# Exercises: Git and GitHub

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

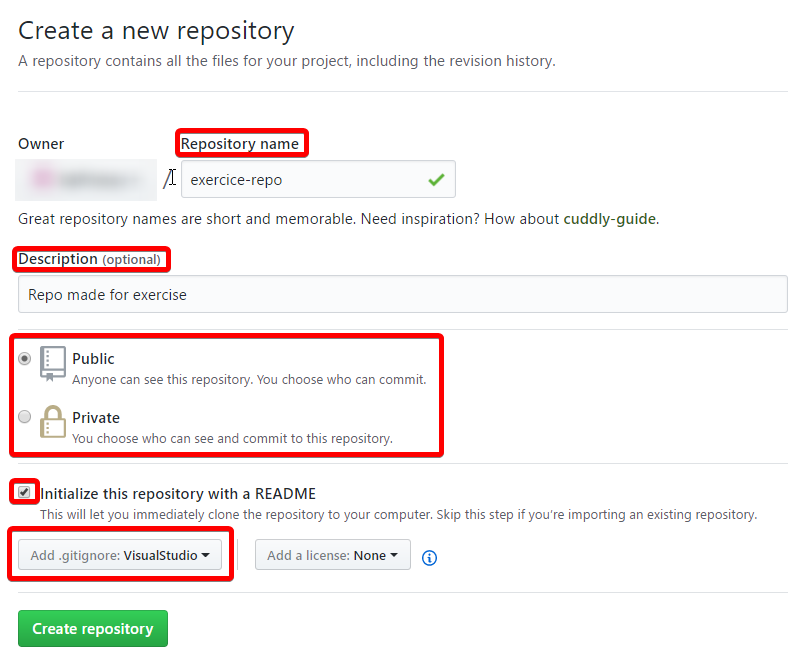
# TortoiseGit

## Upload Projects to 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. **Create a Remote Repository for Your Current Project**

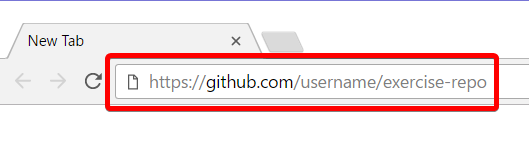
Go to <https://github.com/>. Click on the **New repository** button and you should see the following screen:



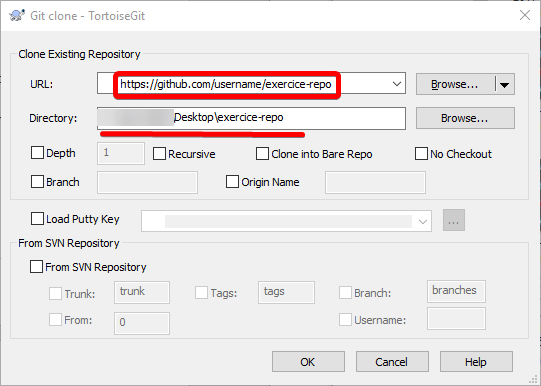
Under “**Repository name**” you can write the name of your new repository. You can also add **description** and modify the **visibility** of your repo. It’s good practice to initialize **README** with your repo. This way you can write more information about your project. Just check the option for creating **README** and GitHub will create the file for you.

1. **Clone** it on your device:

You can just copy the **URL** of your repo:



After that, paste the **URL** into **TortoiseGit** and it will **clone** the repo **locally** on your computer:

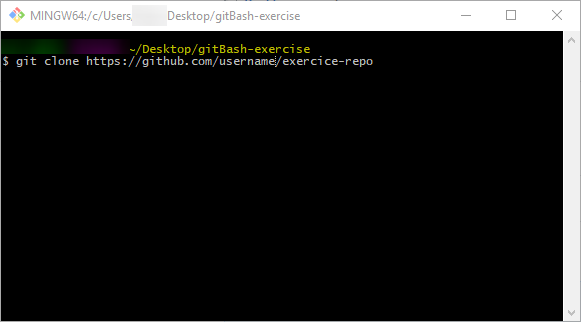


In the example the repo is cloned on the **Desktop**, but you can clone it in directory of your choice.

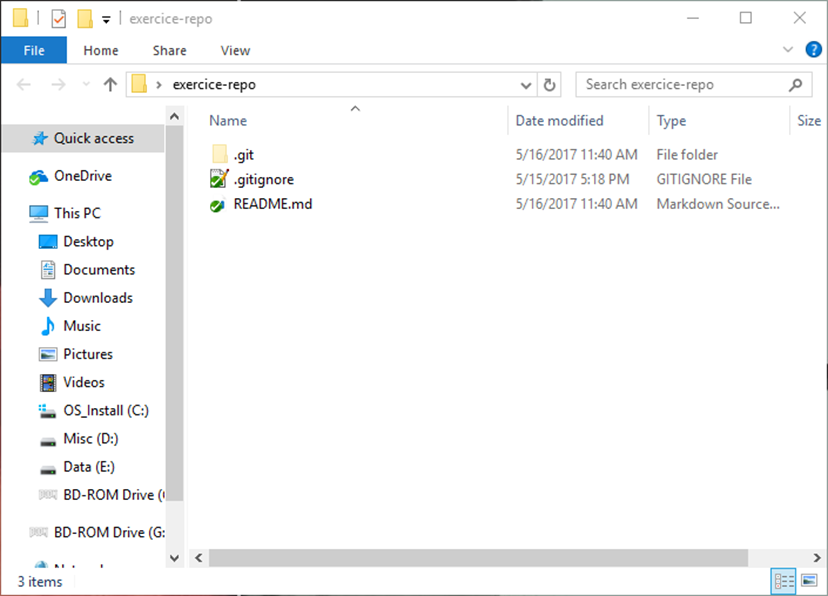
Note that **all (free) projects** you upload at GitHub will be **open-source** and will be accessible for **anyone** on the Internet, so **be careful** about **passwords** or **code** which you might **not** want to be **visible** by **someone** else. Additionally, you can read more about licenses [here](https://choosealicense.com/).

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

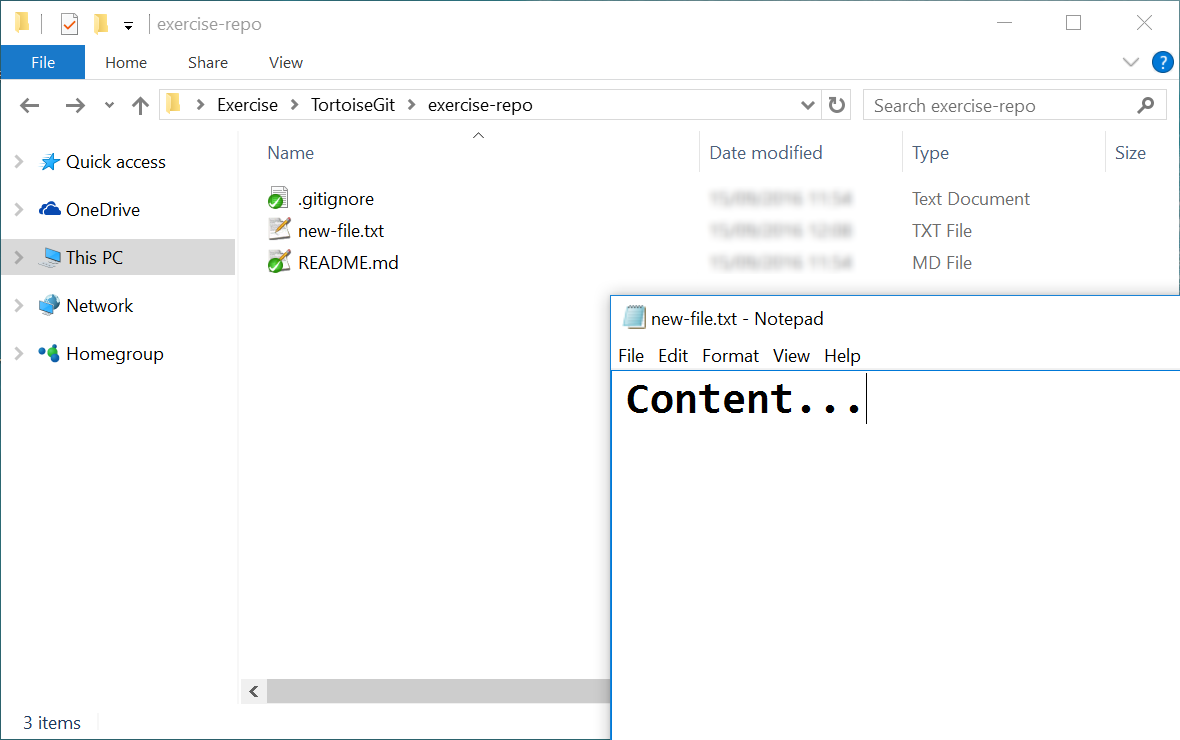
1. **Clone** the same repository again, but in a different directory (this time use **GitBash** -*"*git clone*"*):



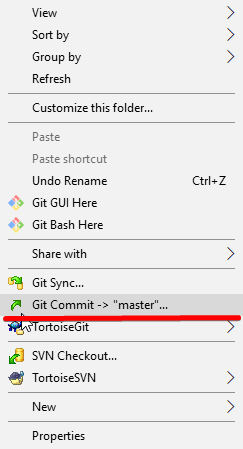
1. Return to the previous clone and **open it** in **Windows Explorer**. Add a new file in the directory:



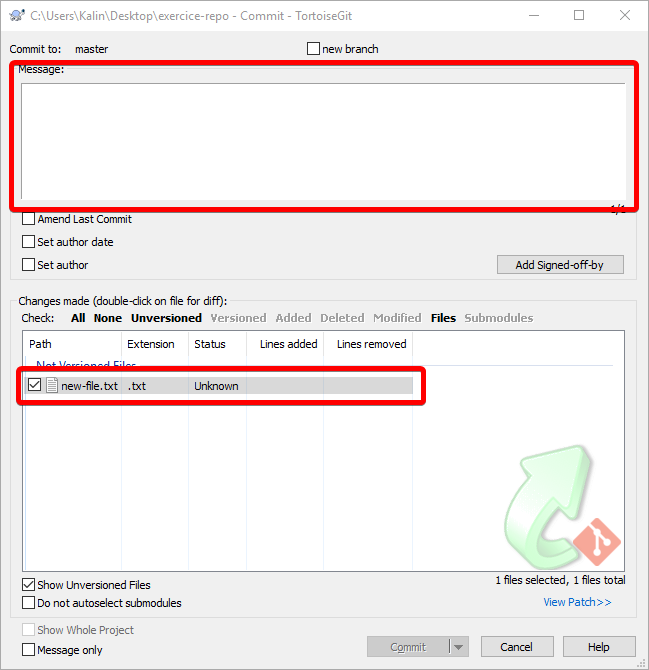
1. Make some **changes** new-file.txt:

****

1. **Commit** your local changes to your local repository. Click with the **right mouse** button and click on “**Git Commit**”

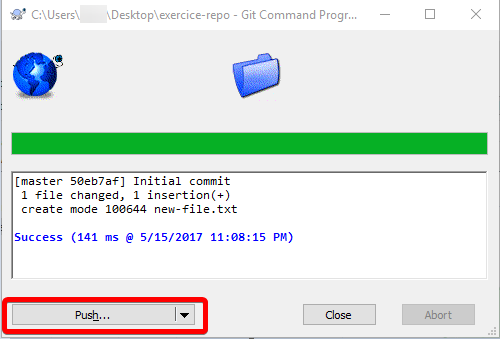


You should see the following window:



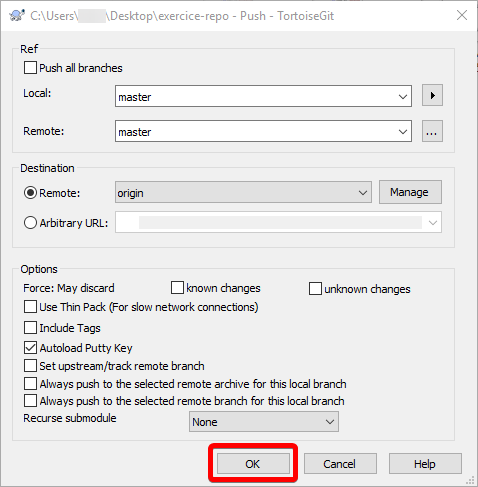
In the message section, write a **short summary** of your commit. It is good practice to always have **meaningful messages**. **Do not forget** to **add** your files in the **bottom part** of the window.

When you are done with these steps, you can click on **[Commit]** and you should see the **following window**:



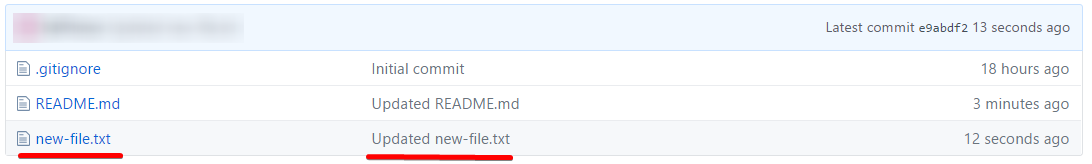
In this window, you can see **how** **many** fileswere **changed** and **how many insertions** and/or **deletions** were made. If you are satisfied with the information just click on **[Push]**.

1. **Push** your changes to the remote repository in GitHub:

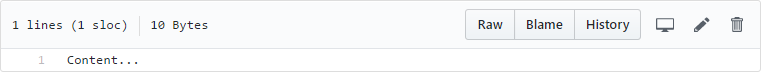


In this window, you can manage to which **branch** you are pushing your files, but we will talk about branches later in the exercise. For now, just click on **[OK]** and your file will be **pushed** to the **master branch**.

1. Check whether your changes are **online**:



Open **your GitHub repository** in your browser and click on **new-file.txt**. In there you should see the content, which you have written. On the last screenshot, you can see the **commit message** is written in the **second column** of the **commit** you have made. You can use this **column to get more information** about which files were changed and what has been changed. **That’s why it’s always good practice to write meaningful commit messages**. After clicking on your file, you should see something like this:

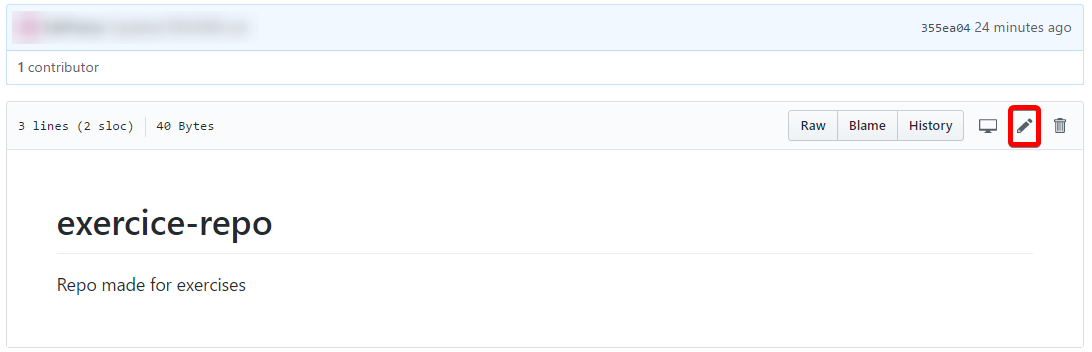


## Make Conflicts and Resolve Them

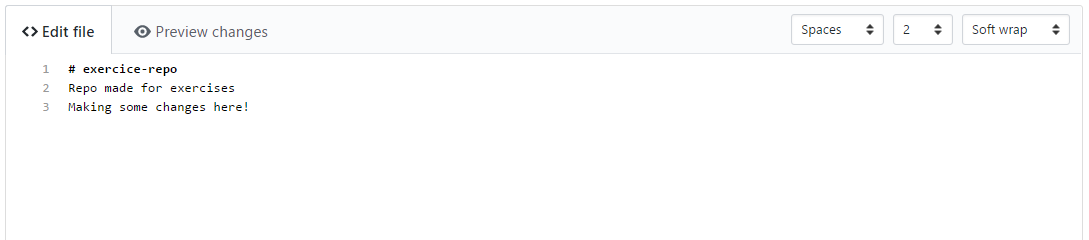
In your repo, you probably noticed a file named **README.md**. This file is used to write a **guide** for your **application** or just giving some **more information** about the project. The file uses a **markup language** called “[Markdown](https://en.wikipedia.org/wiki/Markdown)”. This language is primarily used for **formatting text** and **writing readme files**.

Now, let’s make a **conflict** in our **README.md**.

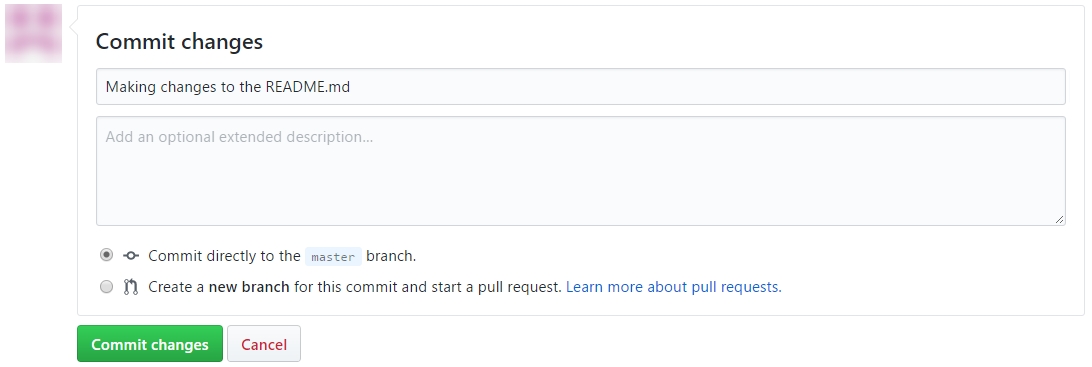
1. Open your GitHub account from your **Web browser**. Click on **README.md** and after that click on the **pencil** in the **upper right corner**:



You will see the **GitHub text editor**:

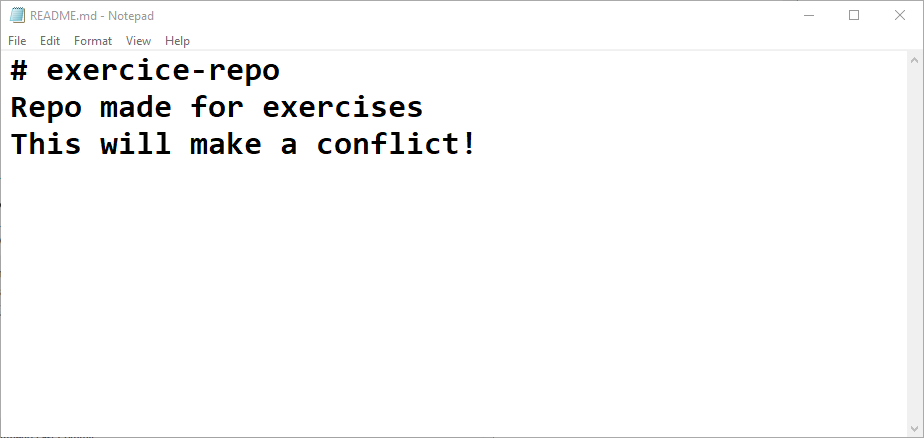


**Add some changes** to the file and **scroll down**. At the bottom of the page you will see this:

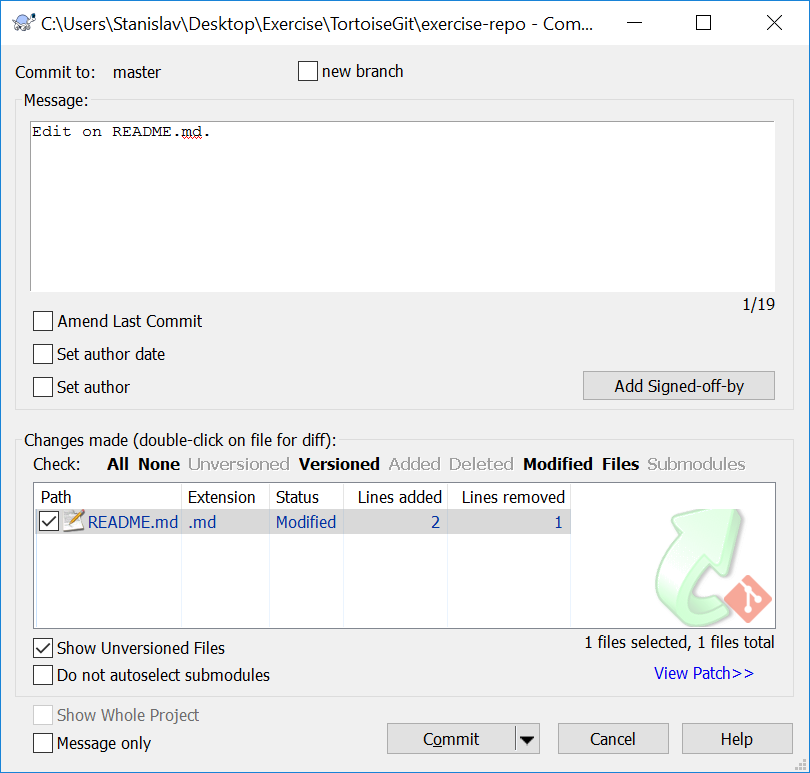


Here you can write your **commit message** and after you are done just click on **[Commit changes]**

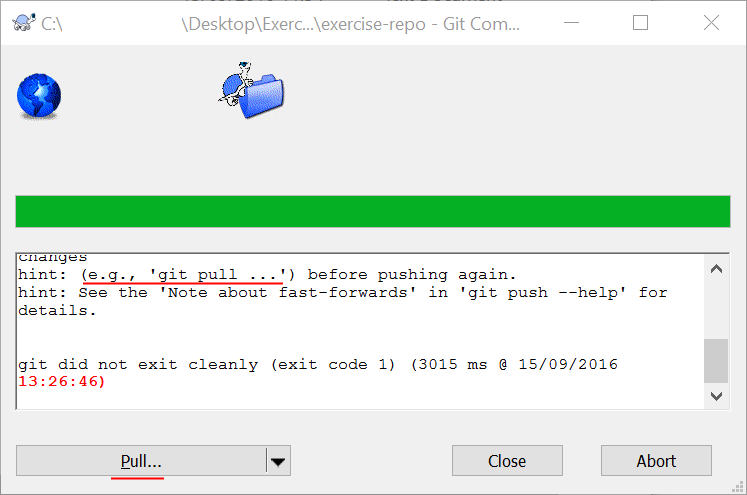
1. Open your local copy of the repository and open the **README.md** file (**do not pull the changes**). After that just add some different text to the file:



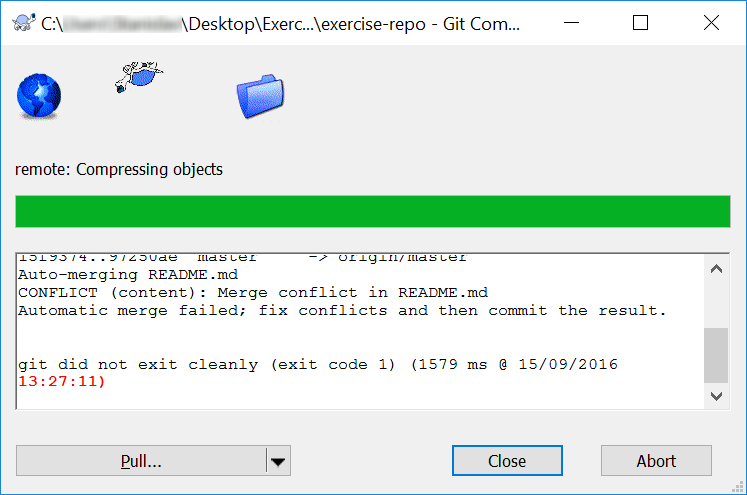
1. Now **commit** the local changes using **TortoiseGit**:



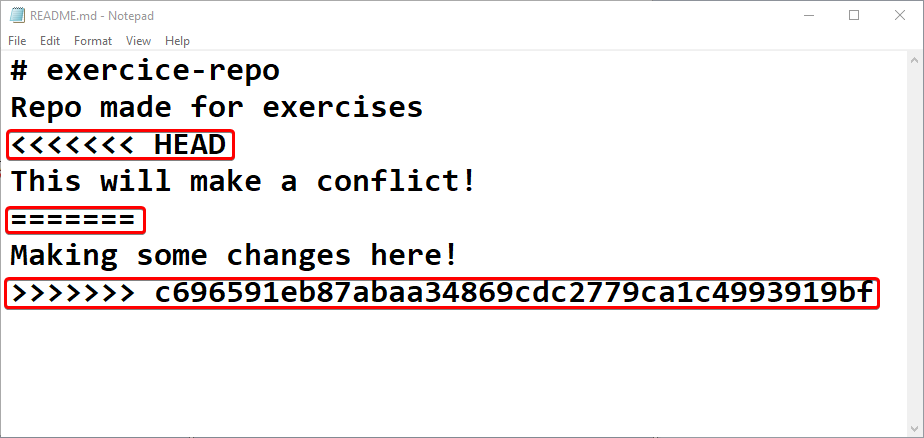
1. Try to **push** the local changes to the **remote repository**. The operation will **fail**, since the remote repository is **updated** and the local one is **not**:



1. After the pull **TortoiseGit** will **try** to pull and merge but it will **fail**, so we have to merge **manually**.



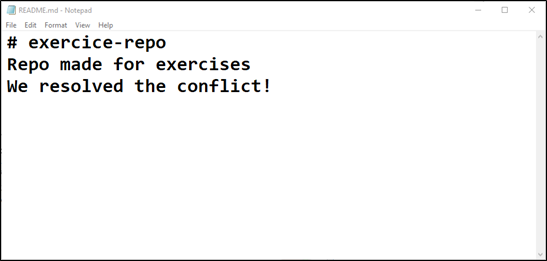
1. Now **resolve the conflict**:



(**<<<<<<< HEAD**)marks the **beginning of your local variation of the file**; (**========**) **separates the local version** **from the version in the repo**. (**>>>>>>>**) **marks the end of the file** and after that is written the **number of the commit**. To resolve the conflict, you have to delete all of the three special marks and choose which version of the file to keep. You have 3 options:

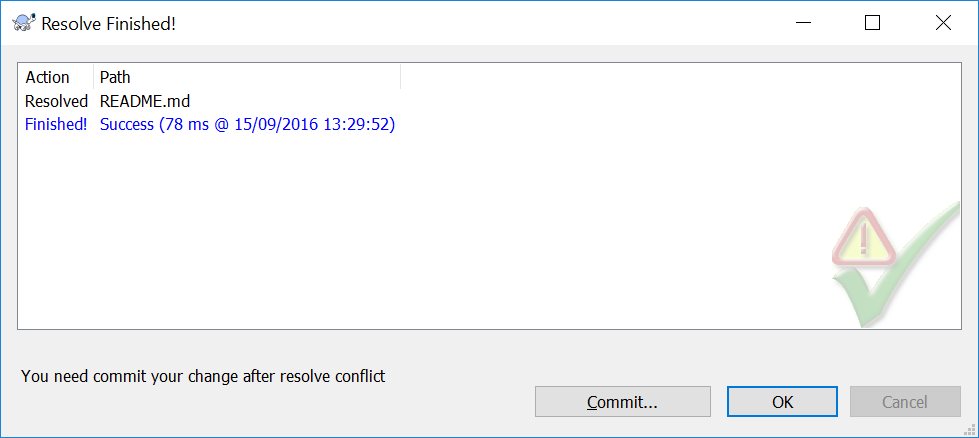
* You can **delete** “**This will make a conflict!**” or “**Making some changes here!**”;
* You can **keep both files** and **delete only the marks**;
* You can write third completely different sentence

In the screenshot is chosen the **third** option and we are writing **new message**:

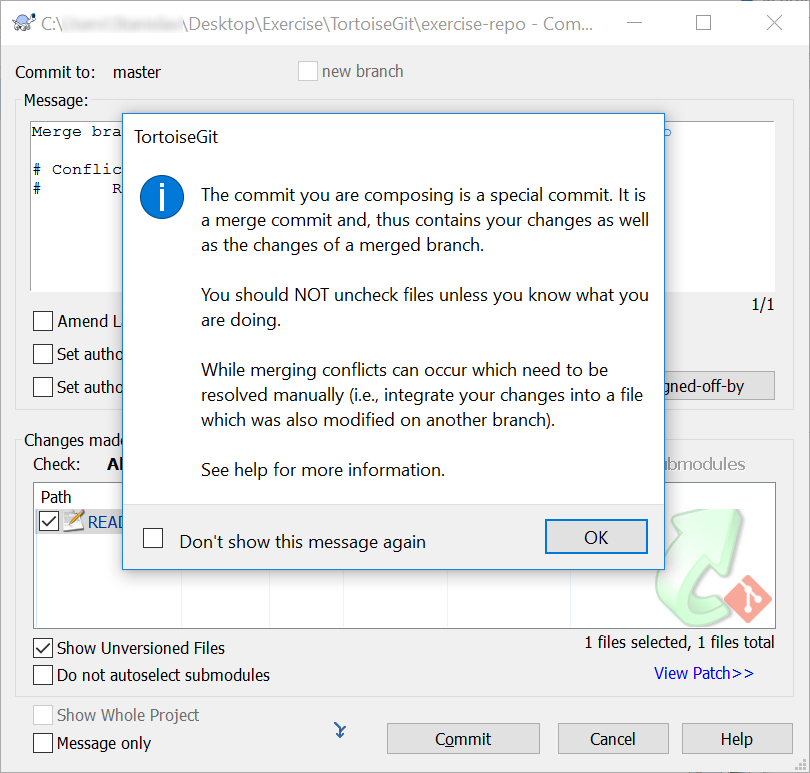


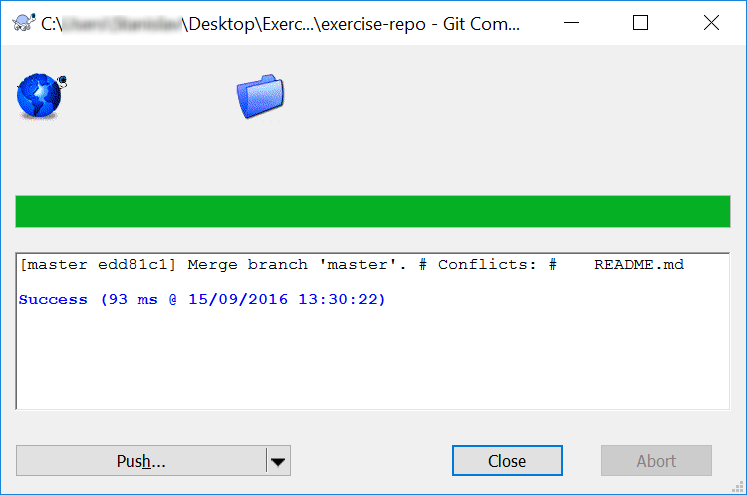
In the examples, we are using **Notepad** for editing files, but most **IDEs** (Visual Studio, Eclipse, IntelliJ, WebStorm and others)have **Git integration** and will show you the **conflict differences**.

1. Resolve current file with **TortoiseGit** -> **Resolve**

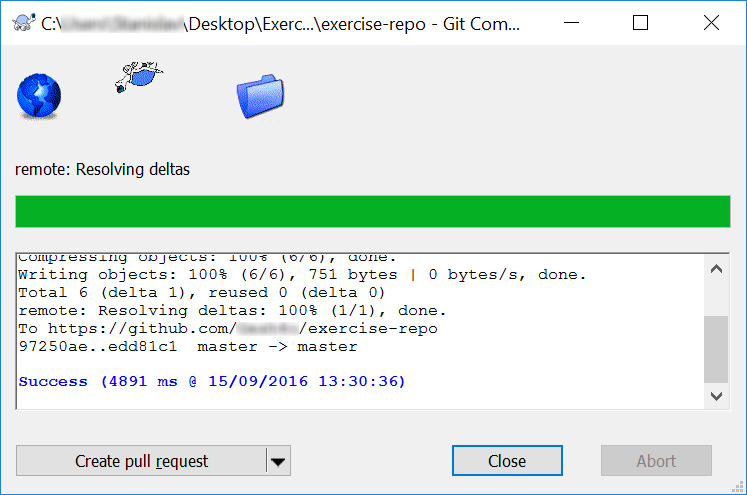


1. **Commit the merged changes** (your local changes and your changed made from the Web):

****

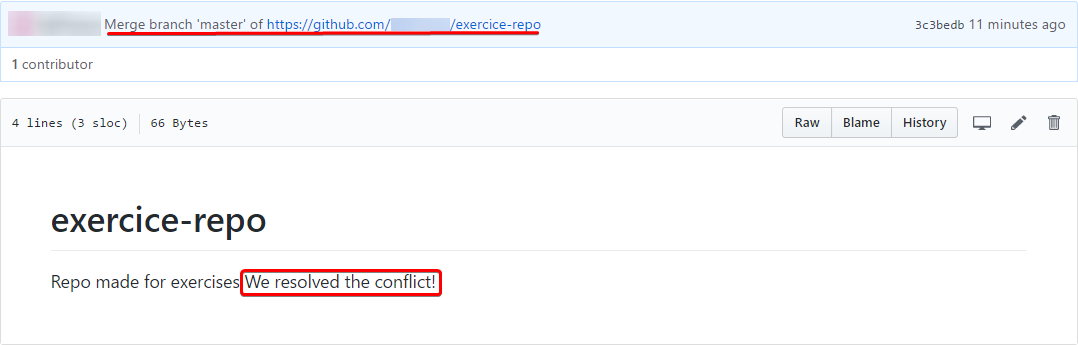


1. Now **push again** to push your changes online to GitHub.



Great, the **push should be successful** with **no conflicts**!

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



It’s worth noting that when you merge, a **commit** is made with a **special description**.

## Create a Branch and Merge Changes

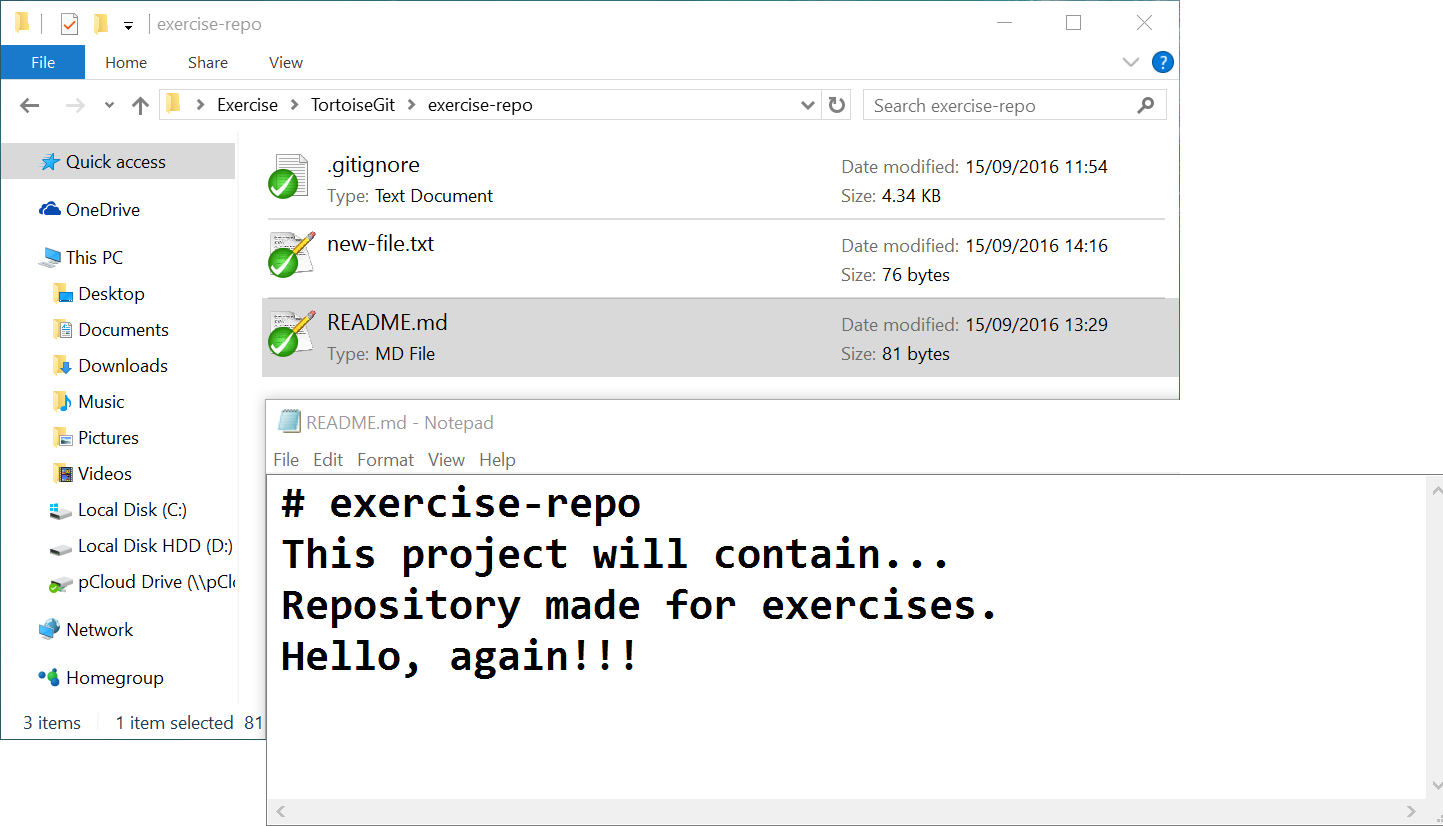
Branches are very **useful when many people are working on the same project**. Such cases are **premises for lots and lots of conflicts**. With **branches** the developers have the possibility to work on **separate parts of the project** without causing conflicts. When one developer finishes the feature that they are working on – **the branch can be merged with the main one**.

1. Create **branch**. (Here the branch name is: **develop**)

|  |  |  |
| --- | --- | --- |
|  | 🡺 |  |

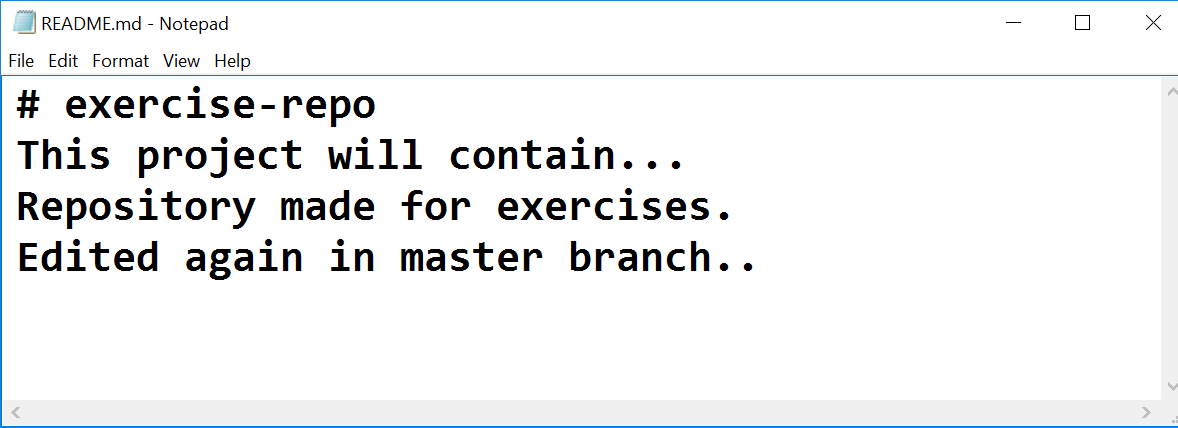
1. **Switch** to that branch.   
   

|  |  |  |
| --- | --- | --- |
|  | 🡺 |  |

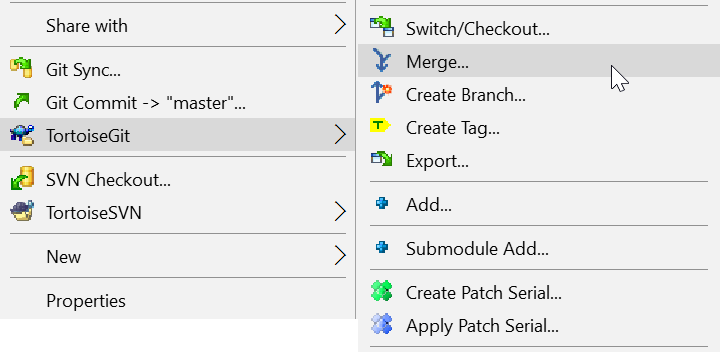
1. Make some **changes**. **Edit** one of the files in your repository.  
   ****
2. **Commit** them as before.
3. **Switch** to the main branch.



1. Make some changes to the main branch (on the same file you edited before). **Commit** them and then **push**.

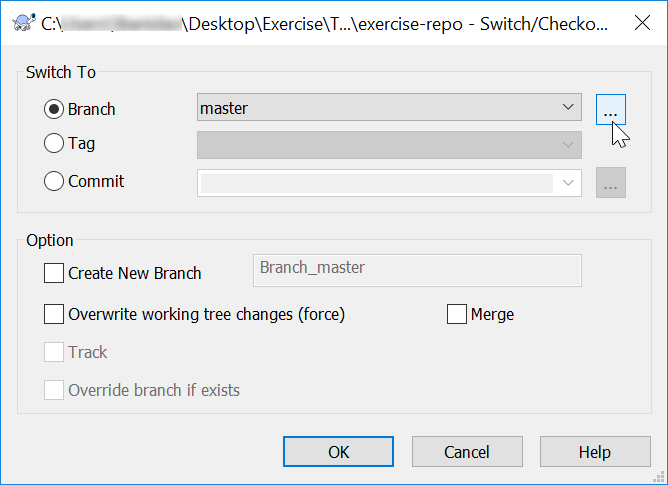


1. **Merge** with previous branch (in this case - **develop**).

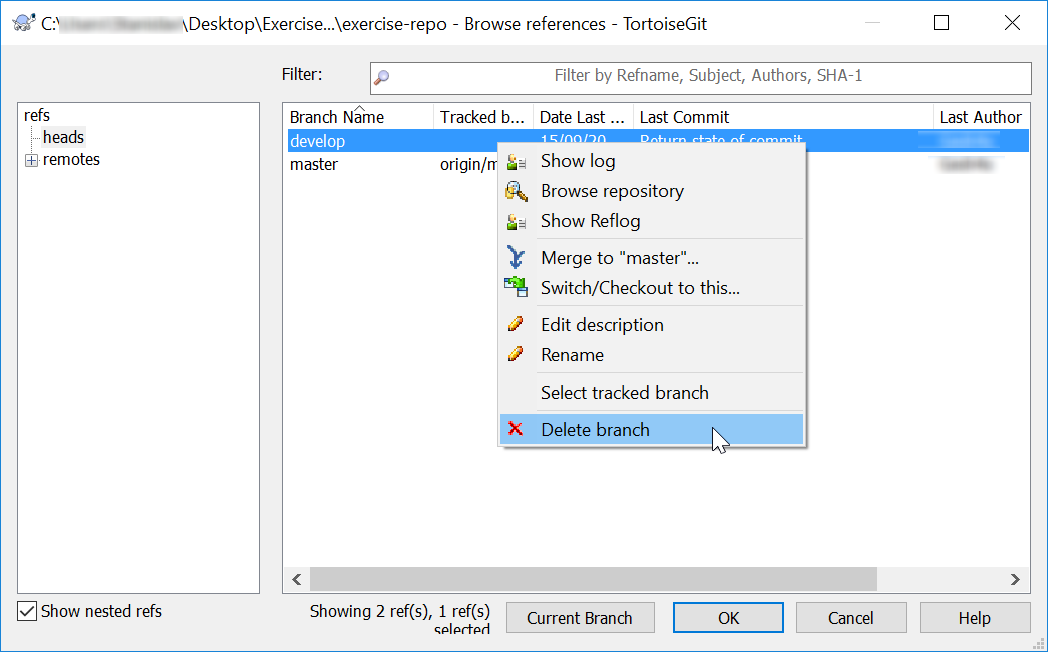


|  |  |  |
| --- | --- | --- |
|  | 🡺 |  |

1. **Resolve** the new conflicts and commit.
2. **Delete** the newly created branch.
   1. Use **TortoiseGit** -> **Switch/Checkout…**



* 1. Right click on the **hovered** element above and window like this must appear:



* 1. You can now **delete** your branch and **commit** your changes.

1. **Update** the remote repository.

# GitBash

**GitBash** is the console client for **GitHub**. Most developers use it because it **gives more control** **and executes only the commands, which you typed in**. Most graphic clients like TortoiseGit **execute some commands in the background** and that can be a problem in bigger projects.

## Upload a Few Projects at GitHub

**\*** If you have already cloned your repository with **GitBash** you can safely skip this step.

1. **Clone** the same repository that you worked with for the previous tasks it on your device:

* Use "git clone" command.

1. Open the project files in **Windows Explorer**.
2. Make some **changes** in your favorite text editor:
3. **Commit** your local changes to your local repository.

* Use "git add" command. You can write "git add ." as a command in **GitBash**. This command **prepares** (**stages**) all **new** and **modified** files for commitment.
* Use "git commit" command.

1. **Pull** then **push** your changes to the remote repository in GitHub:

* Use "git pull" command.
* Use "git push" command.

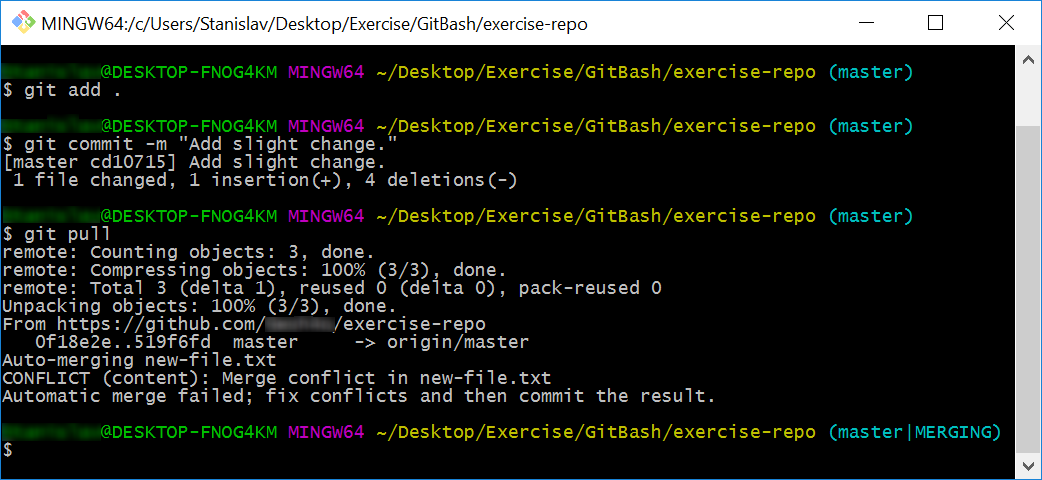
1. Check whether your changes are online.

## Make Conflicts and Resolve Them

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

1. Make some **changes** in your working directory, e.g. edit the file README.md.
2. **Don’t commit** and **don’t push** your changes yet.
3. Open your **GitHub** account from your **Web browser** or **TortoiseGit**. Make some changes on the same file.
4. Now **commit** them.
5. Try to **update** the local changes with the **remote repository** at GitHub:

* Use "git pull" command.

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

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

1. **Resolve the conflict**. Edit the conflicting files and get them correctly merged. Remove all lines that point the locations of the merge conflicts (like <<<<<<< HEAD):
2. **Commit the merged changes** (your local changes and your changed made from the Web/TortoiseGit):
3. **Sync again** to push your changes online to GitHub.

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

1. Finally, **check the changes** on the Web in your GitHub account or sync your TortoiseGit local repo.

## Create a Branch and Merge Changes

1. Create **branch**.

* Use "git branch branchName" command.

1. **Switch** to that branch.

* Use "git checkout branchName" command.

**\* Note** that the previous **2 steps** can be done also with the **following command**:

"git checkout -b branchName"

1. Make some **changes**.
2. **Commit** your changes.
3. **Switch** to the main branch.

* *Refer to step 2.*

1. Make some changes to the main branch.
2. **Merge** with previous branch.

* Use “git merge branchName”

1. **Resolve** the new conflicts (if any).

* Modify the file to resolve the conflicts
* Use “git add filename” and “git commit”

1. **Attempt to merge again** (**only** if there were **conflicts** in step 8).

* Use “git merge branchName”

1. **Delete** the newly created branch.

* Use “git branch -d branchName” command.

1. **Update** the remote repository.
   * Use “git push --all --prune” command.