lose
rock	scissors	You win
paper	rock	You win
paper	paper	Drow
paper	scissors	You lose
scissors	rock	You lose
scissors	paper	You win
scissors	scissors	drow

Consider all the cases where the player **loses** or the result between them is **equal** and write down the **conditional statements**. That's all it takes for the **game to work**.

```
else if ((playerMove == Rock && computerMove == Paper) ||
        (playerMove == Paper && computerMove == Scissors) ||
        (playerMove == Scissors && computerMove == Rock))
{
    Console.WriteLine("You lose.");
}
else
{
    Console.WriteLine("This game was a draw.");
}
```

After you run it, the game should look like this:



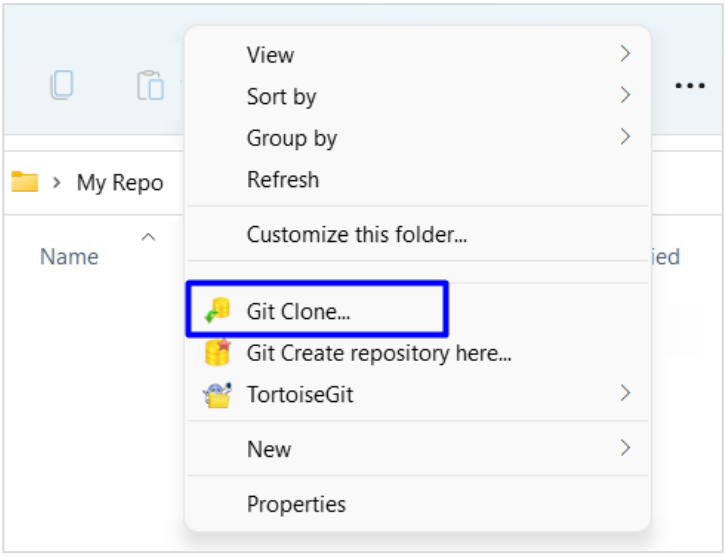
3. Upload Your Project to GitHub

Now we want to deploy our project to **GitHub** so the other developers can see it, and if they want to test it, they can clone it and try it them self on their personal machine. You have **two options**, choose one and follow the steps.

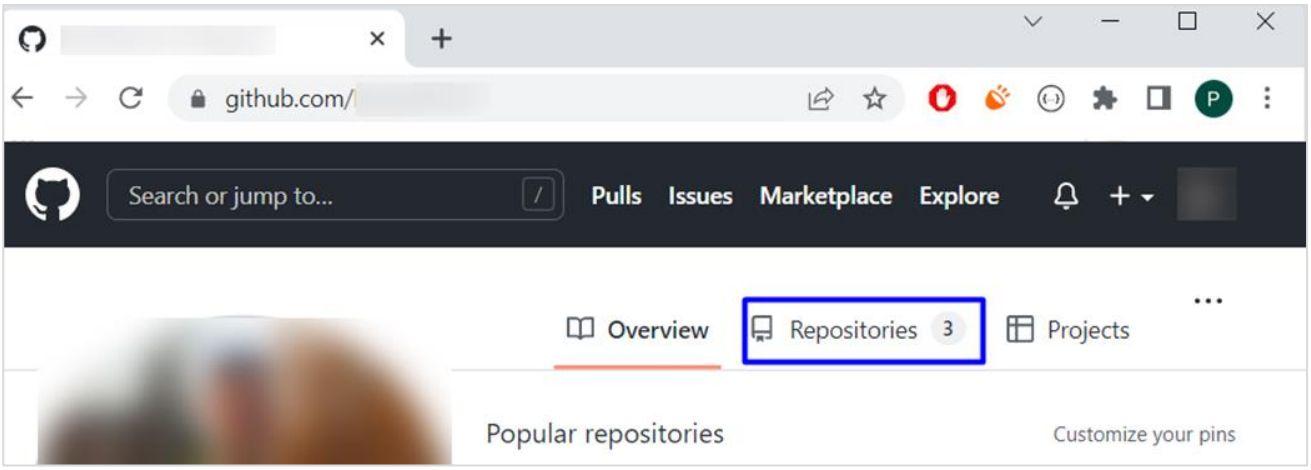
Use TortoiseGit (Option 1)

If you don't have **`TortoiseGit`** on your computer, first download and install it from https://tortoisegit.org.

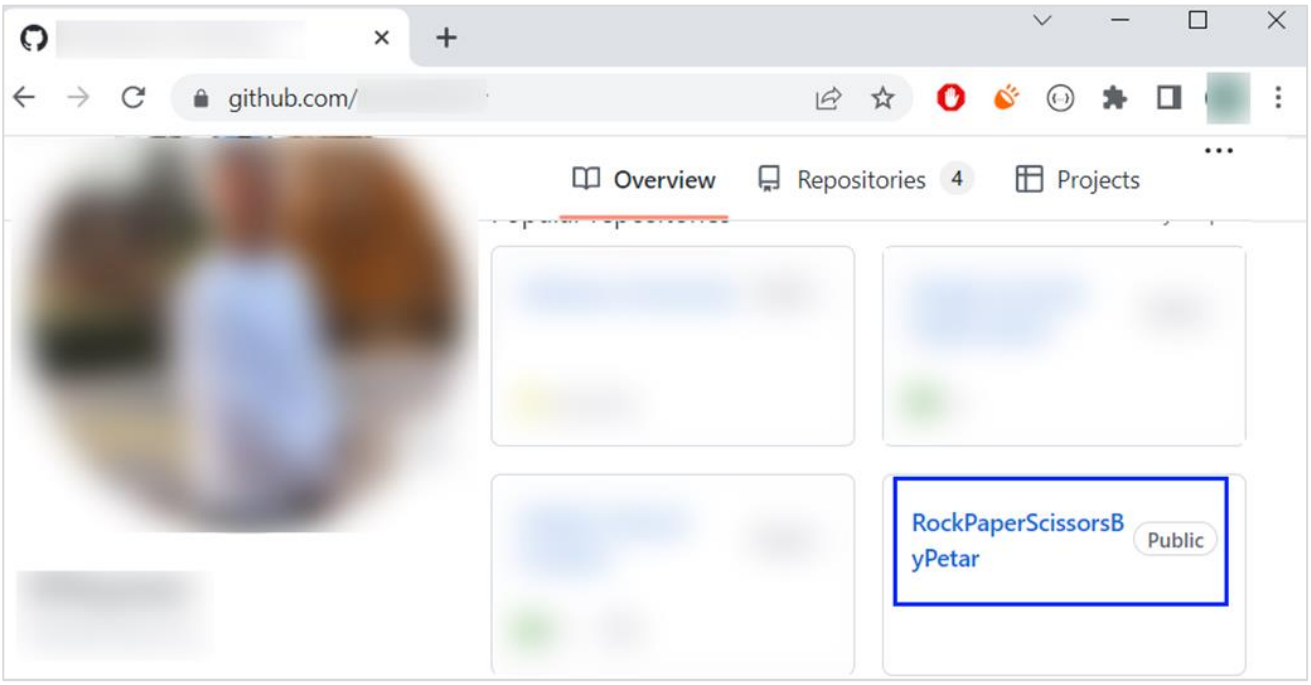
Use Git **clone** for cloning with **`TortoiseGit`**. Go in the desired directory, **right-click** on blank space anywhere in the folder and click **`[Git Clone]`**.



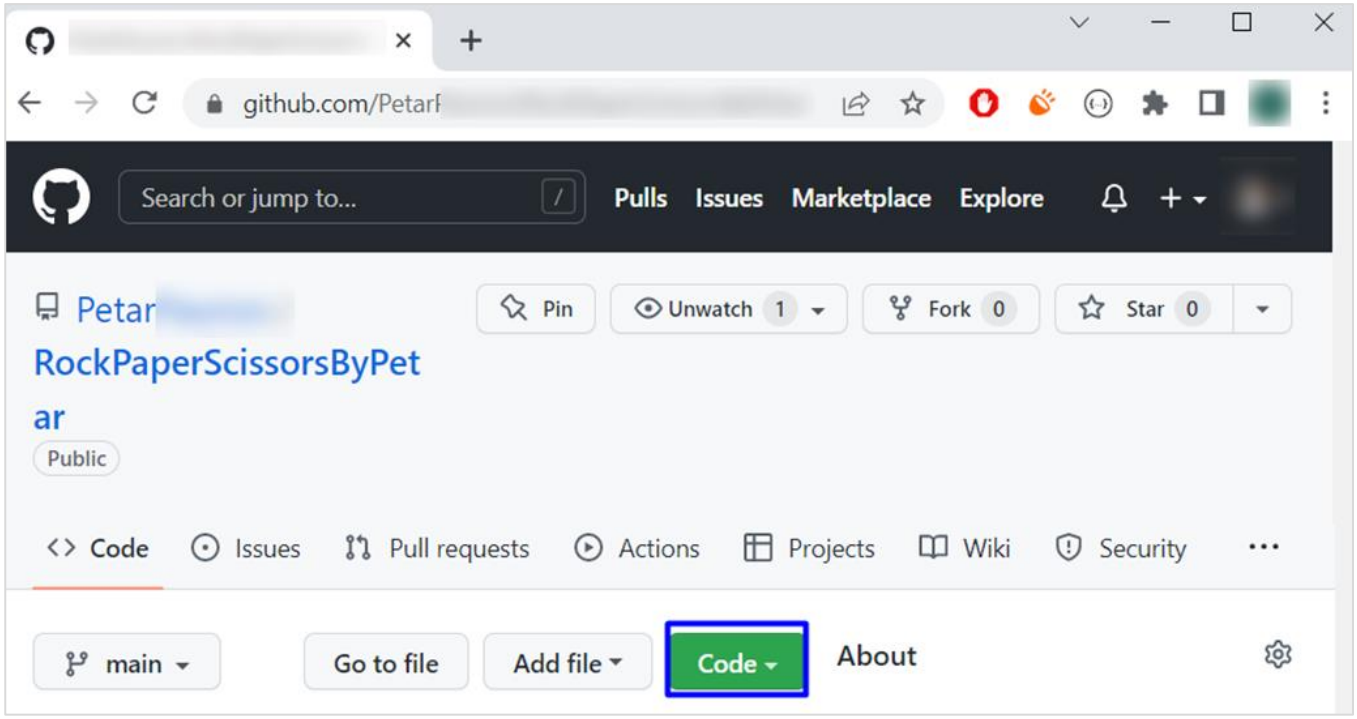
Now we will go to our **GitHub** profile, open our newly created **repository** and copy the **repo URL**. First click on **`[Repositories]`**:

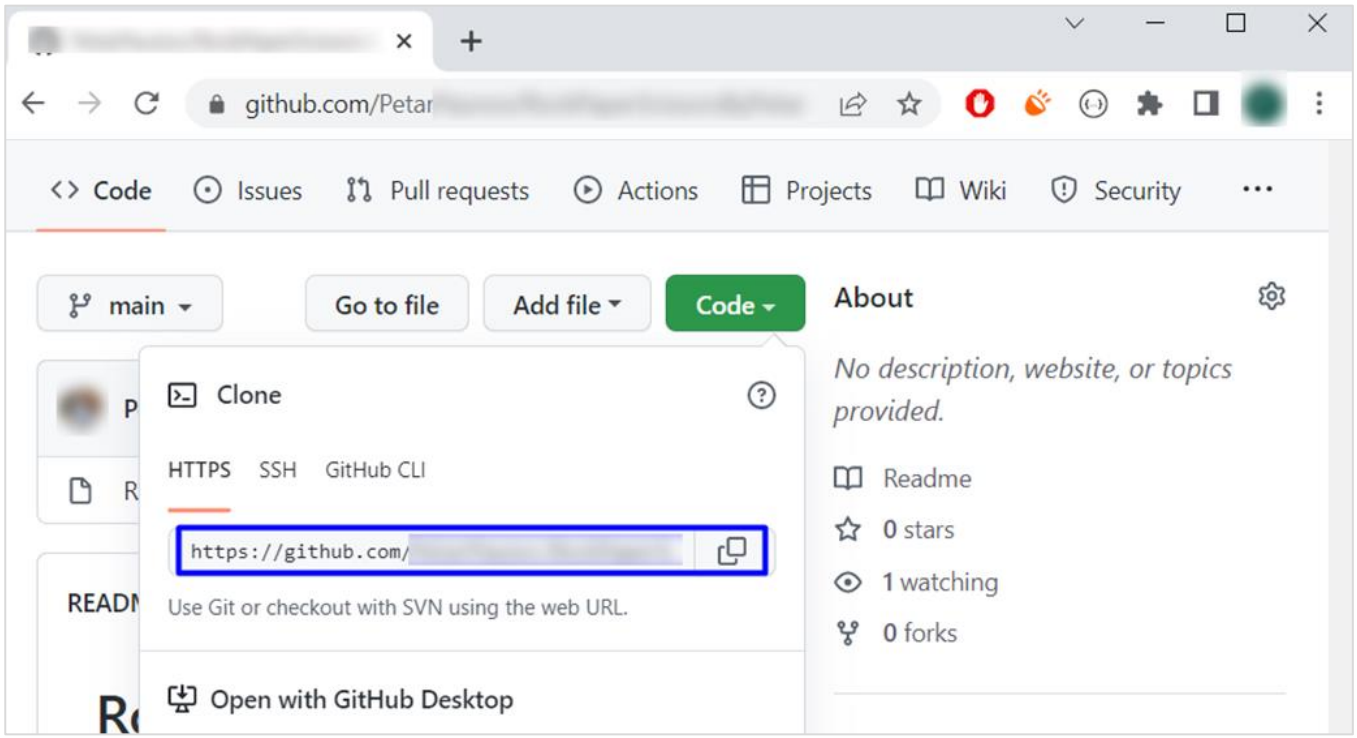


Then choose your project repo:

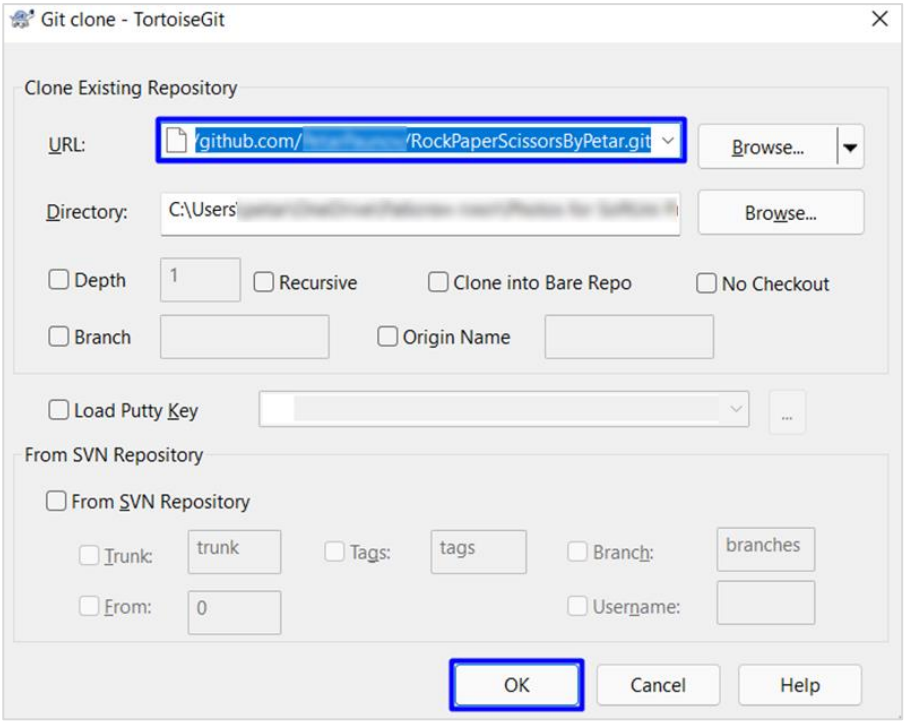


Now click on **`[Code]`** and copy the **URL** of the **repository**.

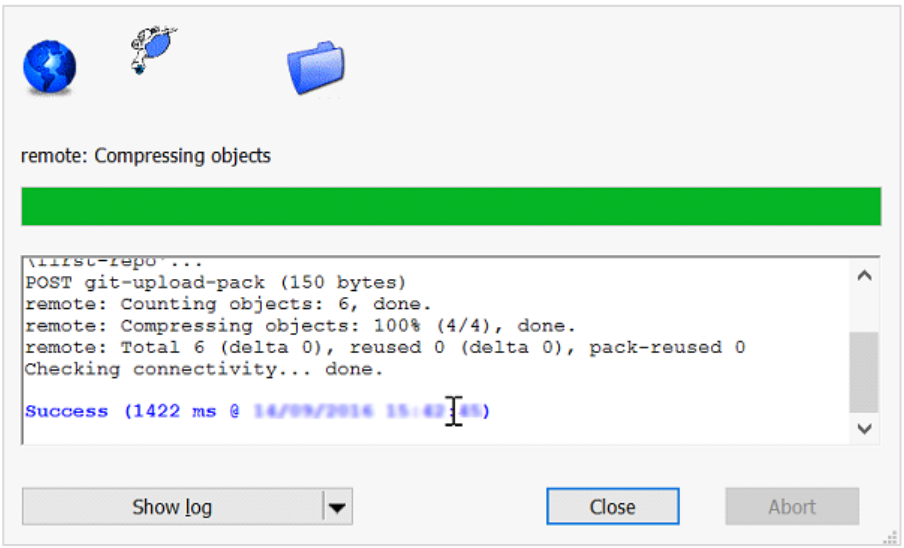




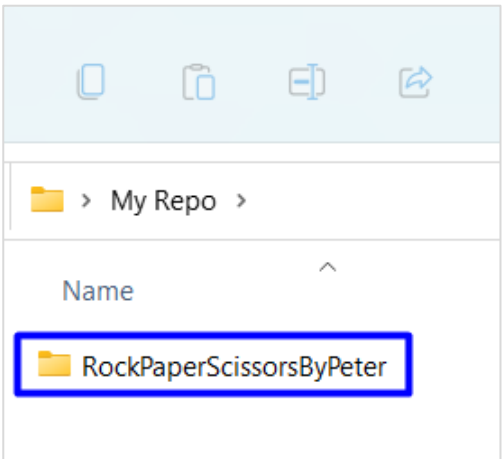
Now we go back to **TortoiseGit** and paste the **URL** and click **[OK]**:



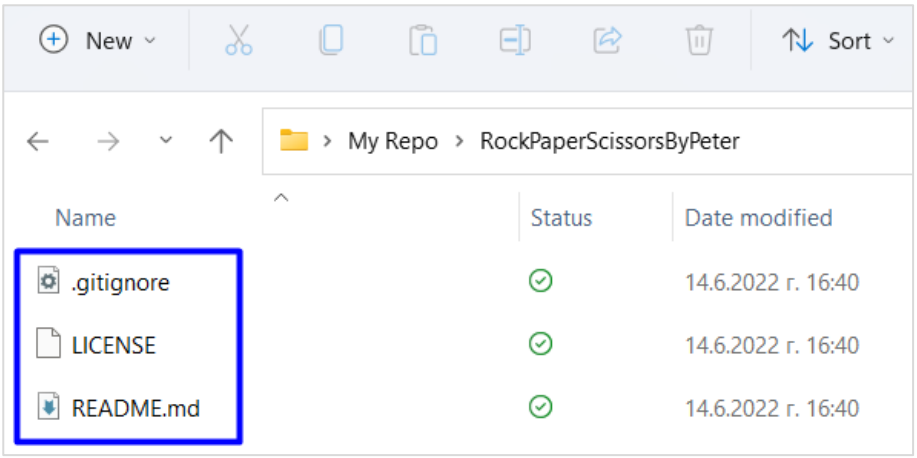
The results should be something like this:



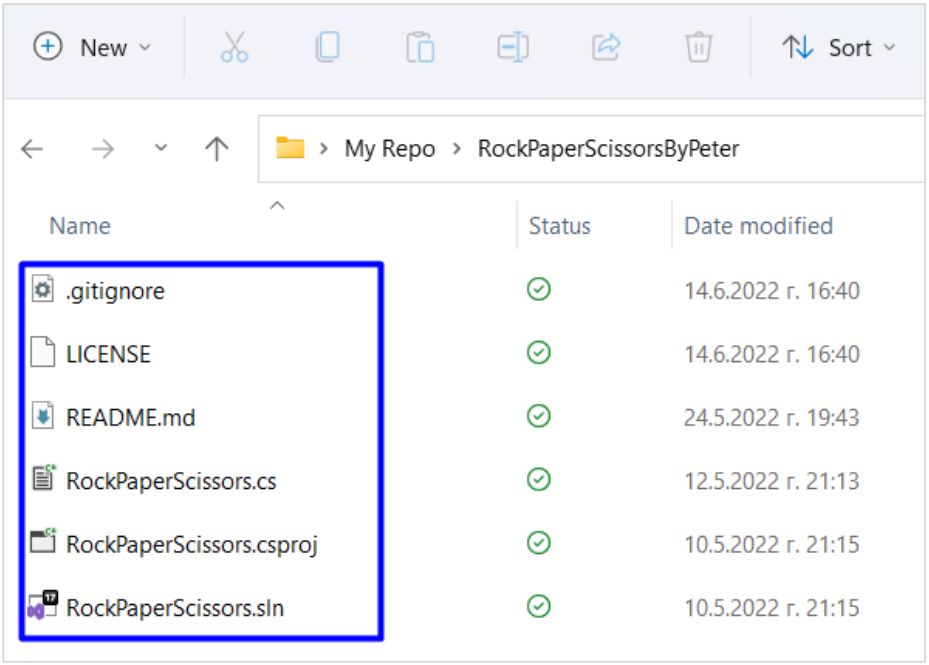
Your files from your GitHub repo will be downloaded to a **sub-folder** called as your project in GitHub, "**RockPaperScissorsByPeter**" in our case.



When we open the sub-folder, holding the cloned **repository**, it should look like this:

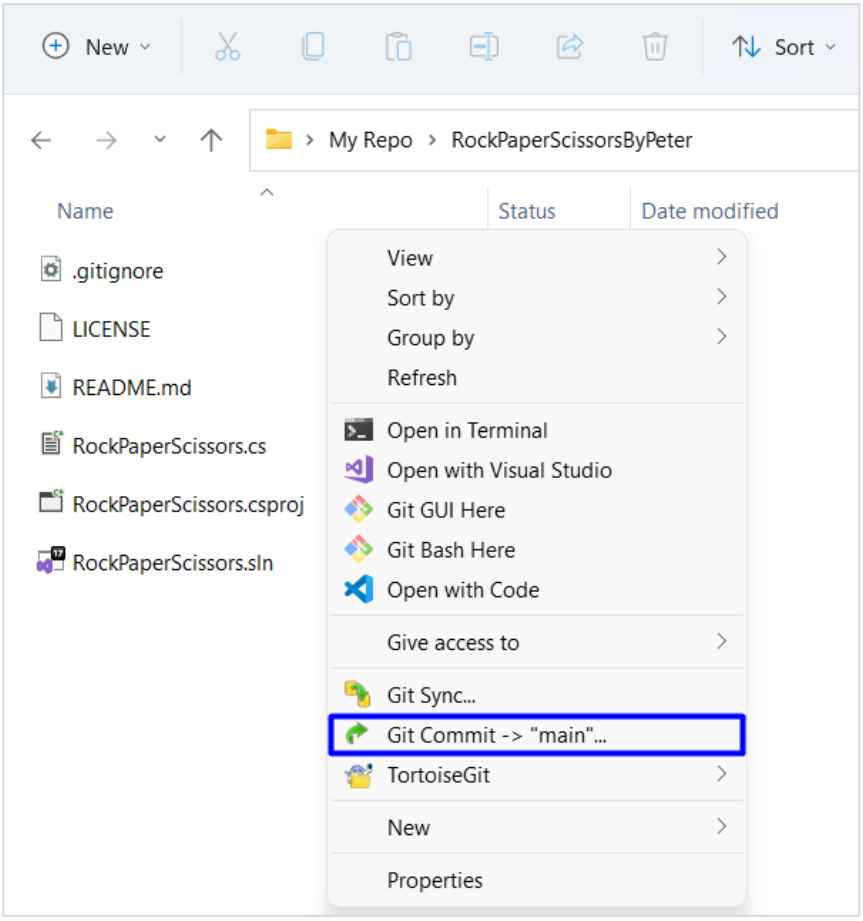


Next thing to do is to **add** the **project** into our **cloned repository**. You can move your C# source code files from your old project folder to your new repo folder. You can use "**Cut & Paste**". It should look like this:

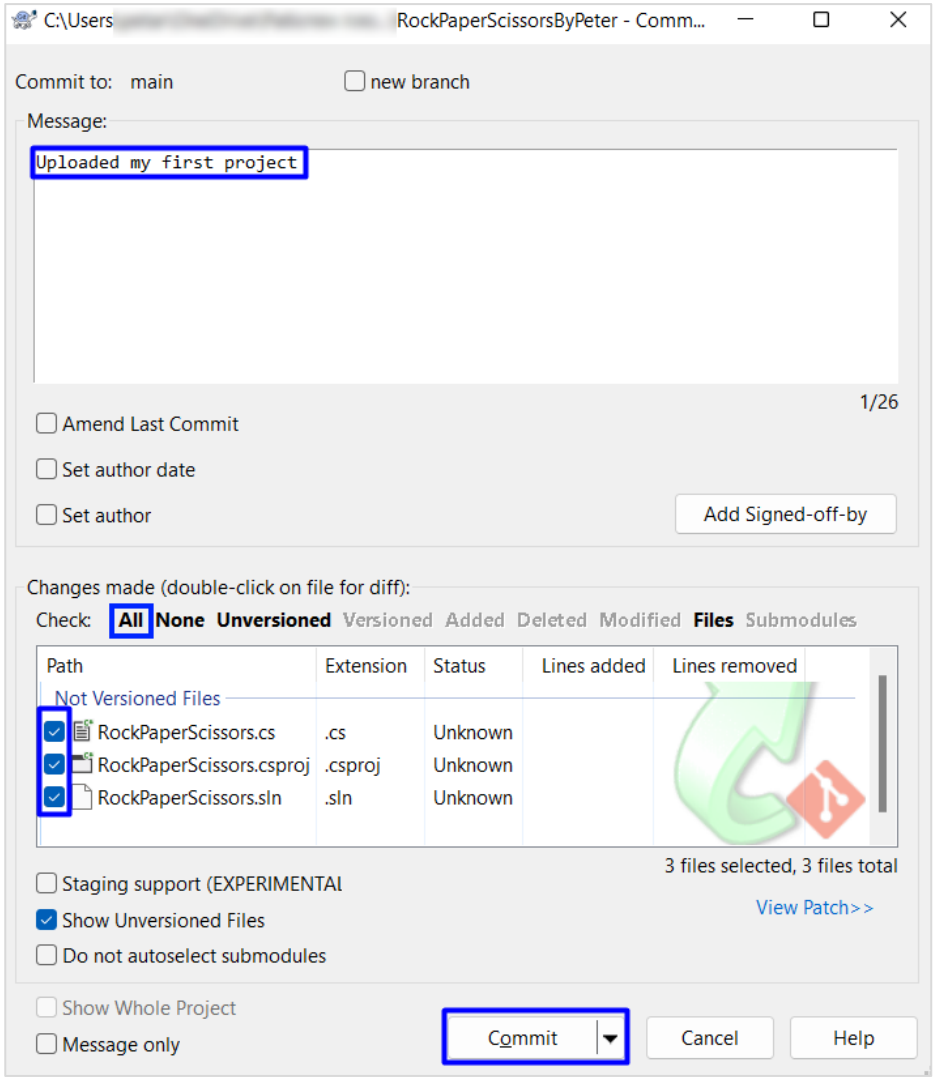


Now to **upload** our changes from our working project folder to GitHub.

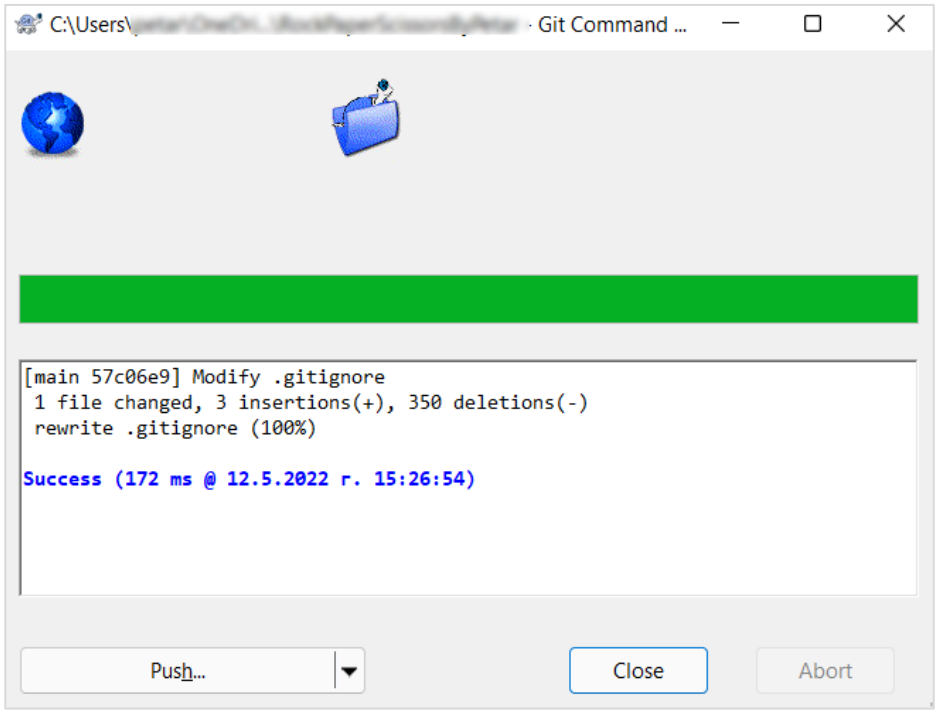
We can use TortoiseGit's **`[Git Commit…]`**. Go to your project's folder, **right-click** on blank space anywhere in the folder and click **`[Git Commit -> "main"…]`**.



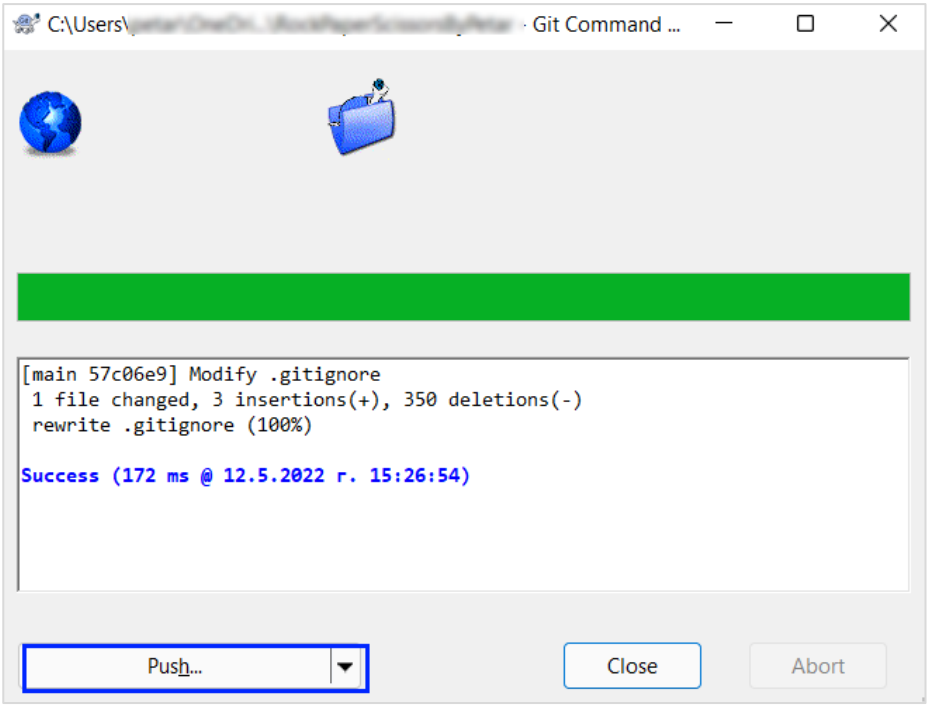
Add an **appropriate** message and click **[Add]** so you don't miss any files, finally click **[Commit]**.

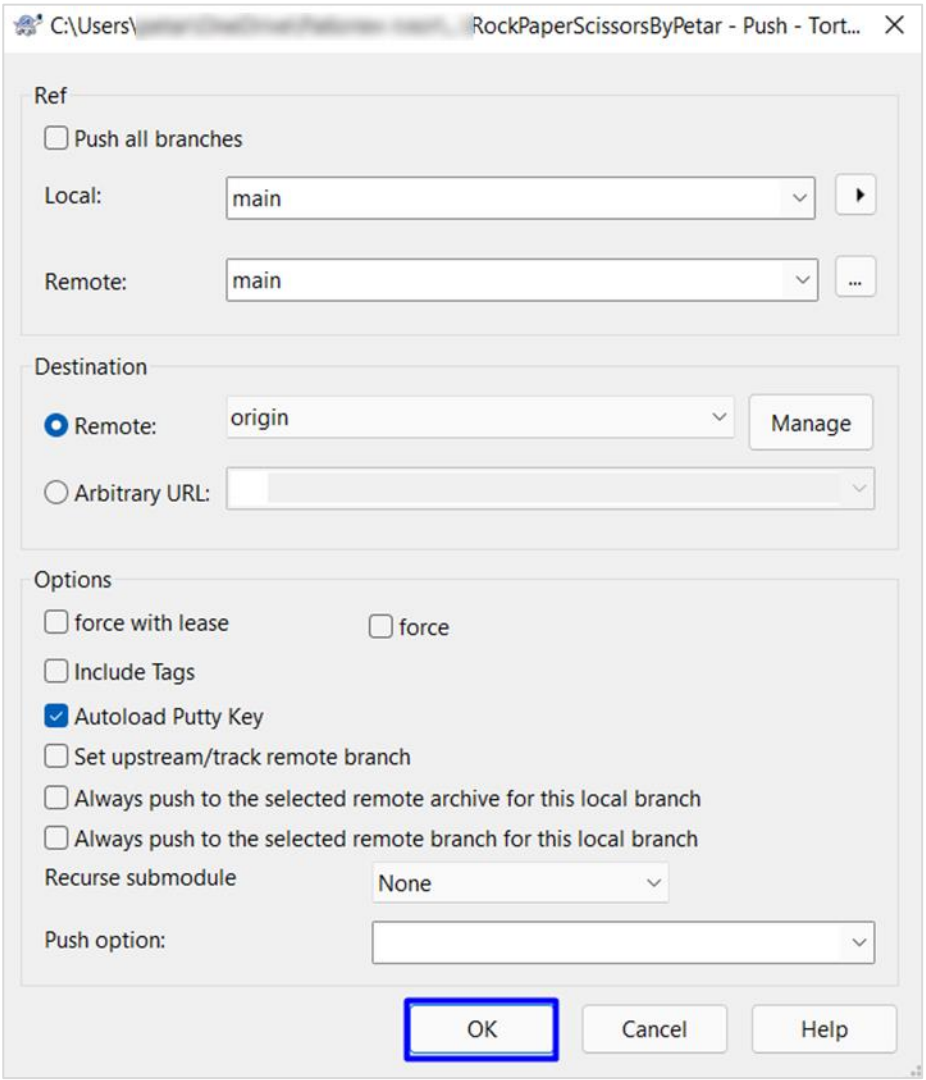


The results should be like this.

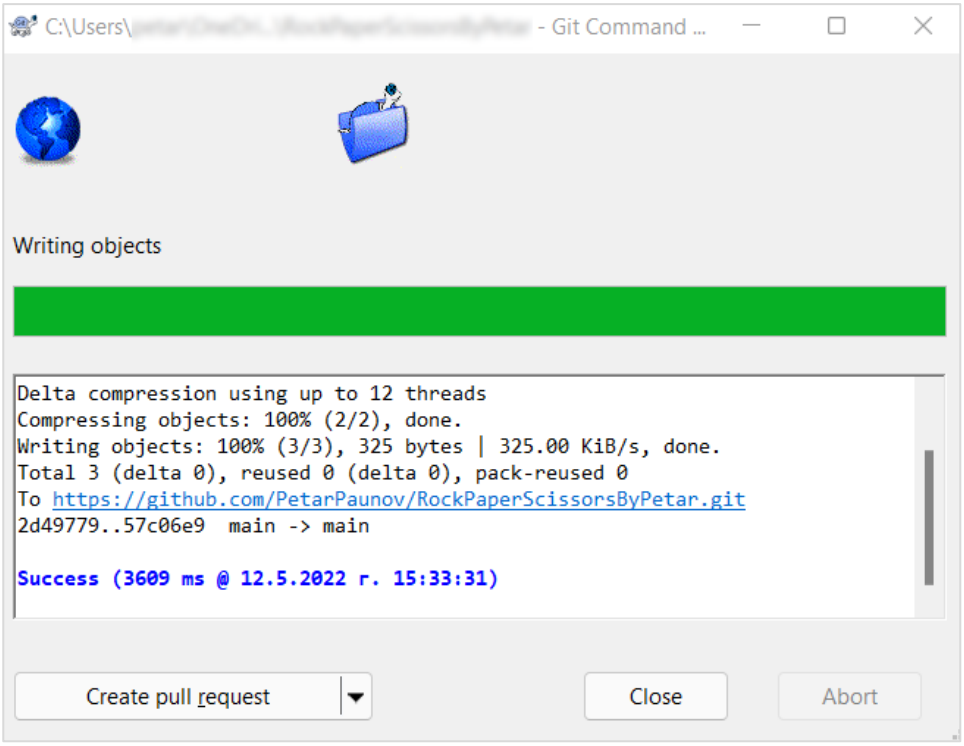


After that click **[Push]** and then **[OK]**.





The results should be like this.



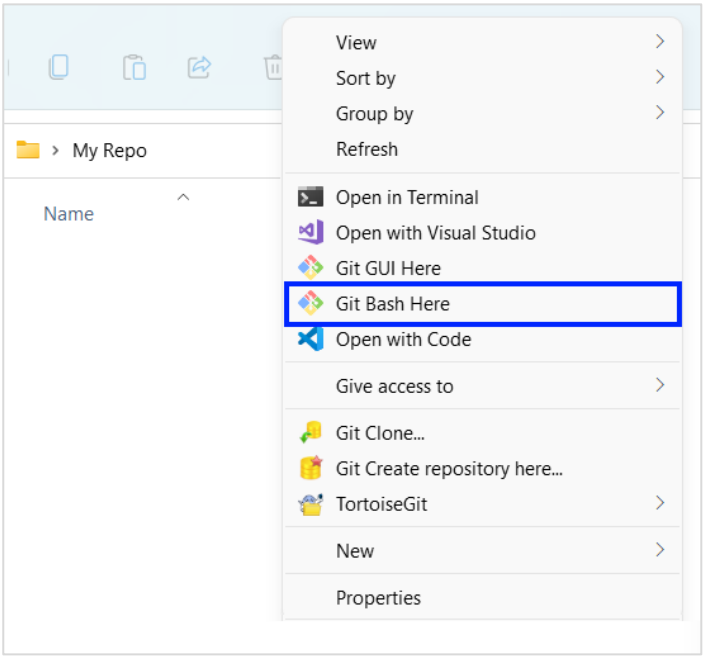
This is all you need to **upload your project source code** to your **GitHub repository** using **`TortoiseGit`**.

Use Git Bash (Option 2)

As **alternative to the previous step**, if you don't have "**TortoiseGit**", you could use the "**Git Bash**" command line tool to upload your project to your **`GitHub`** repo.

First, if you don't have **Git** on your **computer**, you should **install it** from https://git-scm.com/downloads.

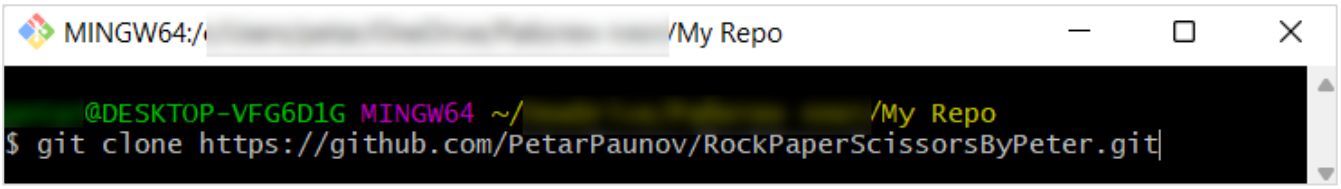
Go to the desired **directory**, right click on blank space **anywhere** in the folder, select "**Git Bash Here**" to open the Git command line console. If the "**Git Bash Here**" menu is missing, you should first install Git.



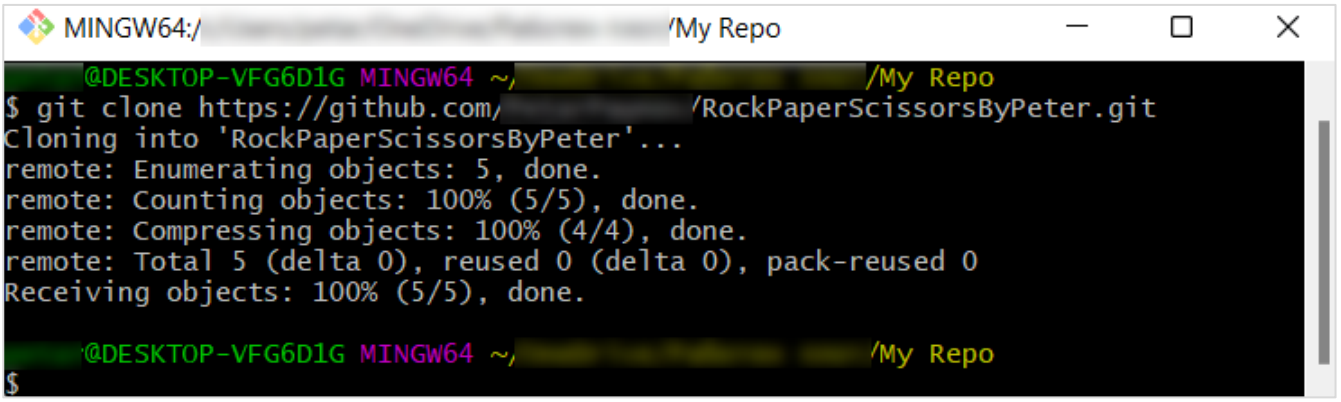
Type "**`git clone`**" command followed by the link of your **repository**:

```
git clone
```

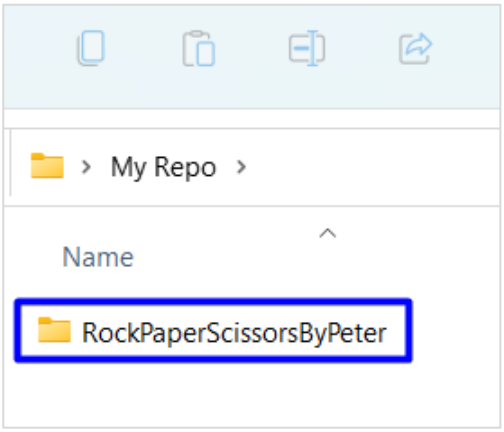
This command is for cloning with **Git Bash**, paste your **repository URL** after the command.



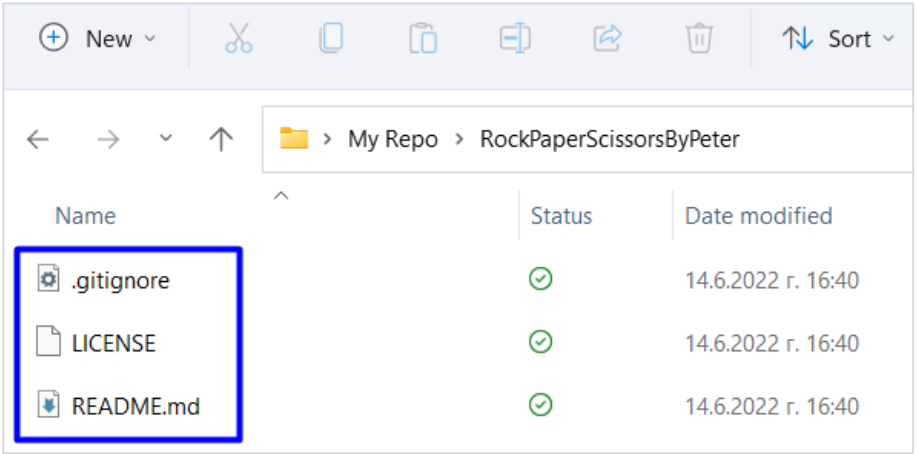
The result should be something like this:



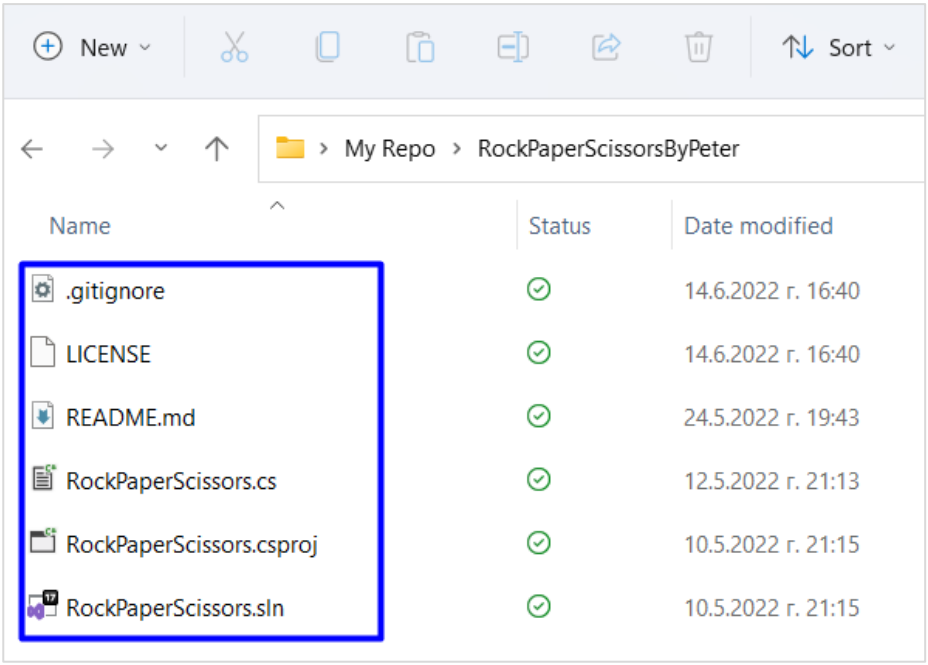
Your files from your GitHub repo will be downloaded to a **sub-folder** called as your project in GitHub, "**RockPaperScissorsByPeter**" in our case.



When we open the cloned **repository sub-folder**, it should look like this:



Next thing to do is to **add** your **project files** into your **cloned repository folder**. It should look like this:

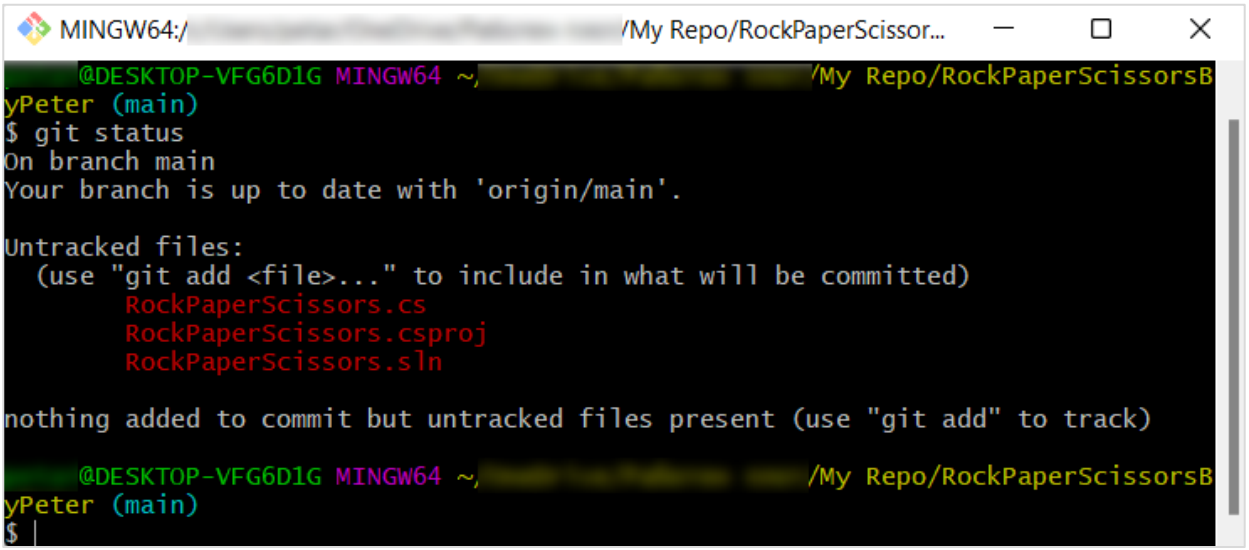


Now we are ready to upload our changes from "**Git Bash clone**". Go to the desired **folder**, right click on blank space anywhere in the folder, select "**`Git Bash Here`**" and run the following **commands**.

Type the following command:

```
git status
```

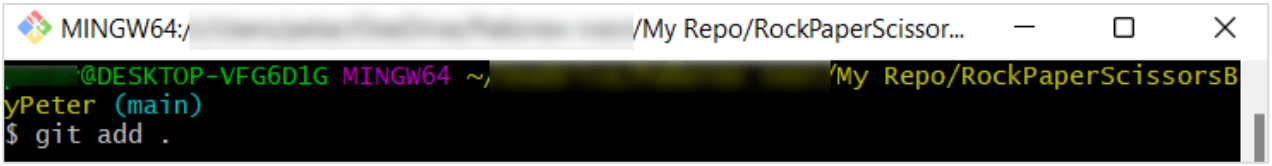
The **git status** command displays the state of the working directory and the **staging area**.



Now type:

```
git add .
```

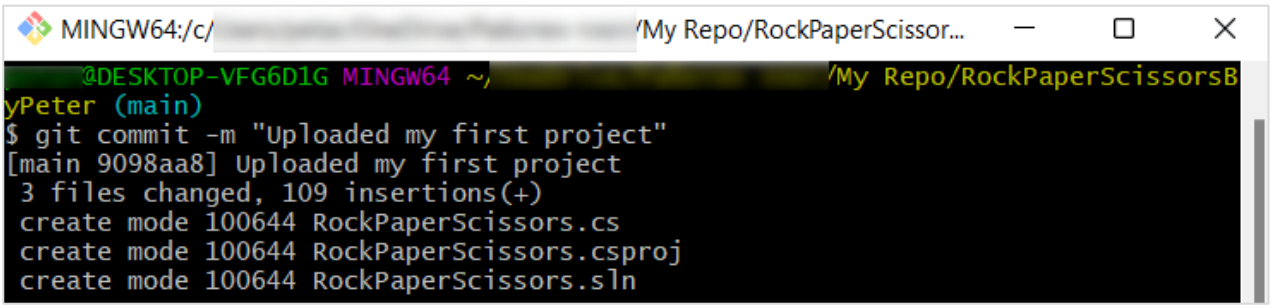
The above command **adds** all modified files to your local **Git repo**.



Now type:

```
git commit -m "Uploaded my first project"
```

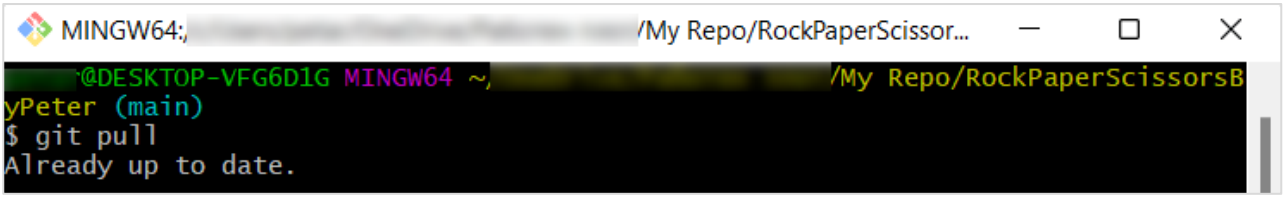
This command **commits** your changes to your local **Git repo**. We also should **add** an appropriate **commit message**.



We have **two** more **commands** left. Second to last type.

```
git pull
```

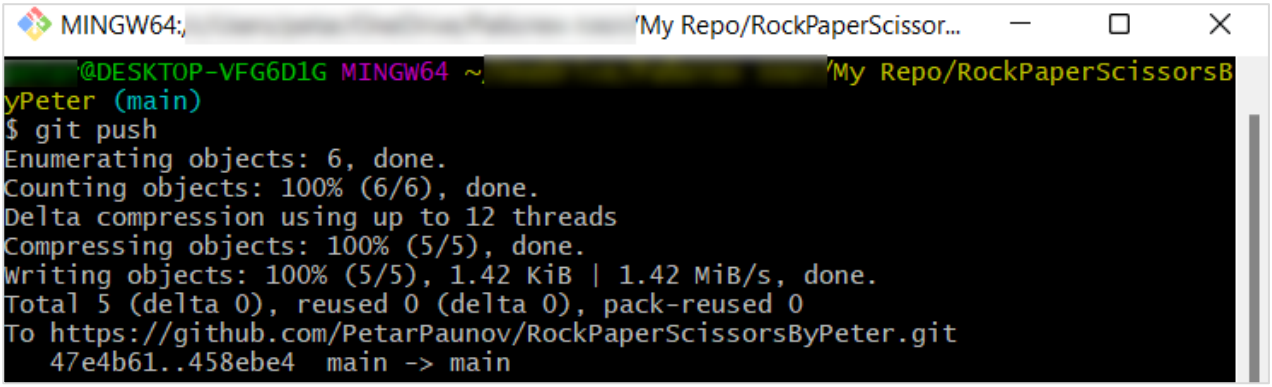
This command **updates** your local **repository** from GitHub. It downloads the latest project version from GitHub and merges it with your local copy.



Now the last thing that we should do is to **push** our changes by using the command.

```
git push
```

This command **pushes your local changes to GitHub**.



This is all you need to **update** your **repository** using **Git Bash**.

A little more information about Git Bash: https://git-scm.com/about.

4. * Modify the Code, Write Your Own Features

Now, it's time to **play with the code** and **modify it**.

This is your own project. **Be unique**. Don't be a copy-paster!

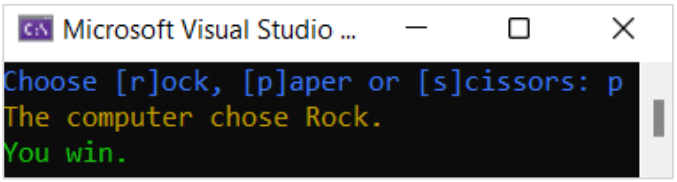
- Implement your **own features**.
- **Implement the code yourself**, using your own coding style, code formatting, comments, etc.

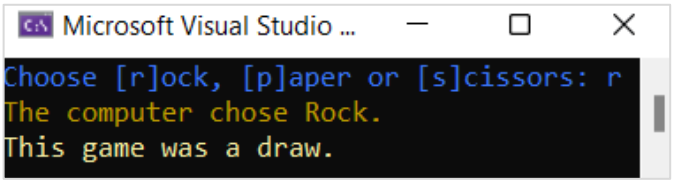
	• Make the project **more interesting**. Learn by playing with the code and adding your own changes.

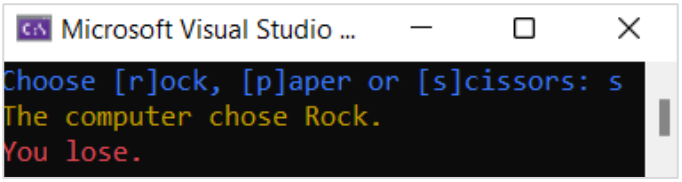
Below are a few **ideas** what you can implement or modify as addition to your code.

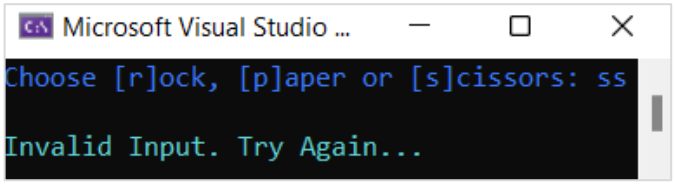
Add Colors

You can modify the **text color** and **text background** in the console: https://www.c-sharpcorner.com/article/change-console-foreground-and-background-color-in-c-sharp.



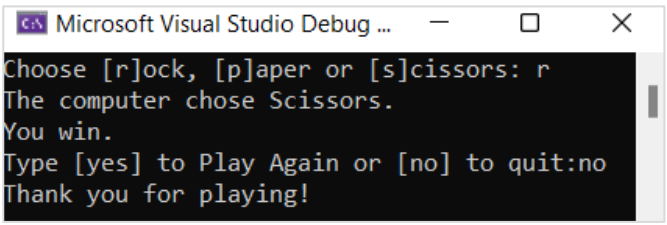


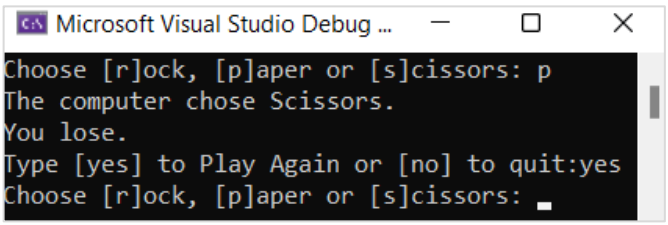




Restart the Game

You can automatically **restart the game** after it is finished (or ask the player to play again).





Scoring System

You can add **scoring system** and display the player's and the computer's score after each game session.

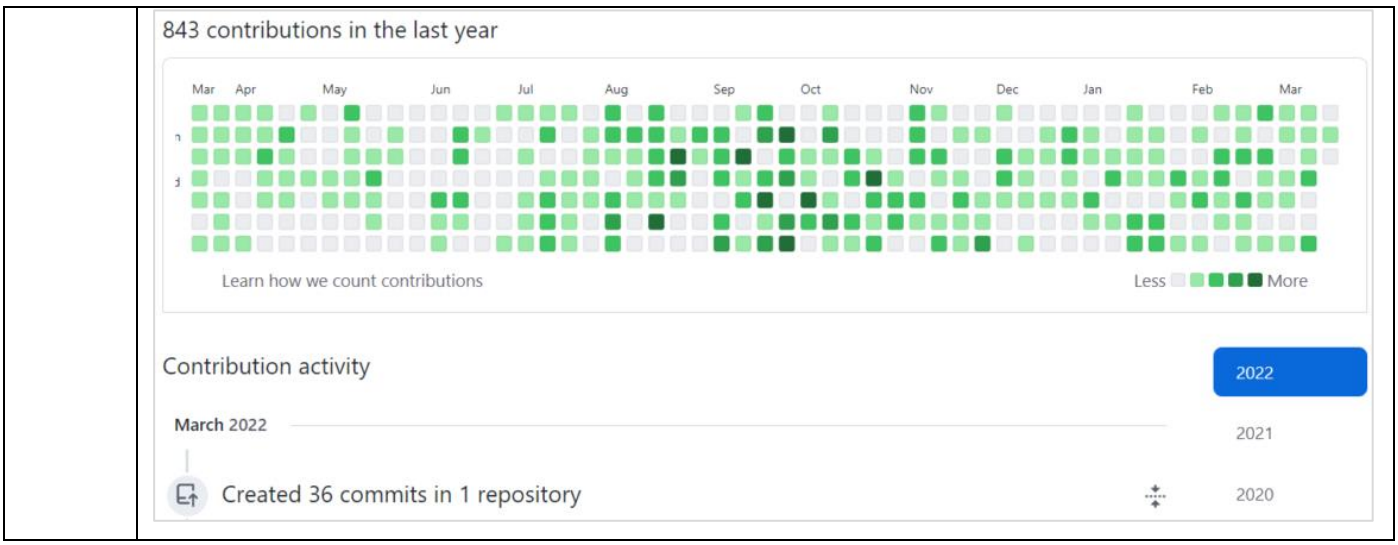
Additional Ideas

- Can you change your logic, so you can **increase the chances of the player to win**?
- Can you add **anything else** in your code, based on you own ideas?

Commit to GitHub

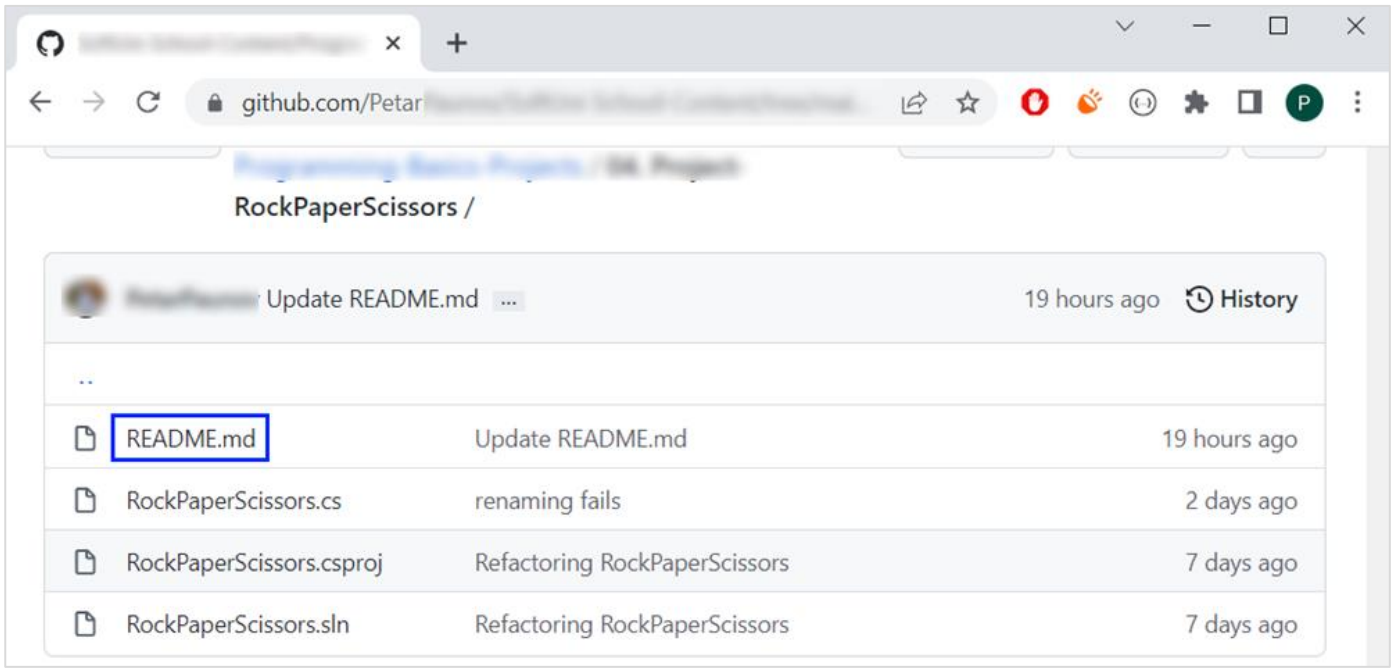
Now **commit and push your code changes** to your GitHub repo!

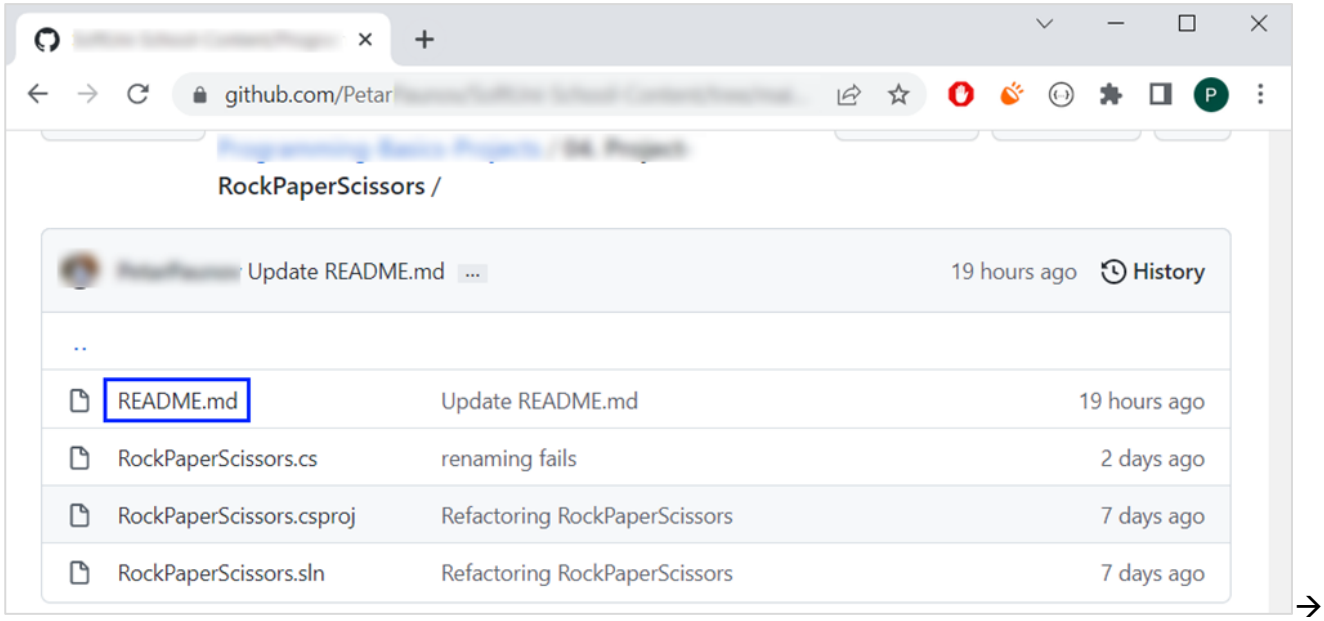
	It is very important to **commit frequently** your code to GitHub. This way you create a **rich commit history** for your project and your **GitHub contribution graph** is growing:

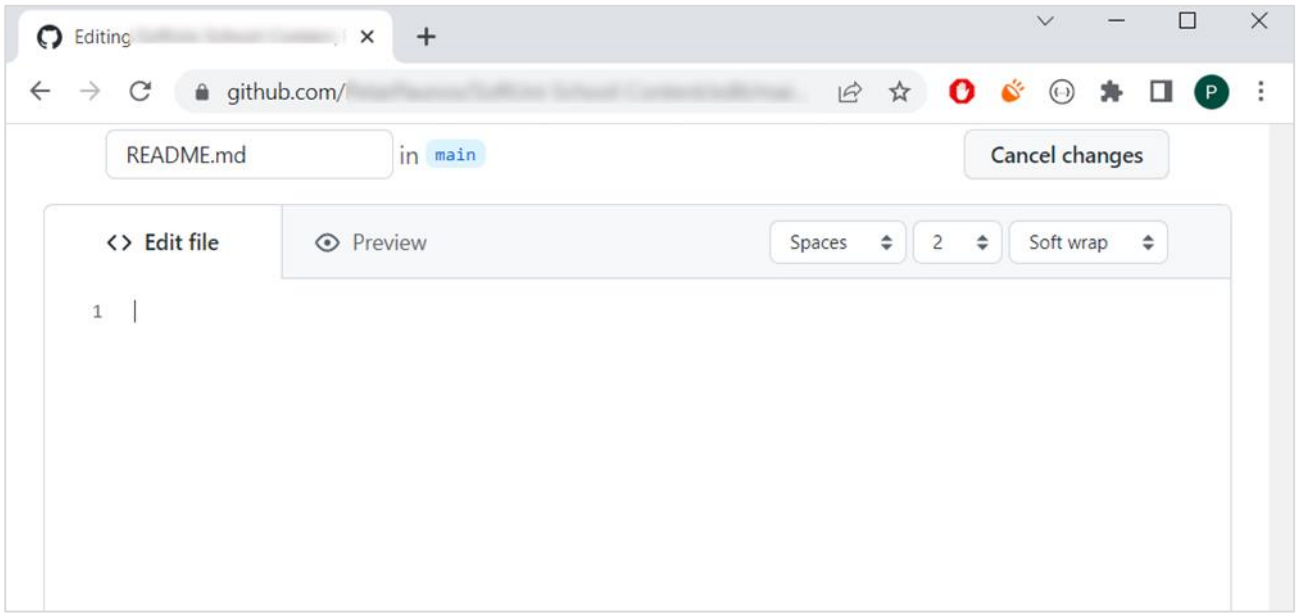


5. Create a README.md File

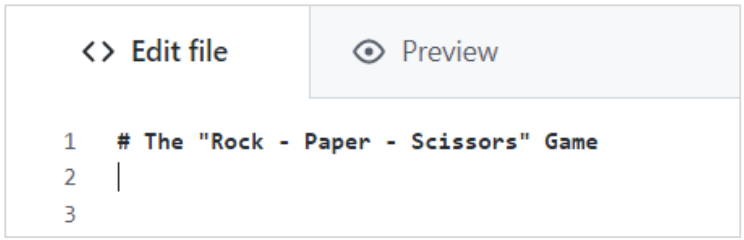
It's highly recommended to provide **documentation as part of your project in GitHub** to describe what the project is **doing**. So, let's make one for this **project**. Let's start by editing the **`README.md`** file from our repo at GitHub:



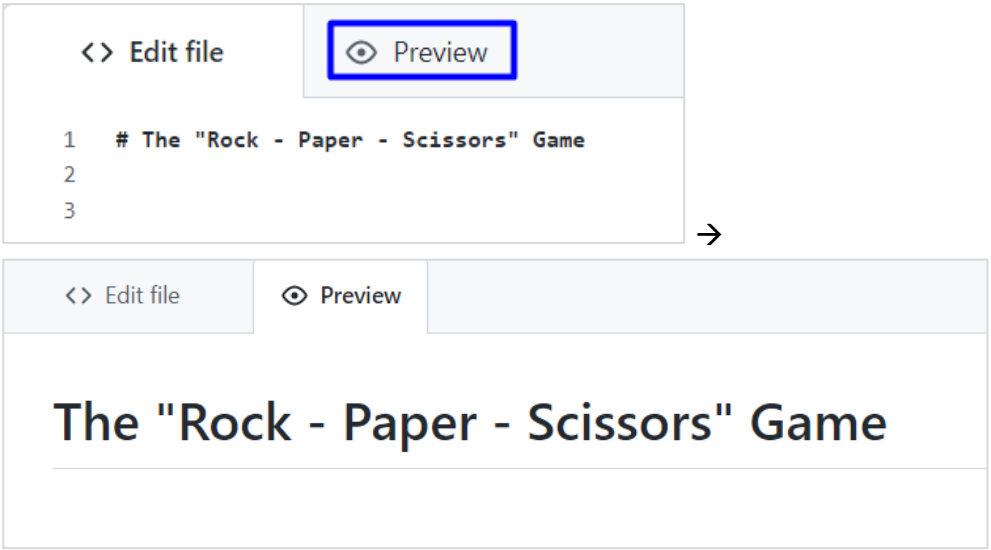




Add a project name. Use **"#"** in front of the text to indicate the **title**:



You can **view** the current progress by pressing the **`[Preview]`** button:



Documentation Sections

Add **information** about your project in your **README.md** file: project goals, technologies used, screenshots, live demo, etc. Typically, you should have the following **sections**:

- **Project title** (should answer the question "What's inside this project?")
- **Project goals** (what problem do we solve, e. g. we implement a certain game)
- **Solution** (should describe how do we solve the problem → **algorithms**, **technologies**, **libraries**, **frameworks**, **tools**, etc.)
- **Source code link** (give a direct link to your source code)
- **Screenshots** (add screenshots from your project in different scenarios of its usage)
- **Live demo** (add one-click live demo of your code)

Use Markdown

Note that the GitHub **README.md** file is written in the **Markdown language**. Markdown combines text and special formatting tags to describe formatted text document.

You can learn more about **Markdown** here: https://docs.github.com/en/get-started/writing-on-github/getting-started-with-writing-and-formatting-on-github/basic-writing-and-formatting-syntax.

Project Goals

Start your documentation by describing your **project goals**. What problem do your project solve?

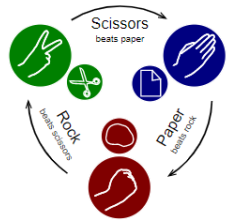
Sample Documentation

This is an **example** how you can document your project. Don't copy and paste it!

Edit file | Preview | Show diff

The "Rock - Paper - Scissors" Game

A console-based C# implementation of the "Rock - Paper - Scissors" game.



Rock - Paper - Scissors is a simple **two player game**, where you and your oponent (the computer) simultaneously choose one of the following three options: "**rock**", "**paper**" or "**scissors**". The rules are as follows:

- **Rock beats scissors** (the scissors get broken by the rock)
- **Scissors beats paper** (the paper get cut by the scissors)
- **Paper beats rock** (the paper covers the rock)

The **winner** is the player whose choice beats the choice of his oponent. If both players choose the same option (e.g. "paper"), the game outcome is "**draw**".

Write the project documentation yourself. Don't copy and paste it!

This is your **unique GitHub profile** and your own unique project. **Be different** from others.

You can add **appropriate images** to make your documentation better. You can add **image** as follows:

```
<img alt="Image" width="200px" src="[redacted]" />
```

You can add information about the **inputs** and **outputs** of the project:

Input and Output

The player enters one of the following options:

- `rock` или `r`
- `paper` или `p`
- `scissors` или `s`

The computer chooses a **random option**, then reveals the **winner**.

Your Solution

Describe how do you **solve** the problem: **algorithms**, **technologies**, **libraries**, **frameworks**, **tools**, etc.

For example, for our simple game you may analyze all possible game **situations** in a **table**:

Solution

You	Computer	Outcome
rock	rock	Draw
rock	paper	You lose
rock	scissors	You win
paper	rock	You win
paper	paper	Drow
paper	scissors	You lose
scissors	rock	You lose
scissors	paper	You win
scissors	scissors	drow

We handle all these situations using a series of checks.

Link to the Source Code

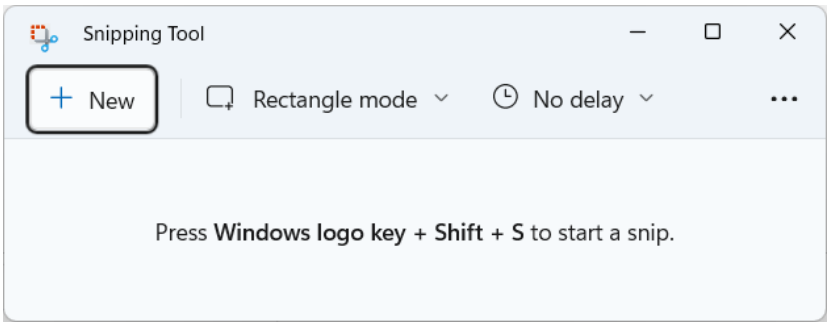
Add a **link** to your **source code** as follows:

```
[Source Code](RockPaperScissors.cs)
```

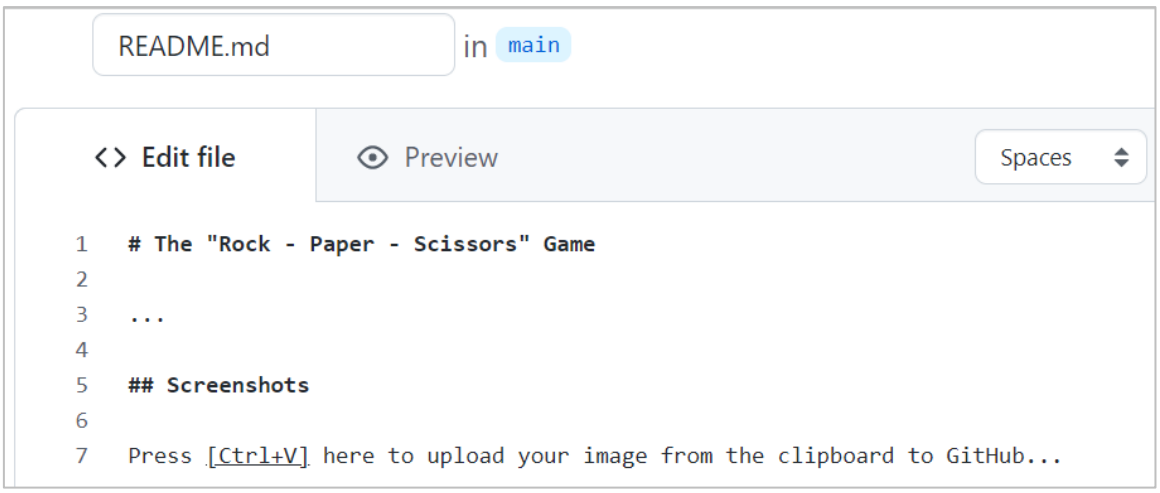
Screenshots

Add **screenshots** of your project:

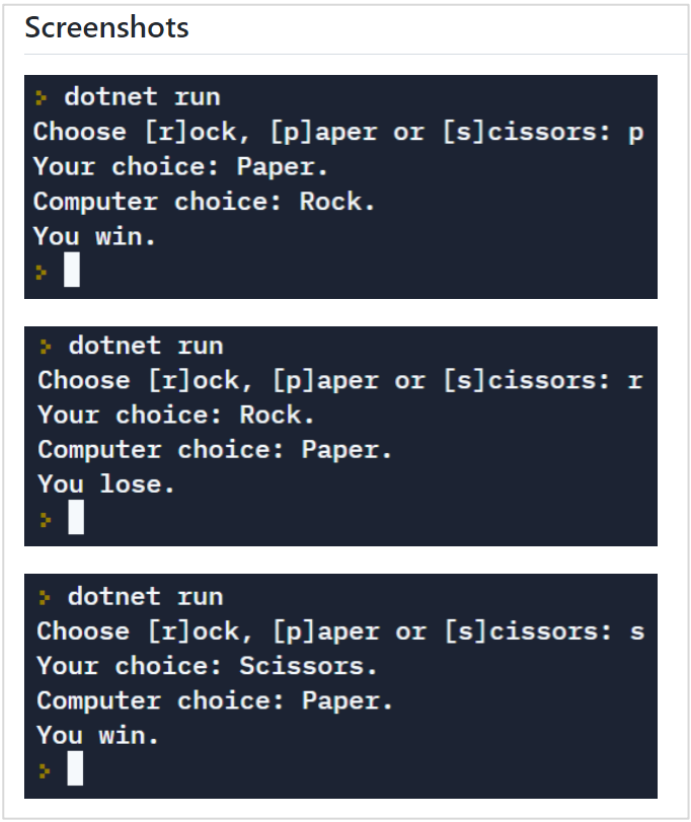
1. **Take a screenshot** with your favorite tool (e.g. the Snipping Tool in Windows).



2. **Paste** the screenshot in the GitHub Markdown editor, using **`[Ctrl+V]`**:



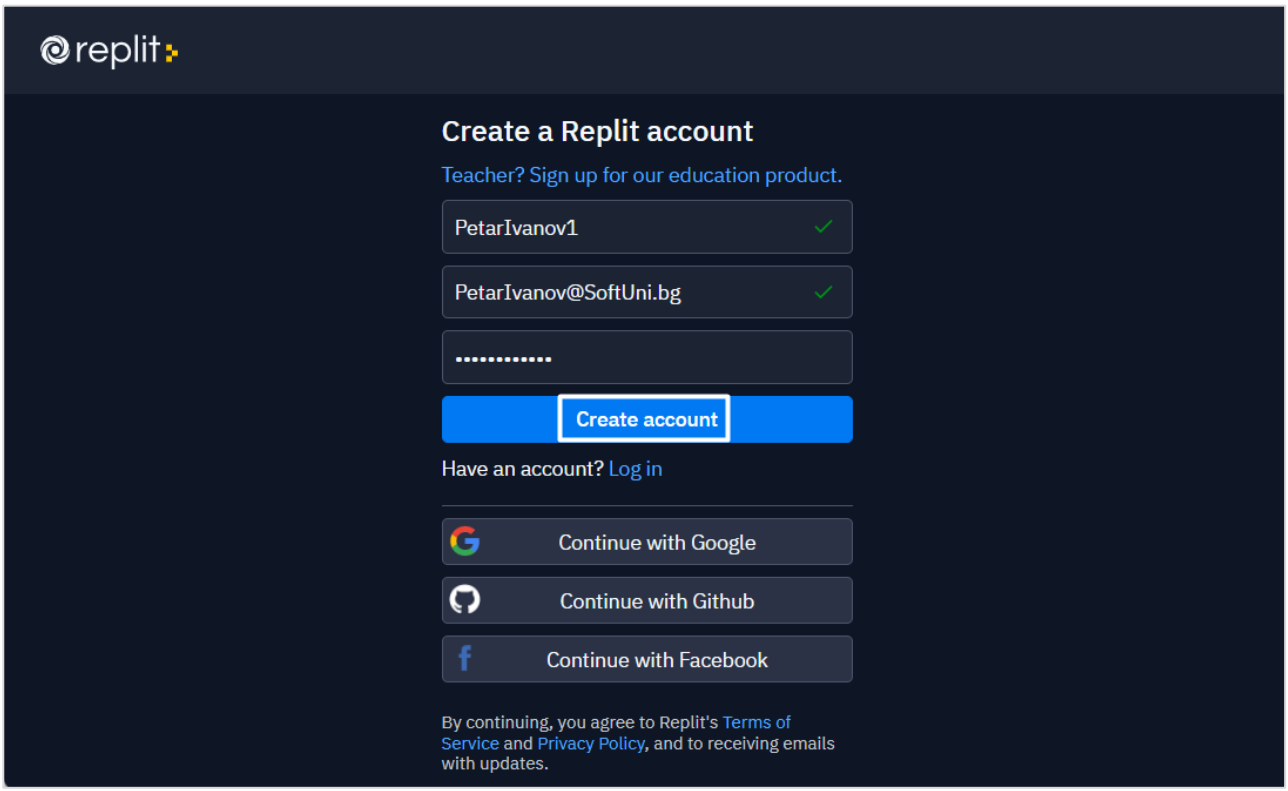
Example screenshots for the "**Rock Paper Scissors**" game:



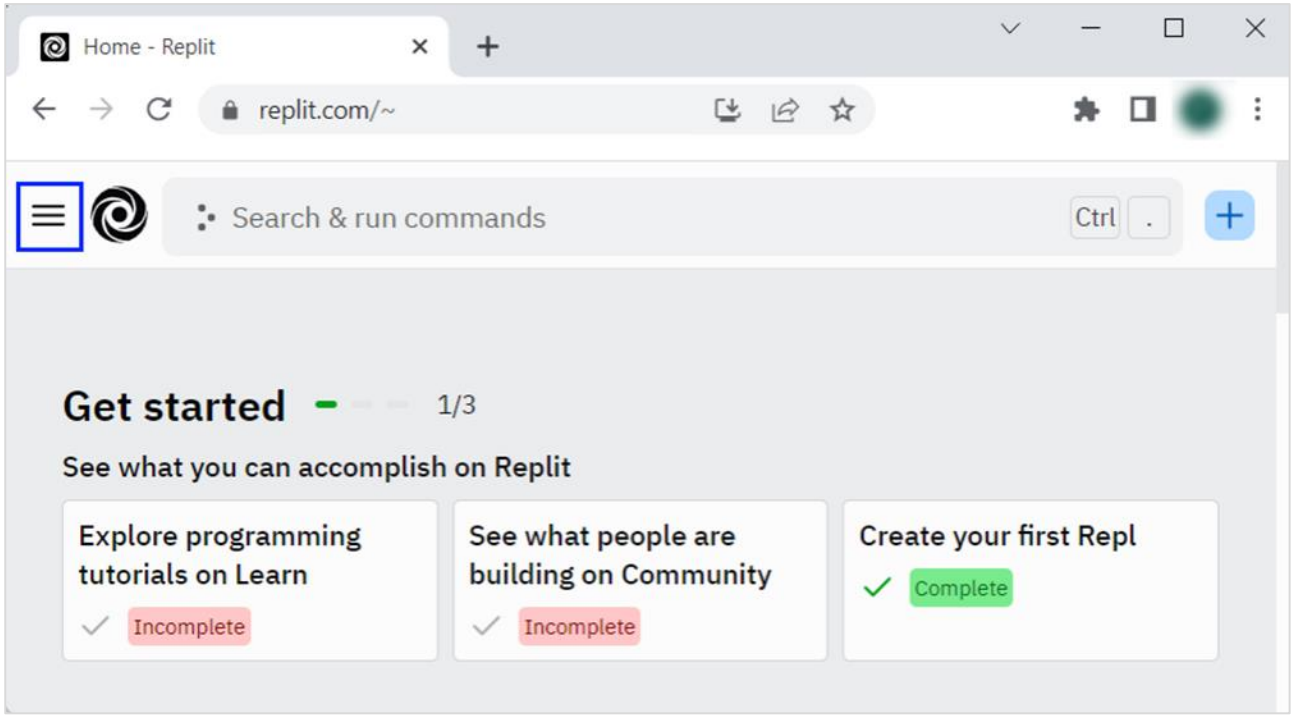
6. Upload Your App to Replit

Replit is an online coding environment (online IDE), which allows you to **write** software projects, **share** them though a simple link and **run** your projects directly in the Web browser. We shall upload our project in **Replit** to allow the users to **run and interact with the project** with just **one click**.

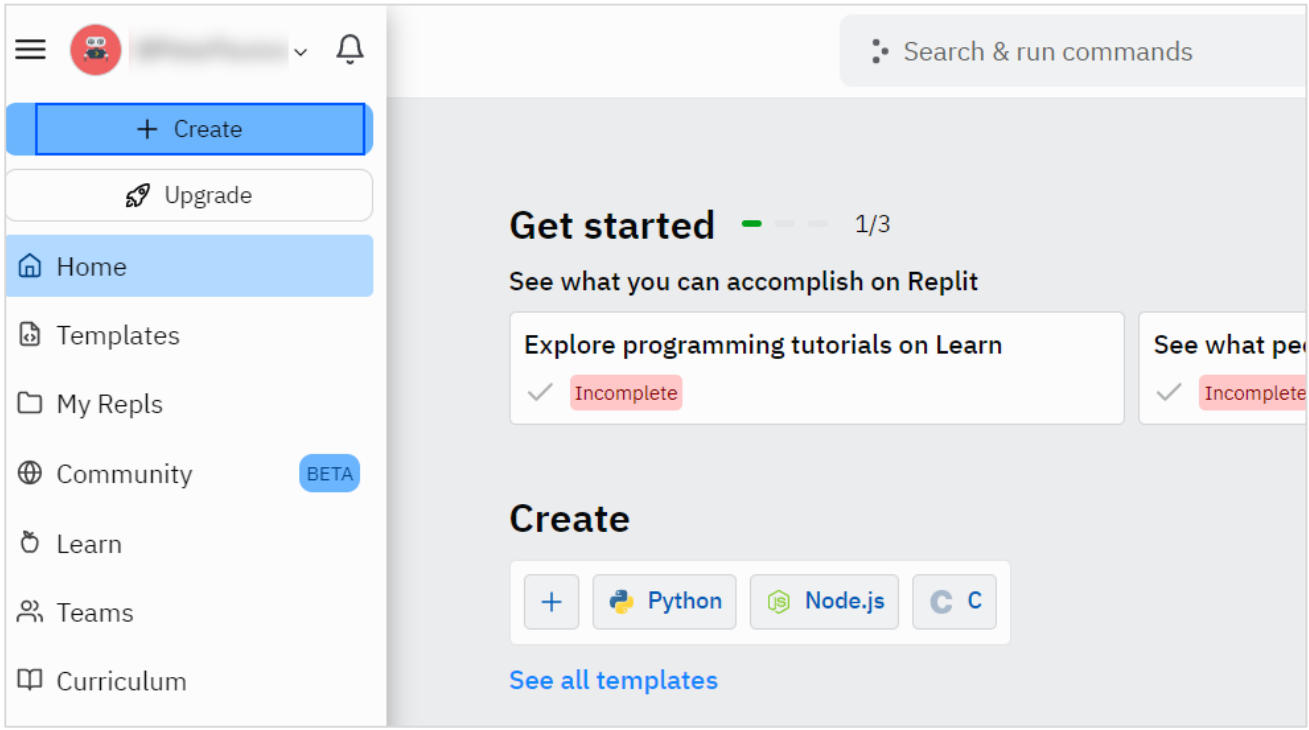
Create your own **Replit** profile so you can show your **projects** to your friends and also put "**live demo links**" it in your **GitHub** project documentation. Create a **Replit** account for **free**: https://replit.com.



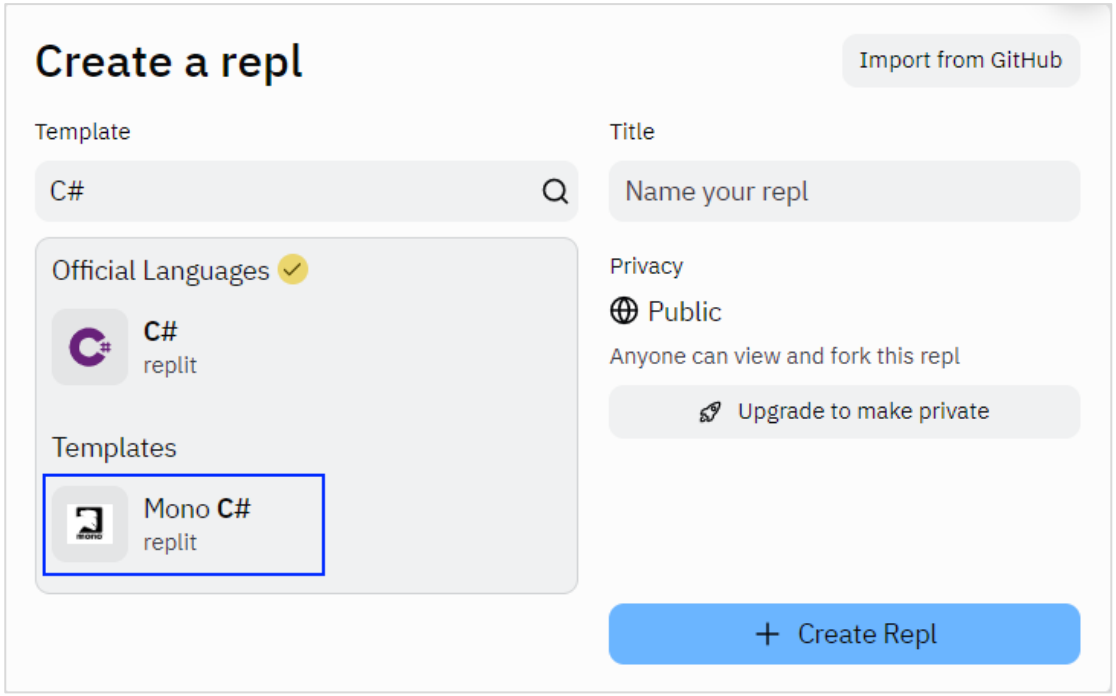
Create a **new project** in **`Replit`**, open the **menu** in upper **left corner**.



Click **[Create]**, then select the **language** in which your project is **written**, select a name, and **create** the project.

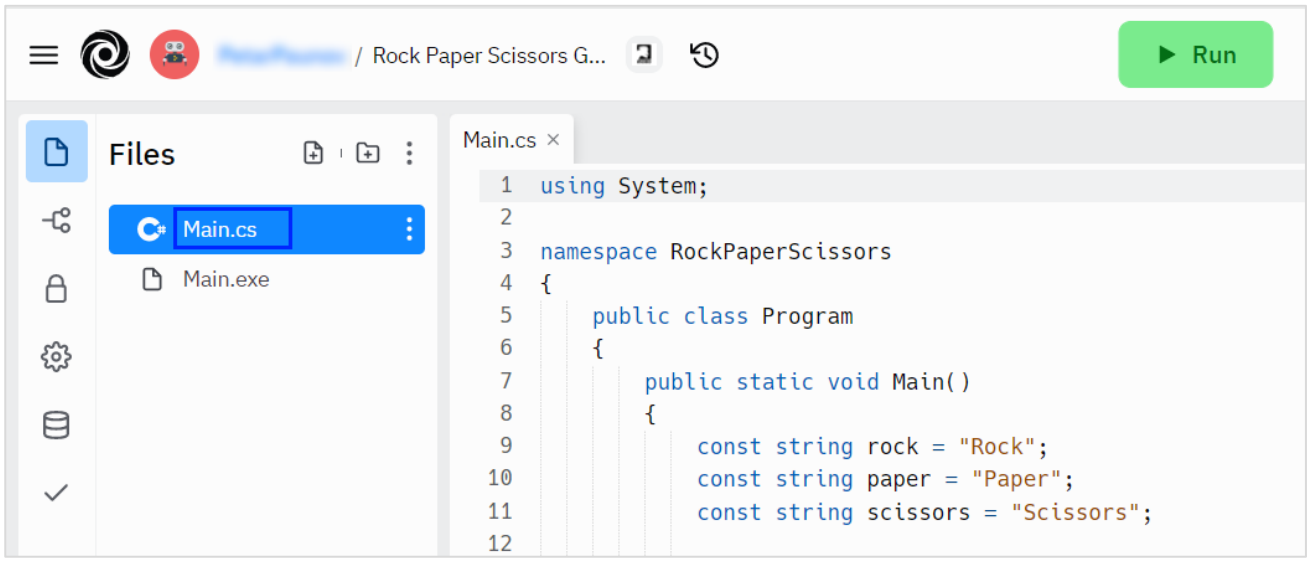


If your project is in **C#**, choose "**Mono C#**". In **Replit** the C# projects work faster with Mono, than with .NET 6.

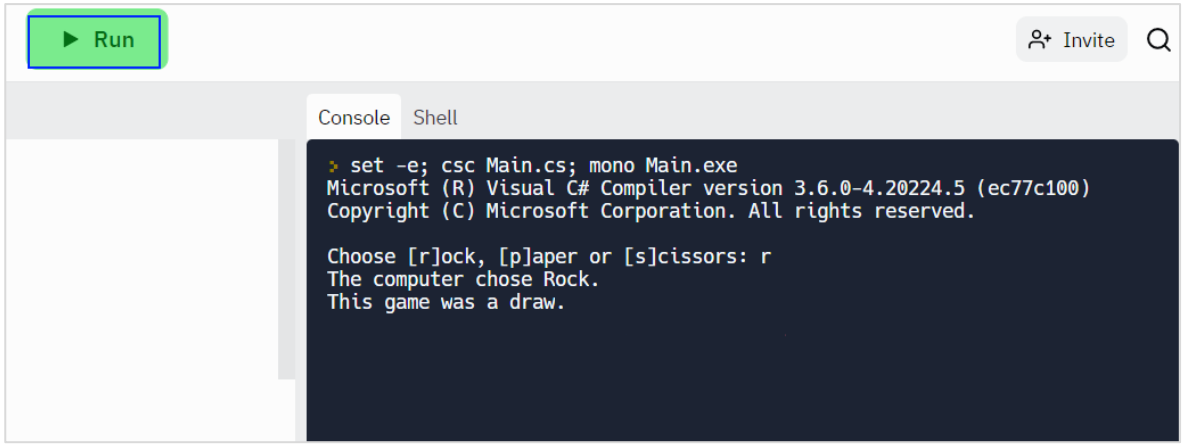


Add a meaningful **name** to your **`Replit`** project, e.g. "**RockPaperScissors-Game-by-Peter**".

Paste your code in "**`Main.cs`**" file:



Click [`Run`] and enjoy your console application directly in the Web browser:



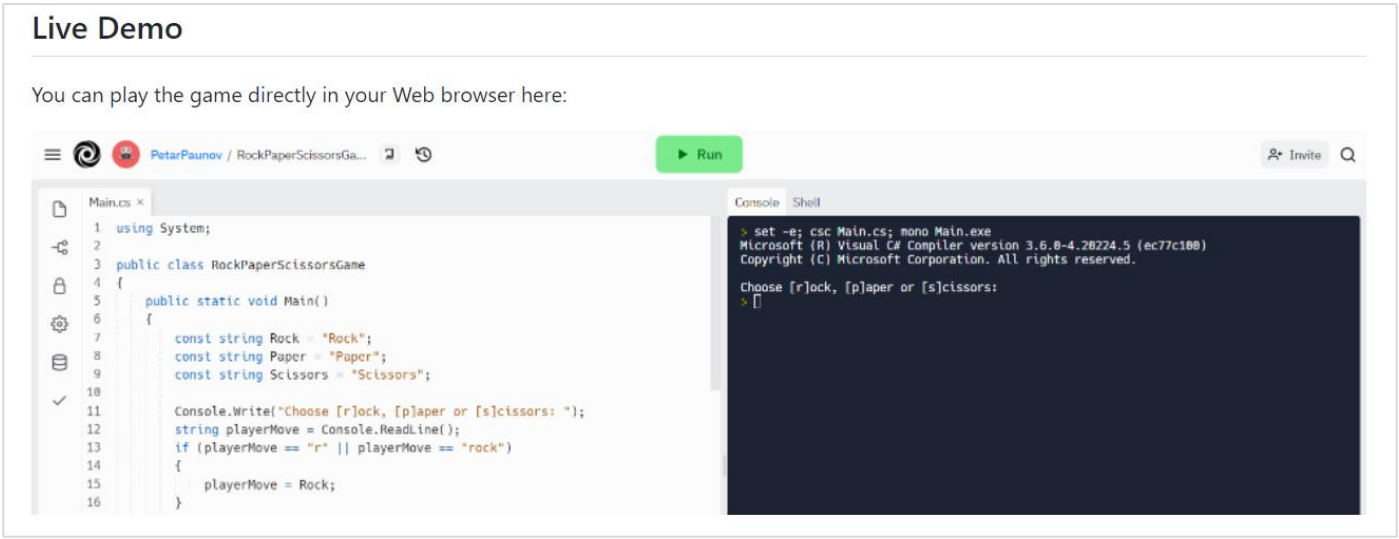
7. Add Replit Link to Your README.md

Now add a "**one-click live demo**" of your project from your GitHub project documentation. You can do it as follows:

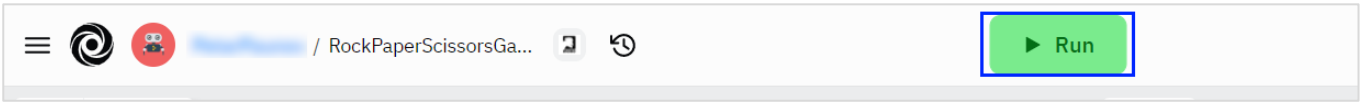
```
## Live Demo

You can play the game directly in your Web browser here:

[<img alt="Play Button" src="https://user-images.githubusercontent.com/85368212/167706726-d027f056-fc2b-47b7-bfad-8ff8a3aa7688.png" />](https://replit.com/@PetarPaunov/Rock-Paper-Scissors-Game#Main.cs)
```

You can take a **screenshot** from Replit.com and **paste it** in the GitHub documentation editor directly with **[Ctrl+V]**.

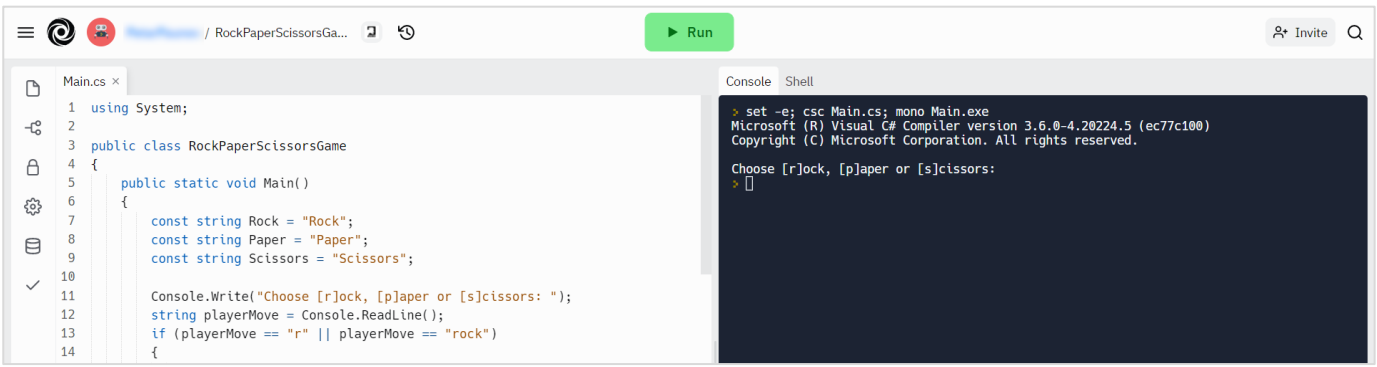
This is how it should look like after the changes in your **`README.md`** documentation:



Now when the [**Run**] button is clicked you will be redirected to your demo in **`Replit`**.



→



Now we have completed our **first console game** and we have our first project in our GitHub portfolio.