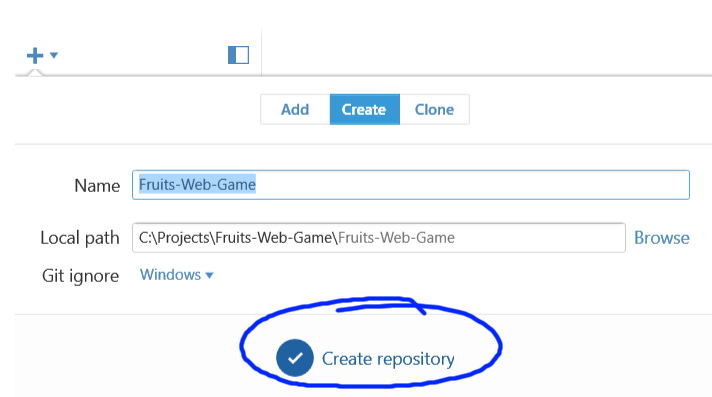
# Exercises: Git and GitHub, Debugging and Troubleshooting

Problems for exercises and homework for the [“Programming Fundamentals” course @ SoftUni](https://softuni.bg/courses/programming-fundamentals).

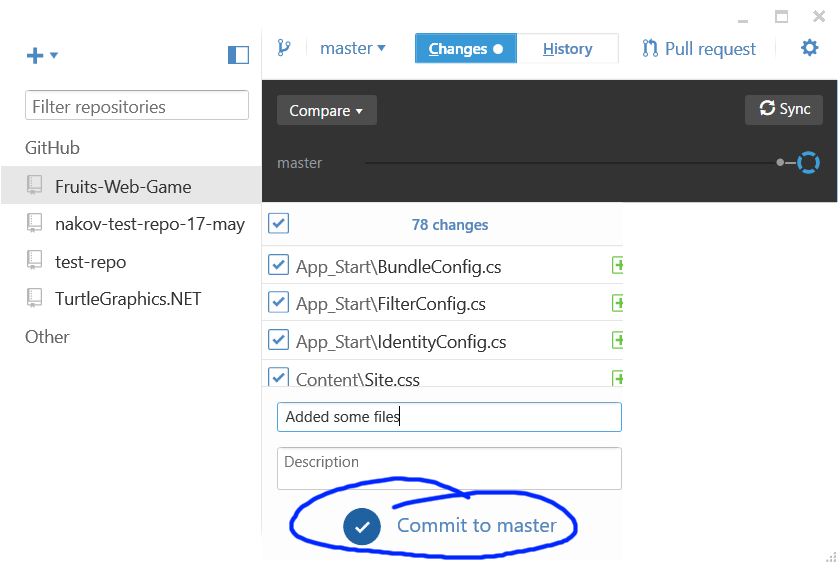
## Upload a Few Projects at GitHub

Create a few **repositories** in your **GitHub** profile and **upload a few of your projects to GitHub**. These could be your **homework exercises** for the last few courses, your **teamwork projects** or any other projects that you might want to share with the developer community. Follow these steps:

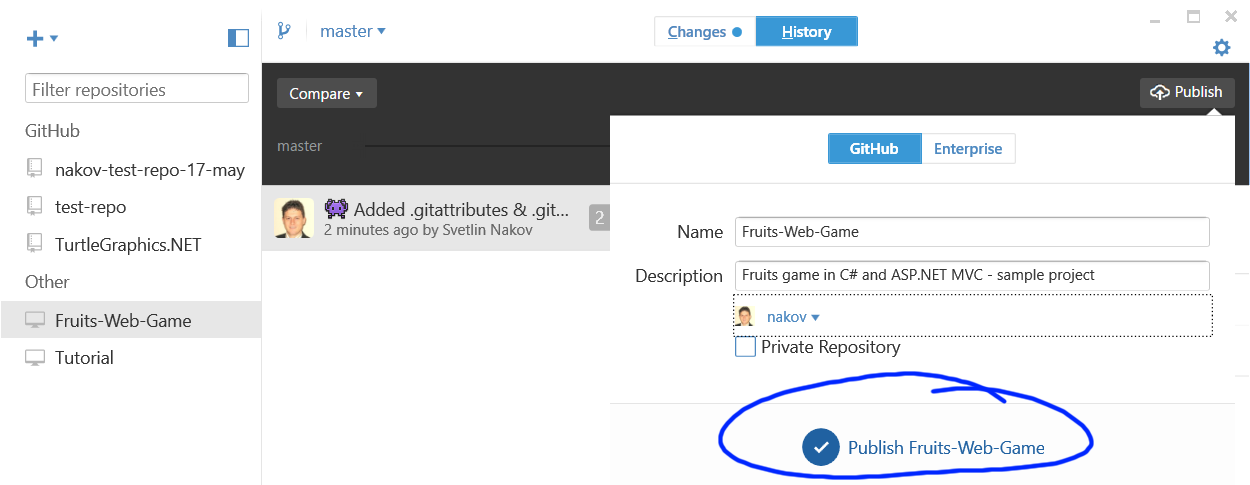
1. First **create a local repository** for your existing project (folder in your hard disk). This will put your project under version control (in fact it will create a hidden folder named .git).



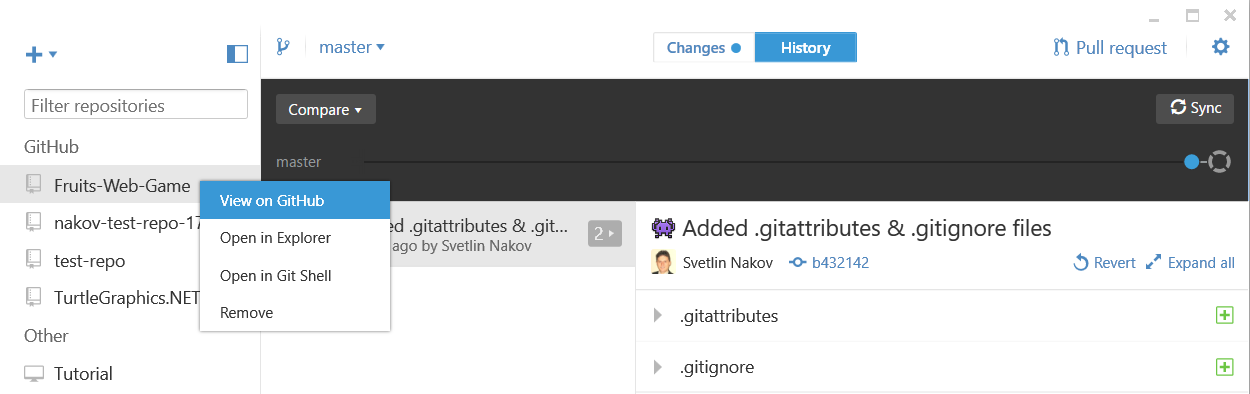
1. **Commit** your local files to your recently created local repository:

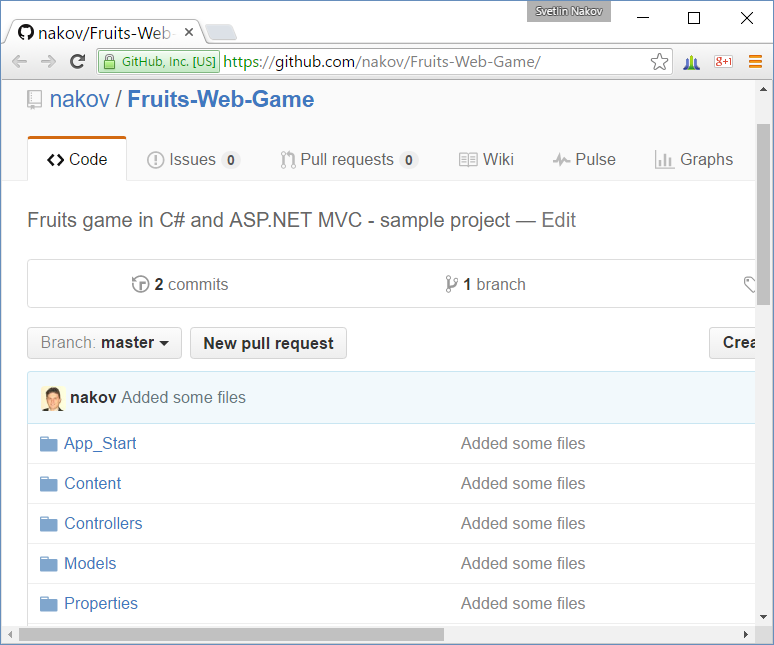


1. Then **publish** your local repository to GitHub:



1. **Open your project at GitHub** to see whether is it really uploaded live on the Web:



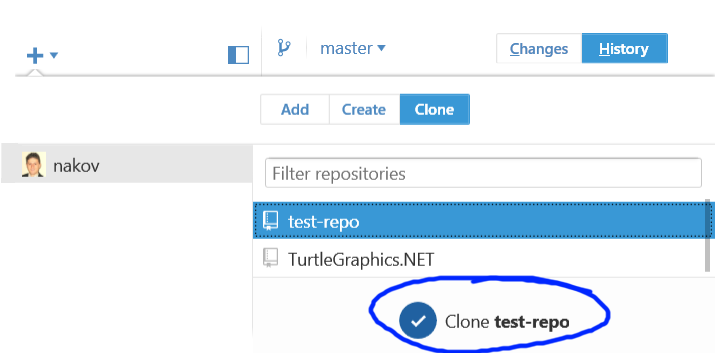


Note that all projects your upload at GitHub will be **open-sourced** and will be accessible for anyone in Internet, so be careful about passwords or code which you might not want to be visible by someone else.

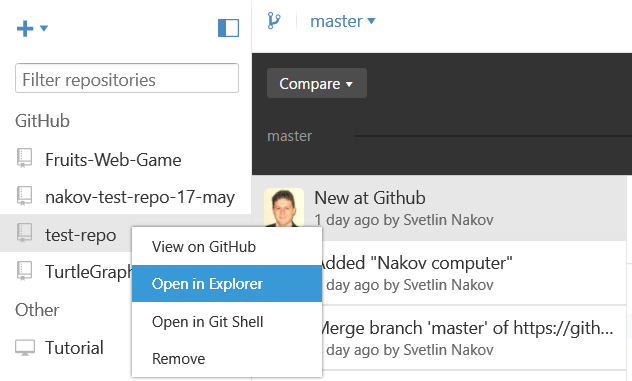
## Play with Git: Clone, Change, Commit, Push

**Clone** some of your GitHub repositories through your **Git client** (e.g. using the **GitHub Desktop** software). Make some **changes** and **commit** them locally, then **push** them to GitHub. Check whether the changes are published in your GitHub profile in Internet. Follow these steps:

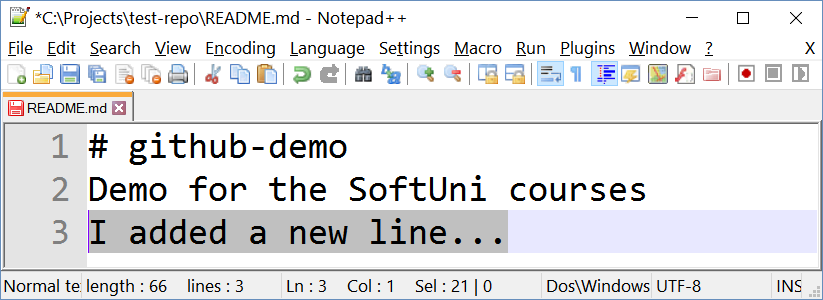
1. **Clone** an existing repository from your GitHub account into a **local folder** on your hard disk:



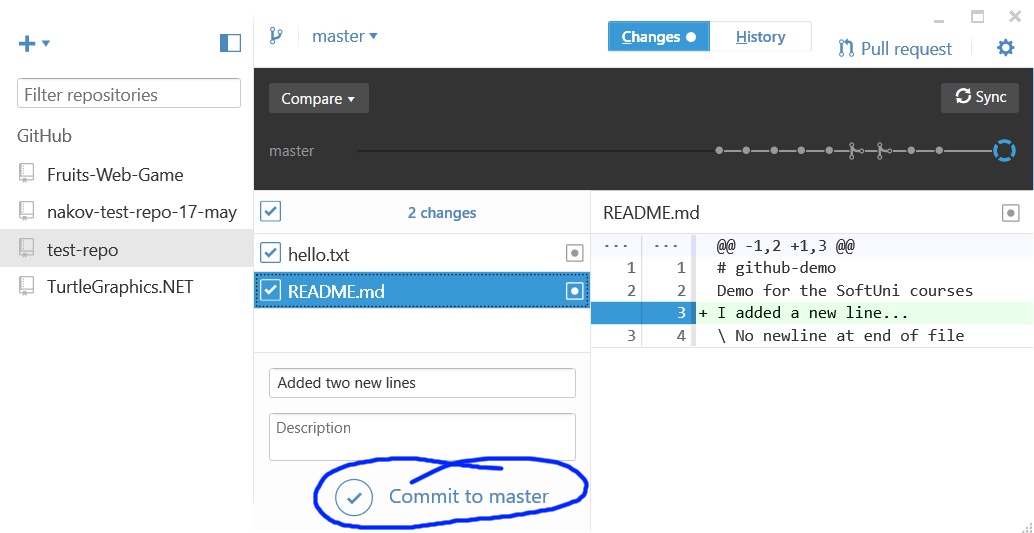
1. Open the project files in **Windows Explorer**.



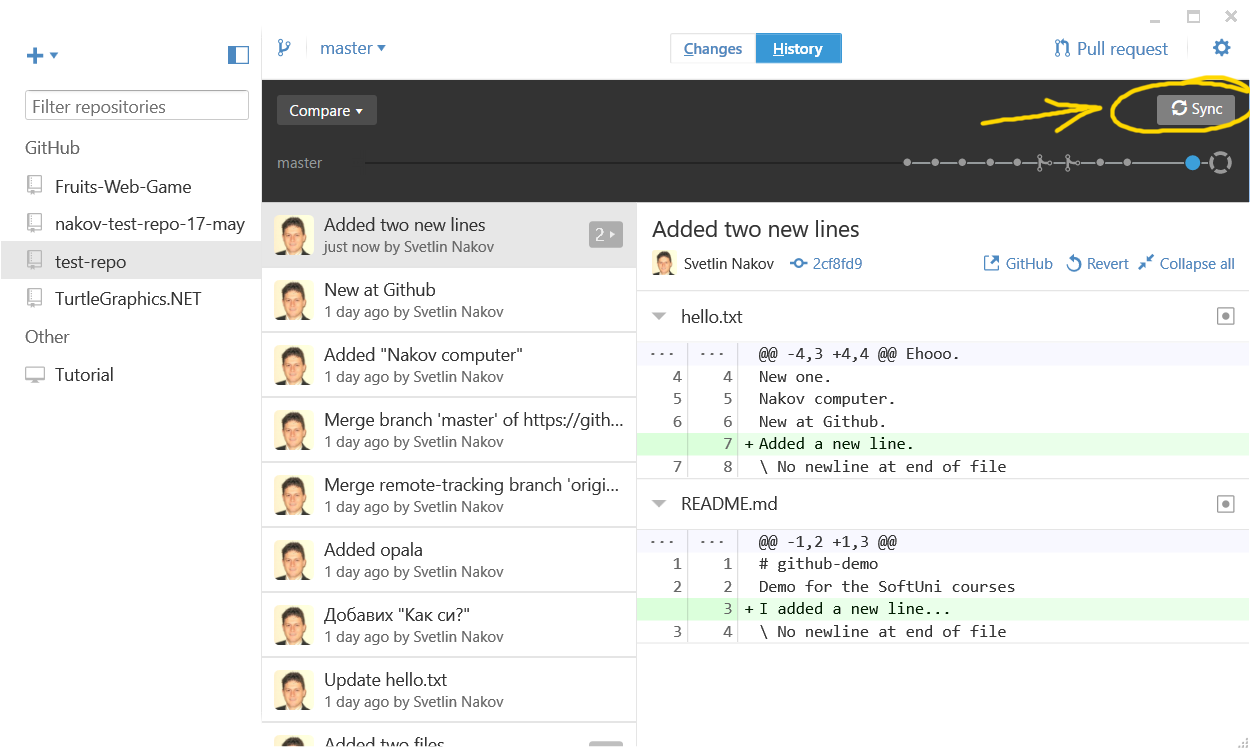
1. Make some **changes** in your favorite text editor:



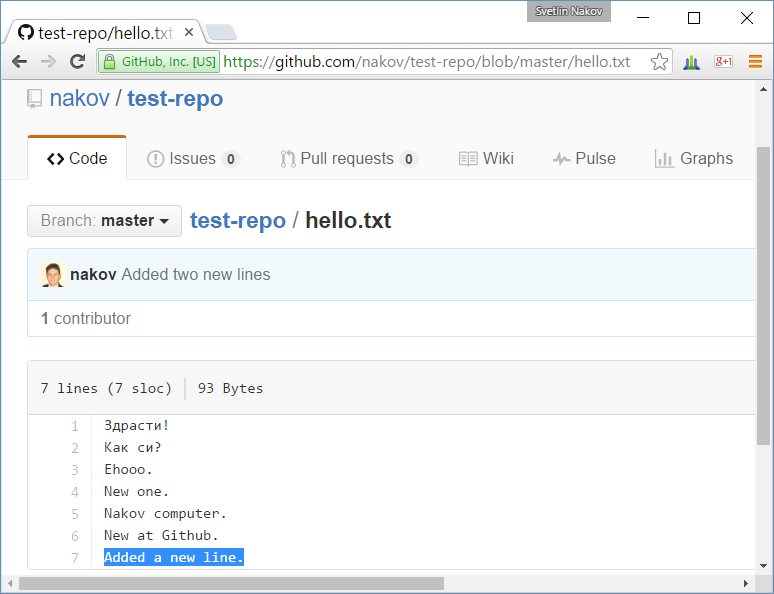
1. **Commit** your local changes to your local repository.



1. **Push** your changes to the remote repository in GitHub (use **Sync**):



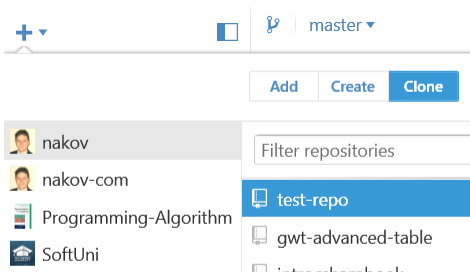
1. Check whether your changes are online:



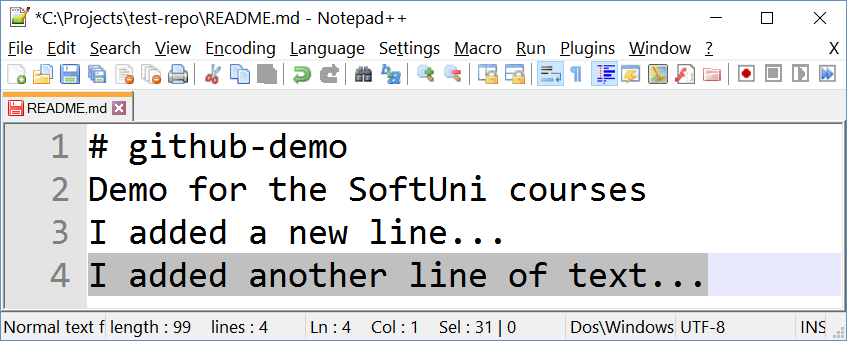
## Merge Conflicting Changes in GitHub

Create **conflicting changes** and **merge them**. Use the following steps:

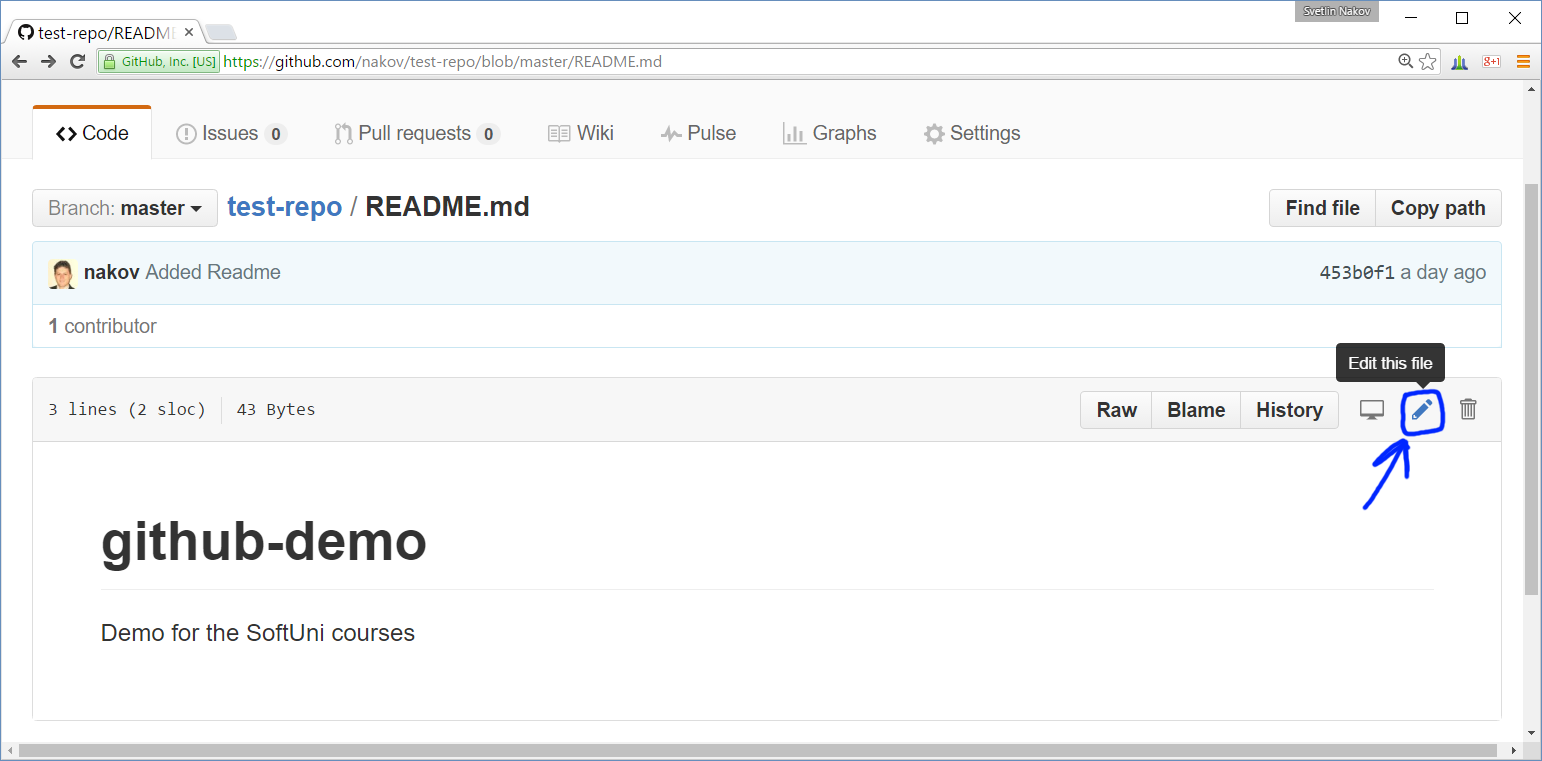
1. **Clone a repository** from your GitHub account (if not already cloned). Use your favorite Git client software:

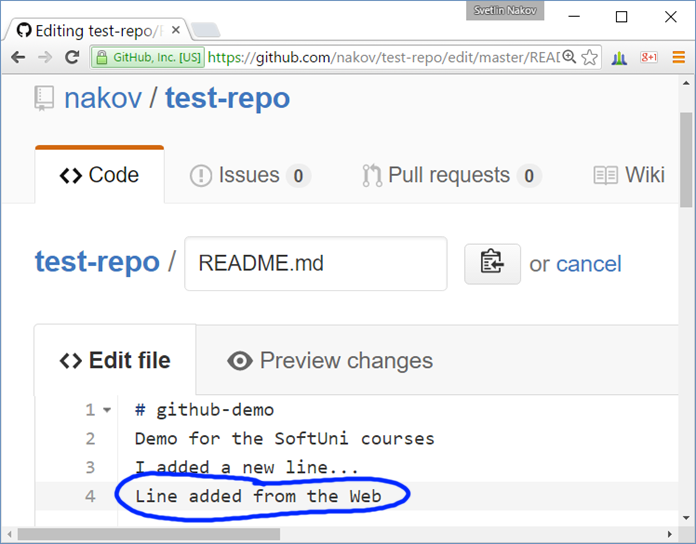


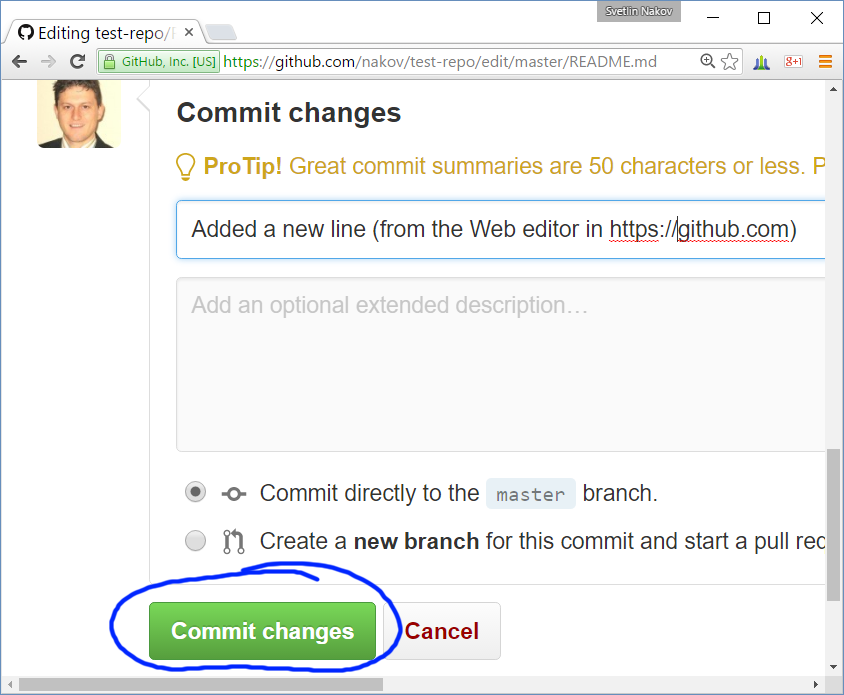
1. Make some **changes** in your working directory, e.g. edit the file README.md.



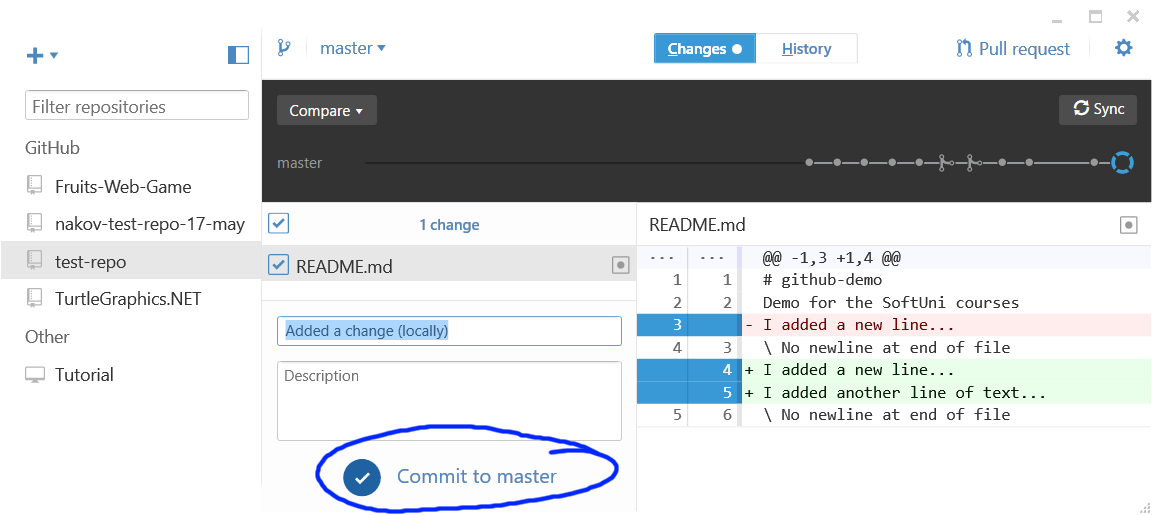
1. **Don’t comm**it and **don’t push** your changes yet.
2. Open your GitHub account from your **Web browser**. Make some changes online:



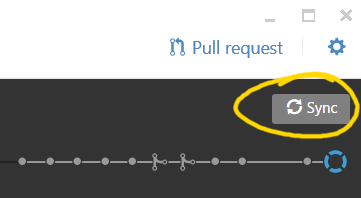




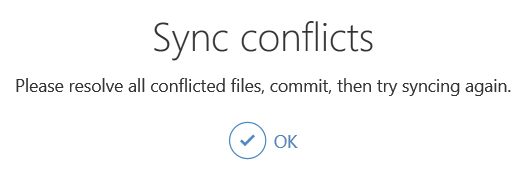
1. Now **commit** the local changes.



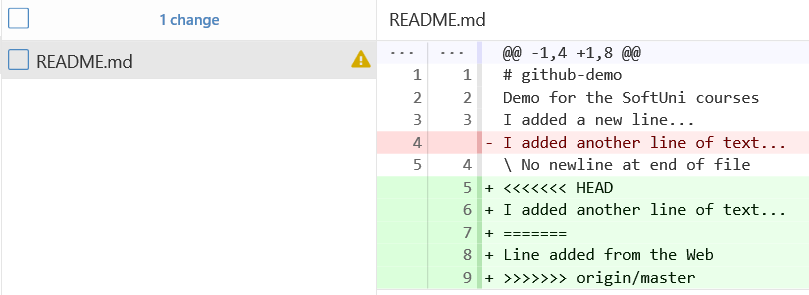
1. Try to **Sync** the local changes with the **remote repository** at GitHub:



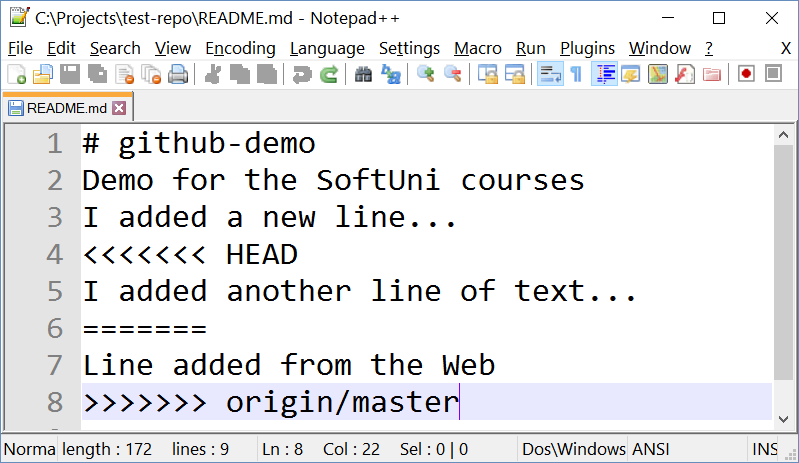
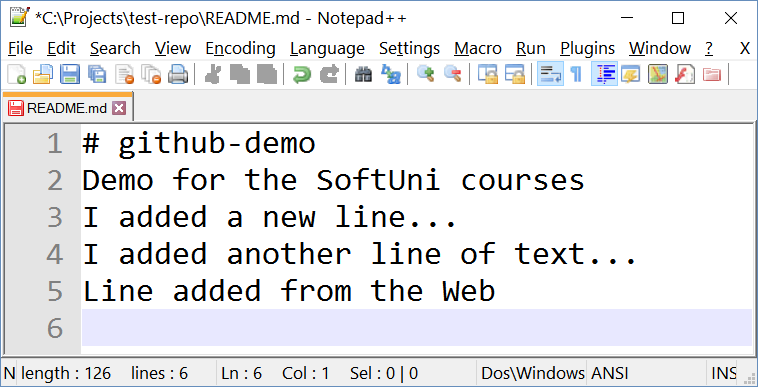
1. You will get **conflict notification**.



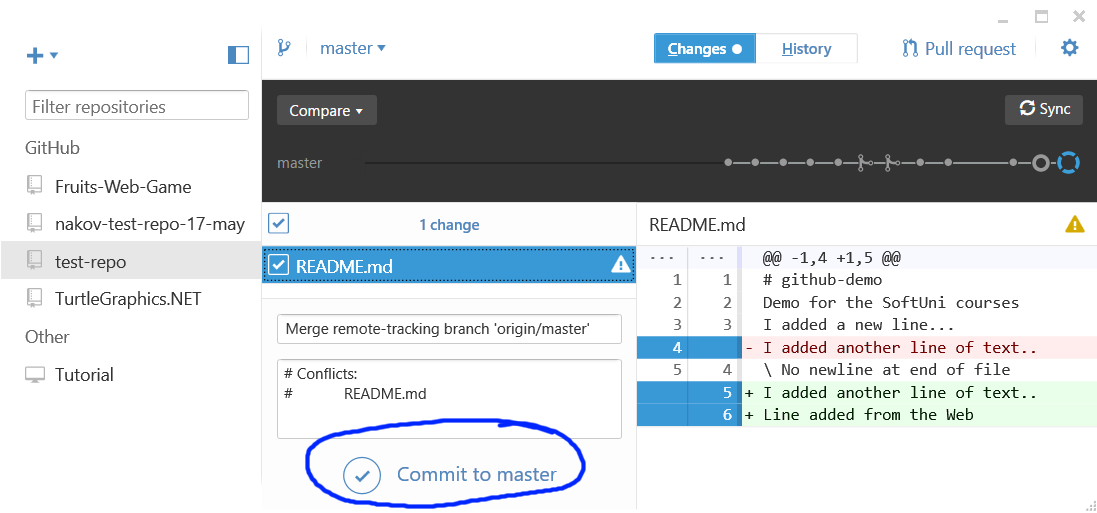
One of the files from the **local repository** will be **merged** with its newer version from the **remote repository**:



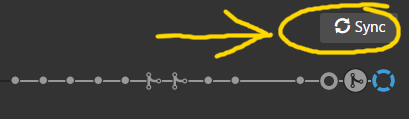
1. Now **resolve the conflict**. Edit the conflicting files and get then correctly merged. Remove all lines that point the locations of the merge conflicts (like <<<<<<< HEAD):

1. Now **commit the merged changes** (your local changes and your changed made from the Web):

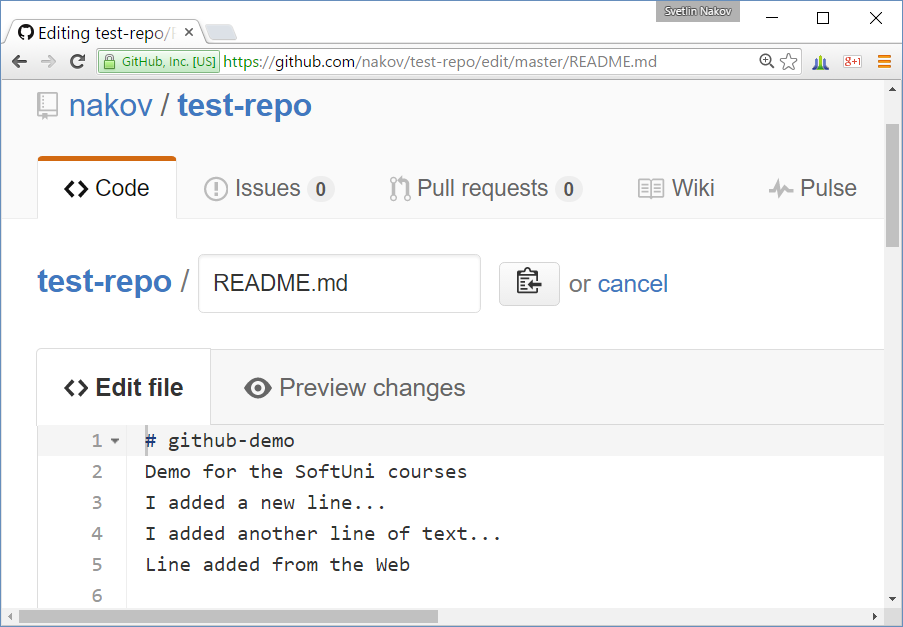


1. Now **sync again** to push your changes online to GitHub.



Now, the **sync should be successful** with **no conflicts**.

1. Finally, **check the changes** on the Web in your GitHub account:



## Debugging Exercise: Instruction Set

The goal of this exercise is to practice **debugging techniques** in scenarios where a piece of code does not work correctly. Your task is to **pinpoint the bug** and **fix it** (without rewriting the entire code). Test your fixed solution in the judge system: <https://judge.softuni.bg/Contests/203/Git-GitHub-Debugging-Searching-Exercises>.

### Problem Description

Write an **instruction interpreter** that executes an arbitrary number of **instructions**. The program should **parse the instructions**, **execute** them and **print the result**. The following instruction set should be supported:

* INC <operand1> – increments the operand by 1
* DEC **<operand1>** – decrements the operand by 1
* ADD <operand1> <operand2> – performs addition on the two operands
* MLA <operand1> <operand2> – performs multiplication on the two operands
* END – end of input

### Output

### The result of each instruction should be printed on a separate line on the console.

### Constraints

* The operands will be valid integers in the range [−2 147 483 648 … 2 147 483 647].

### Tests

|  |  |  |
| --- | --- | --- |
| **Input** | **Program Output (Wrong)** | **Expected Output (Correct)** |
| INC 0  END | 0  0  … (infinite) | 1 |
| ADD 1323134 421315521  END | 422638655  422638655  … (infinite) | 422638655 |
| DEC 57314183  END | 57314183  57314183  … (infinite) | 57314182 |
| MLA 252621 324532  END | 379219748  379219748  … (infinite) | 81983598372 |

## Debugging Exercise: Be Positive

The goal of this exercise is to practice **debugging techniques** in scenarios where a piece of code does not work correctly. Your task is to **pinpoint the bug** and **fix it** (without rewriting the entire code). Test your fixed solution in the judge system: <https://judge.softuni.bg/Contests/203/Git-GitHub-Debugging-Searching-Exercises>.

### Problem Description

A program is designed to take some **sequences of numbers** from the console, to **process them** as described below and **print** each obtained sequence.

### Input

* On the first line of input you are given a **count N – the number of sequences**.
* On each of **the next N lines** you will receive some **numbers surrounded by whitespaces**.

### Processing Logic

You need to check each number, if it’s **positive** – print it on the console; if it’s **negative**, add to its value the value of the next number and only **print the result if it’s not negative**. You only perform the addition once, e.g. if you have the sequence: -3, 1, 3, the algorithm is as follows:

* -3 is negative => add to it the next number (1) => -3 + 1 = -2 still negative => do not print anything (and don’t keep adding numbers, you stop here).
* The next number we consider is 3 which is positive => print it.

If no numbers can be obtained in this manner for the given sequence, print **“(empty)”**.

### Example

|  |  |  |
| --- | --- | --- |
| **Input** | **Expected Output** | **Comments** |
| 3  3 -4 5 2 123  -1 -1 3 4  -2 1 | 3 1 2 123  3 4  (empty) | (3) **(-4 + 5 = 1 > 0)** (2) (123)  **(-1 + (-1) < 0)** (3) (4)  **(-2 + 1 < 0)** |

### Output

Print on the console **each modified sequence on a separate line.**

### Constraints

* The **number N** will be an integer in the range [1 … 15].
* The **numbers in the sequences** will be integers in the range [-1000 … 1000].
* The **count of numbers in each sequence** will be in the range [1 … 20].
* There may be **whitespaces anywhere around the numbers** in a given sequence

### Tests

|  |  |  |
| --- | --- | --- |
| **Input** | **Program Output (Wrong)** | **Expected Output** |
| 3  3 -4 5 2 123  -1 -1 3 4  -2 1 | Exception… | 3 1 2 123  3 4  (empty) |
| 1  0 -2 2 -2 3 | Exception… | 0 0 1 |

## Debugging Exercise: Sequence of Commands

The goal of this exercise is to practice **debugging techniques** in scenarios where a piece of code does not work correctly. Your task is to **pinpoint the bug** and **fix it** (without rewriting the entire code). Test your fixed solution in the judge system: <https://judge.softuni.bg/Contests/203/Git-GitHub-Debugging-Searching-Exercises>.

### Problem Description

You are given a program that reads a n **numbers** and a **sequence of commands** to be executed over these numbers.

### Input

* The first line holds an **integer** n – the **count** of numbers.
* The second line holds **n numbers** – integers separated by space.
* Each of the next few lines hold **commands** in format: **“[action] [i-th element] [value]”**.
* The commands sequence end with a command **“stop”**.

### Commands

Commands are given in format **“[action] [i-th element] [value]”**. Elements are indexed from **1** to **n**.

The **action** can be **“multiply”**, **“add”**, **“subtract”**, **“rshift”** or **“lshift”**.

* The actions **“multiply”, “add”** and **“subtract”** have parameters. The first parameter is the **index** of the element that needs to be changed (in range [**1**...**n**]). The second parameter is the **value** with which we manipulate the element.
* The command **“lshift”** moves the first element last. E.g. “**lshift**” over {1, 2, 3} will produce {2, 3, 1}.
* The command **“rshift”** moves the last element first. E.g. “**rshift**” over {1, 2, 3} will produce {3, 1, 2}.

### Output

Print the values of the **n elements** after the execution of each command (except the last “**stop**” command).

### Constraints

* The **number** n will be an integer in the range [1 … 15].
* Each **element of the array** will be an integer in the range [0 … 263-1].
* The **number** i and the **number of commands** will be integers in the range [1 … 10].
* The **number value** will be an integer in the range [-100 … 100]. If the command is “**rshift**” or “**lshift**” there are no parameters.

### Tests

|  |  |  |
| --- | --- | --- |
| **Input** | **Program Output (Wrong)** | **Expected Output** |
| 5  3 0 9 333 11  add 2 2  subtract 1 1  multiply 3 3  rshift  stop | 3 0 9 333 11  3 0 9 333 11 | 3 **2** 9 333 11  2 2 9 333 11  2 2 **27** 333 11  11 2 2 27 333 |

## Debugging Exercise: Substring

The goal of this exercise is to practice **debugging techniques** in scenarios where a piece of code does not work correctly. Your task is to **pinpoint the bug** and **fix it** (without rewriting the entire code). Test your fixed solution in the judge system: <https://judge.softuni.bg/Contests/203/Git-GitHub-Debugging-Searching-Exercises>.

### Problem Description

You are given a **text** and a number count. Your program should search through the text for the letter '**p**' (ASCII code **112**) and print '**p**' along with countletters to its right.

For example, we are given the **text** "**phahah put**" and count = **3**. We find a match of '**p**' in the first letter so we print it and the 3 letters to its right. The result is "**phah**". We continue and find another match of '**p**', but there aren't 3 letters to its right, so we print only "**put**".

Each match should be printed on a separate line. If there are no matches of '**p**' in the text, we print "**no**".

### Input

* The first line holds the **text** to be processed (string).
* The second line holds the **number** count.

### Output

For each match, print the **matched substring** at separate line. Print "**no**" if there are no matches.

### Constraints

* The number countwill be in the range [0 ... 100].

### Tests

|  |  |  |
| --- | --- | --- |
| **Input** | **Program Output** | **Expected Output** |
| phahah put  3 | no | phah  put |
| No match  5 | no | no |
| preparation  4 | no | prepa |
| preposition  0 | no | P  p |

## Searching in Internet: Find a C# Trie Implementation

Find a **Trie data structure** implementation in **C#** in Internet.