Contents

Please keep in mind that the remaining parts of this documentation are just for help to fulfill the requirements stated above, and they may not contain all the necessary information. You can find further info on the internet (git/stylecop/doxygen official help pages, google). Some links can be found in the various parts of this documentation.

2021. Spring changelog:

- At the end of the document information copied from the GitStats FAQ
- Update StyleCop/FxCop because of FxCopAnalyzers ⇒ NetAnalyzers (dotnet5) transition ("Code analysis" chapter)
- Prog3: small changes, the only big one: "non crud operation" must use BOTH join AND groupby

GIT QUICKSTART

BASICS OF GIT

Git is a distributed version control/source control system that allows us to centrally manage our source code and the changes to this source code (record everything: which file was modified when and how and by whom). This is achieved by not storing every state (every version) of every file, but instead only the difference/increment between file revisions (those differences are called "commits").
The distributed nature of Git is shown if we compare it with other "traditional" source control systems (SVN, CVS): in Git, the changes of the source code are not only stored in a central storage, but also in a local storage as well. According to this, in Git, we have to differentiate between the *local repository* and the *remote repository*. The third storage is the local current state of the development files on the hard drive, this is the *working copy*.
The basic Git operations are listed in this table:

ADD	Add files from the woking copy to the source control / add modified files into the list of files affected by the next commit
COMMIT	Store the modifications (content changes, ADD, file deletion) of the working copy into the local repository
PUSH	Upload the commits of the local repository into the remote repository
FETCH	Download the commits of the remote repository into the local repository, without changing the working copy
MERGE	Merging branches (e.g. after a FETCH, there are two branches in the local repository: our own development branch and the branch downloaded from the remote repository). If there is contradiction (CONFLICT) between the branches (different modifications to the same file), then we have to fix these conflicts manually. In addition to this, the BRANCH / CHECKOUT commands can be used to create new branches, every branch is a separated version of the code, and also very often every new feature is implemented in its own branch (see "A successful git branching model", nvie.com; THIS IS NOT REQUIRED FOR US)
PULL	FETCH + MERGE

A typical development cycle looks like the following:

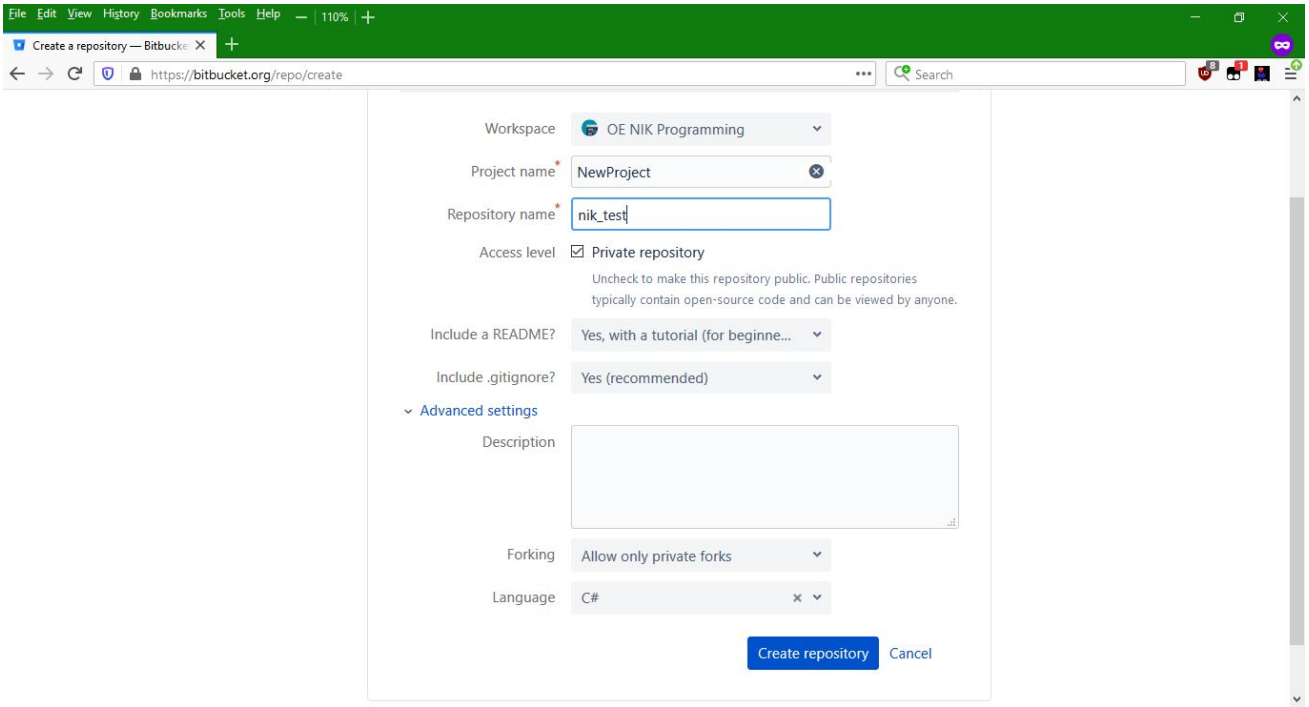
1. At the start of the day, the developer downloads the earlier modifications of the remote repository using a PULL command; into the local repository
2. The occasional conflicts have to be solved manually
3. Work starts on a new feature / bugfix (the new feature usually goes with the creation of a new branch)
4. At the end of a single work-process the modifications are stored with a COMMIT

command
(this can even be one commit per file, but it can be 10 files, depending on the definition of „one work-item”. Basically the principle is to work with multiple small commits, and descriptive commit messages. The basic rule is that one commit fixes only one thing / adds only one feature to the code: „If you have to put the word "and" or "also" in your summary, you need to split it up”, this is the golden rule).

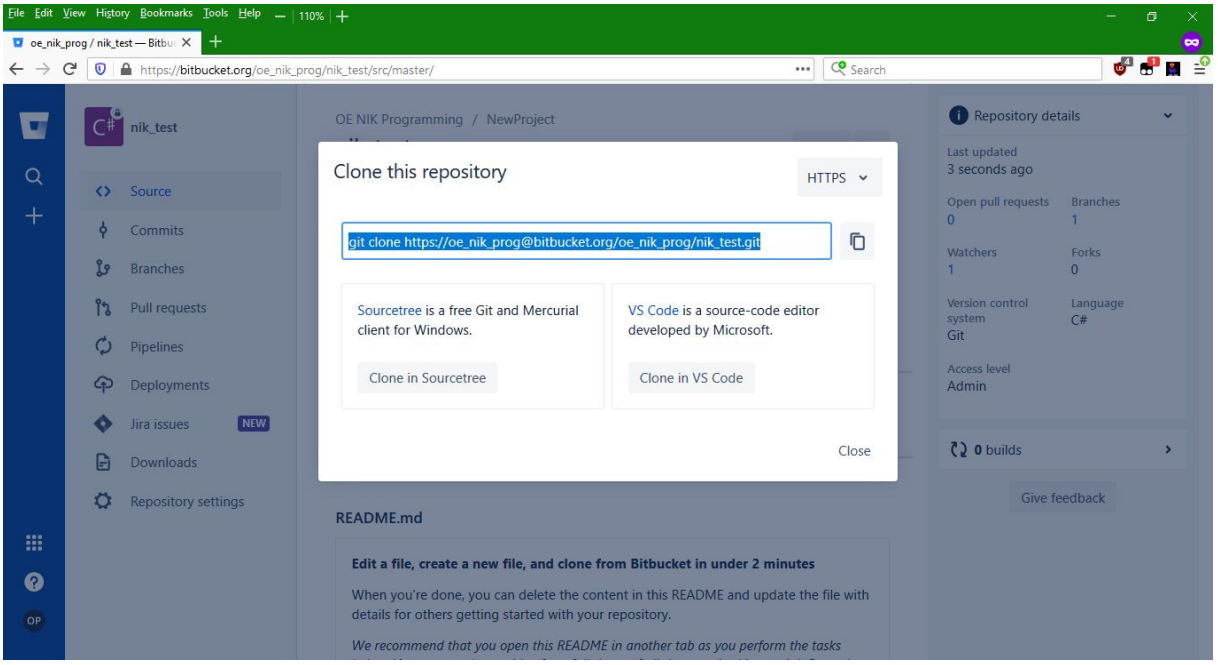
5. At the end of the day / at the end of the feature or bugfix, once again PULL, and the conflicts have to be solved manually
6. PUSH, and we can move on to the next ticket/work-item/bugfix/feature …

Creation of the bitbucket repo

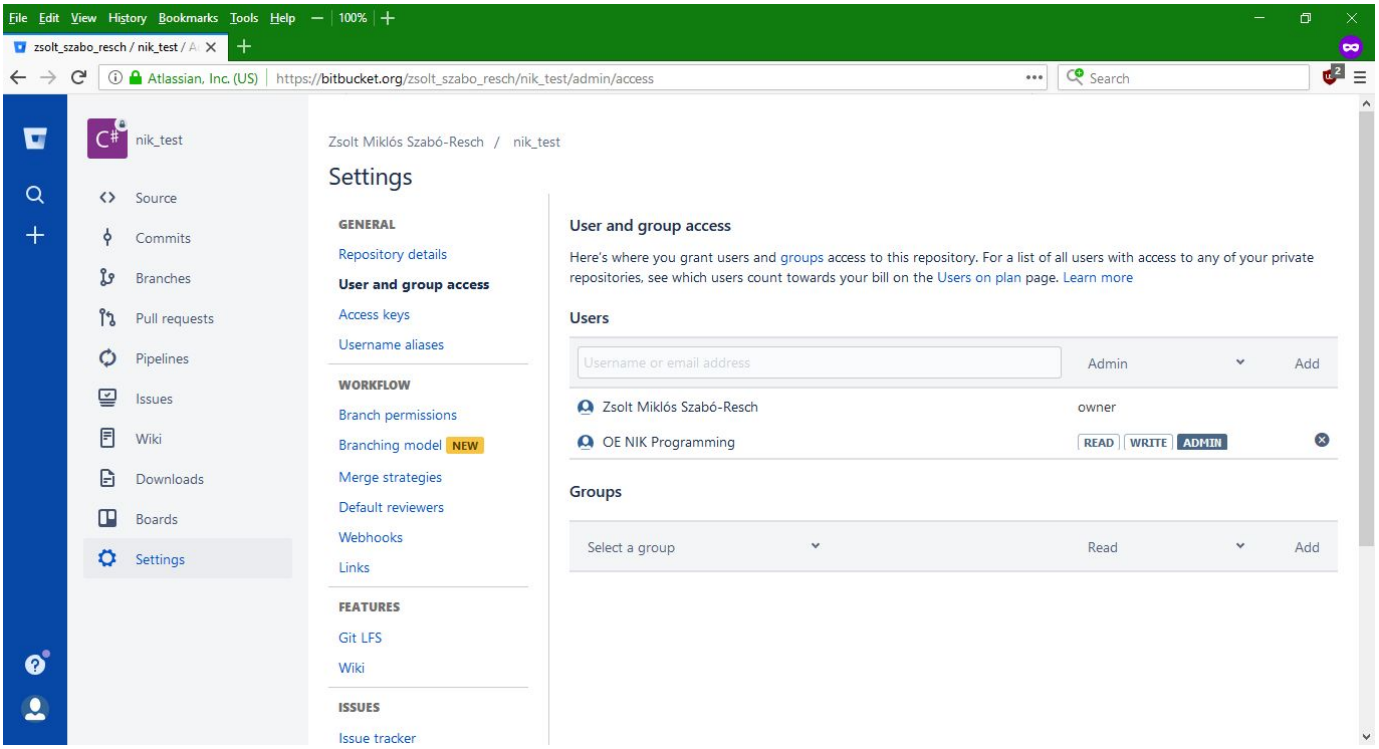
1. The GIT serverside system, that we will use during the semester is: bitbucket.org. For our aims, the github.com could be good as well, but Bitbucket has better API and permission management. Also, any private GitLab installation could be usable as well, but for the univeristy project works, you have to use the bitbucket website.
2. Bitbucket registration, activation, login, then we can create the new repository. Default workspace, and select "new project". IMPORTANT: **OENIK_SUBJECT_YEAR_SEMESTER_NEPTUN** + Private Repository + Include a Readme: YES, so that the GIT repository itself is also created. You **must** use the recommended repository name or otherwise the gitstat verification tool will not work.



3. After creation, we get to the „Source” part of the project site, in the top right corner there is a "Clone" button that shows the repo's git address: https://oe_nik_prog@bitbucket.org/oe_nik_prog/nik_test.git

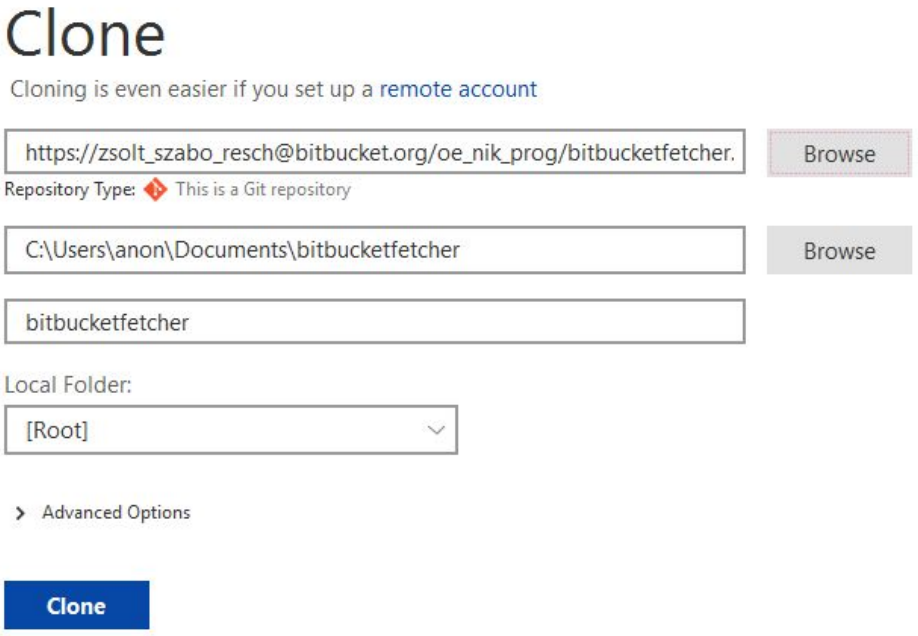


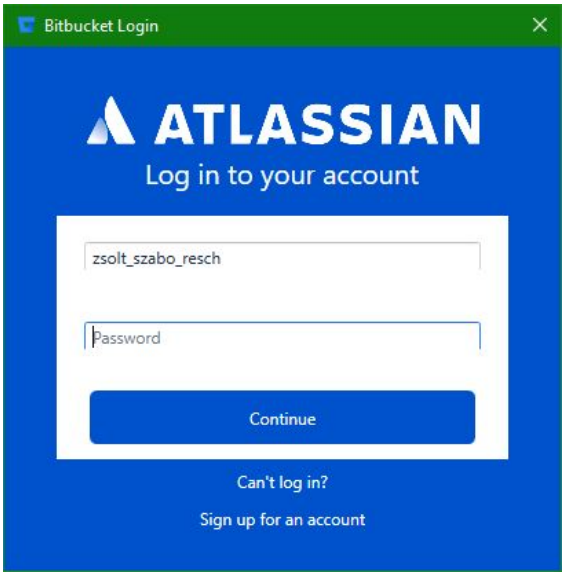
4. In the left-hand side, click on the "Repository Settings" menuitem, then „User and Group Access” should be used to add permissions to other users.
It is **IMPORTANT** to give ADMIN persmissions for the **oe_nik_prog** user (e-mail address: **nikprog@iar.nik.uni-obuda.hu**).


File Edit View History Bookmarks Tools Help
zsolt_szabo_resch / nik_test /
Atlassian, Inc. (US) https://bitbucket.org/zsolt_szabo_resch/nik_test/admin/access
Search
nik_test
Source
Commits
Branches
Pull requests
Pipelines
Issues
Wiki
Downloads
Boards
Settings
Zsolt Miklós Szabó-Resch / nik_test
Settings
GENERAL
Repository details
User and group access
Access keys
Username aliases
WORKFLOW
Branch permissions
Branching model NEW
Merge strategies
Default reviewers
Webhooks
Links
FEATURES
Git LFS
Wiki
ISSUES
Issue tracker
User and group access
Here's where you grant users and groups access to this repository. For a list of all users with access to any of your private repositories, see which users count towards your bill on the Users on plan page. Learn more
Users
Username or email address
Admin
Add
Zsolt Miklós Szabó-Resch
owner
OE NIK Programming
READ WRITE ADMIN
Groups
Select a group
Read
Add

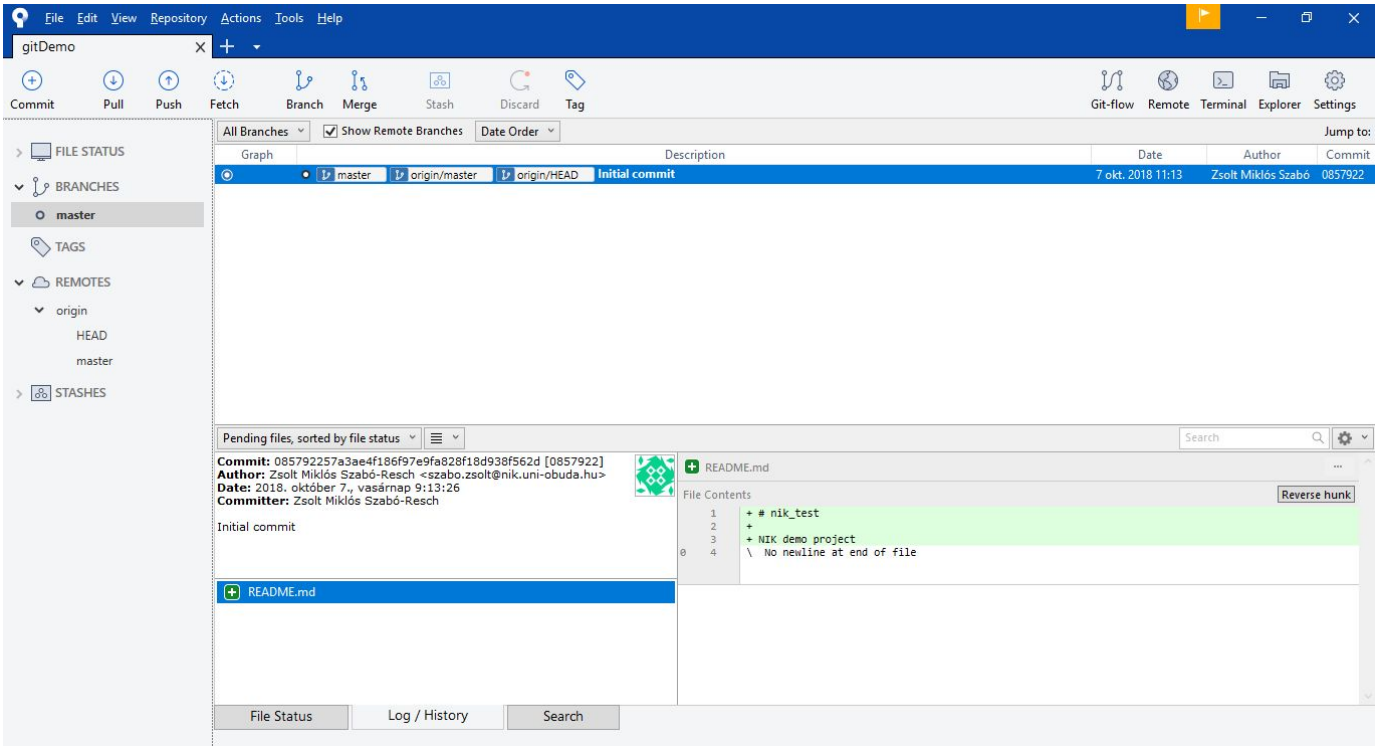
Using Git

1. This section shows the GIT usage using the SourceTree GUI client; naturally other GIT clients are allowed too (TortoiseGit, Git commandline). GIT clients integrated into the IDE (NetBeans, Visual Studio) can work too, but rather it is suggested that an external (and better) GIT client is used instead (the "true professionals" should naturally only use the commandline client :)).
2. Download SourceTree (https://www.sourcetreeapp.com/), install (it will ask for the BitBucket user), then using the Clone button, copy the git repository's address. In the pop-up, enter your BitBucket login data, so the text "This is a git repository" is shown, and the lower CLONE button is enabled.
   If you have authentication problems, removing the saved passwords can help, this is a known current bug. Remove the file below, and inside SourceTree, go to Tools/ Option/ Authentication, and then remove all bitucket.org accounts. Inside Windows, go to Start Menu/ Control Panel/ User Accounts/ Manage Your Credentials, and then remove all saved bitbucket.org accounts.
   The file to delete: c:\Users\<USER>\AppData\Local\Atlassian\SourceTree\passwd



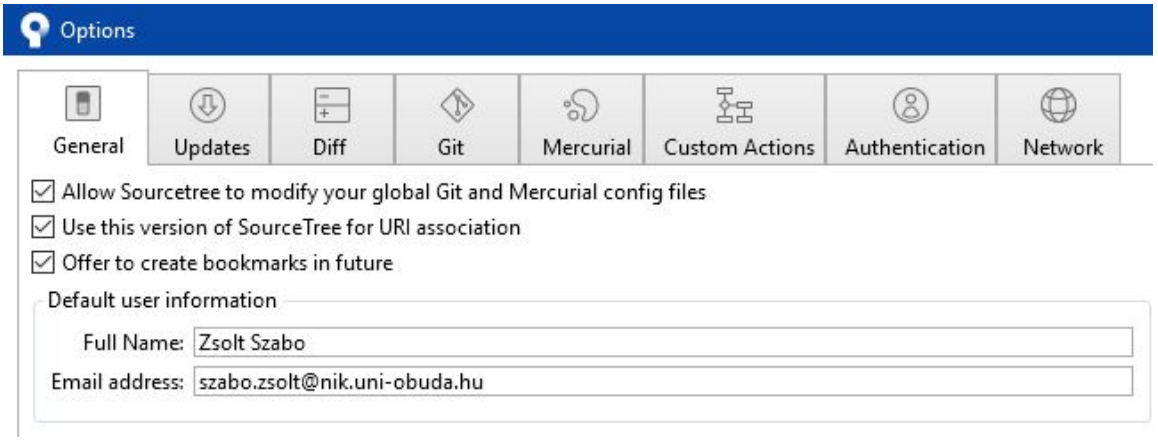


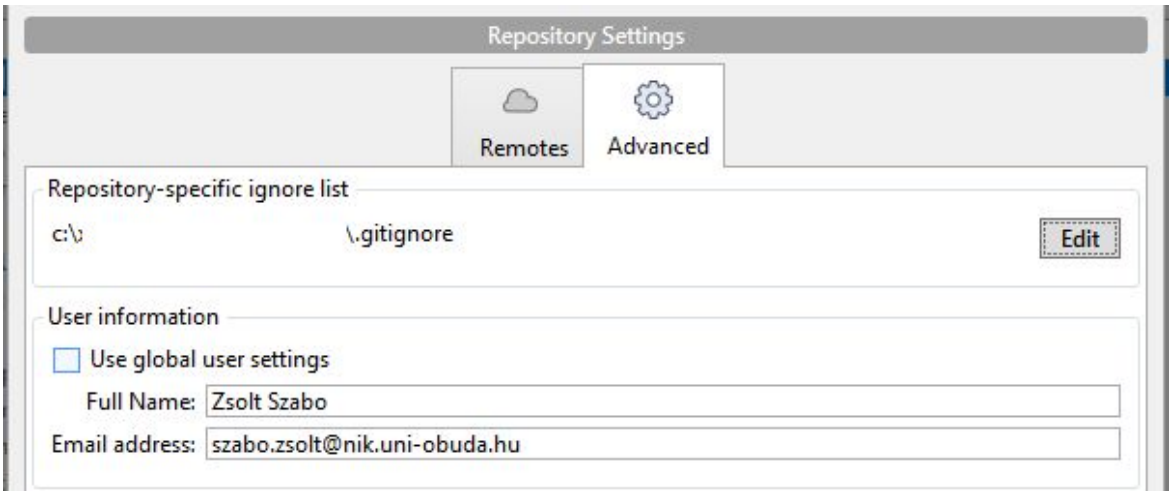
3. In the following screen, on the left side we can see the local and the remote BRANCHes (initially there is a remote origin/master branch, and a locel master branch associated with it). In the middle we can see the list of modifications (COMMITS), down there is the staging area (list of modifications inside a commit), and the button bar above shows the possible operations:
    a. Commit = Staging management (selecting the active modifications that will go into the next commit), and actual commit (saving the modifications into the local branch).
    b. Pull = Download the commits from the remote branch, then merge the commits into the local branch
    c. Push = Upload the commits of the local branch into the remote branch
    d. Fetch = Download the commits from the remote branch, no merge
    e. Branch = Create a new code branch
    f. Merge = Merge a different code branch into the current one

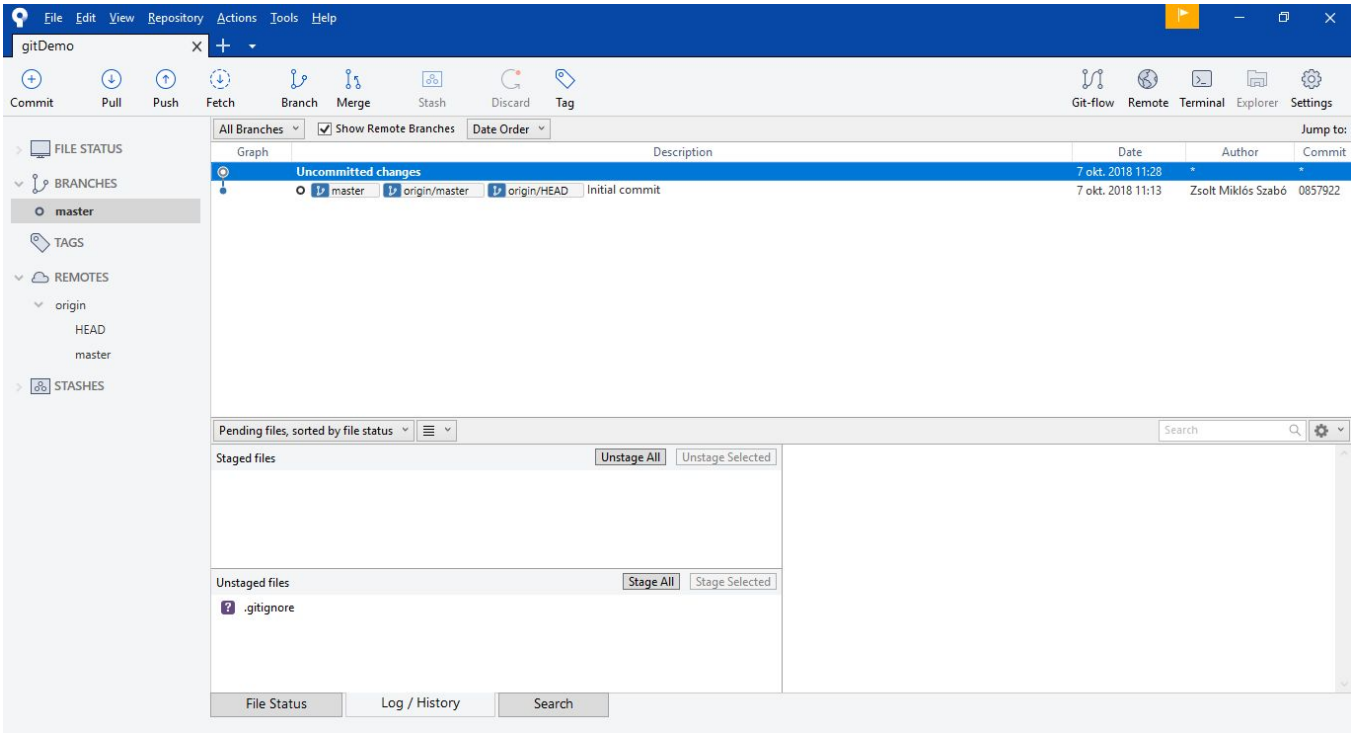


4. From the https://github.com/github/gitignore address, we should download the VisualStudio.gitignore file. Using our favourite text editor, we should create a **.gitignore** file (or edit - yes, there is nothing before the dot in the file name!), and copypaste the contents of the downloaded gitignore file.
IMPORTANT: In the .gitignore file, comment out the lines refering to MDF/LDF files (the database is also part of our repository!)

5. SourceTree sees the modifications (as "Uncommitted changes"). Before creating a Commit, use the Repository or SourceTree settings to set the author data that is used in the Commit messages







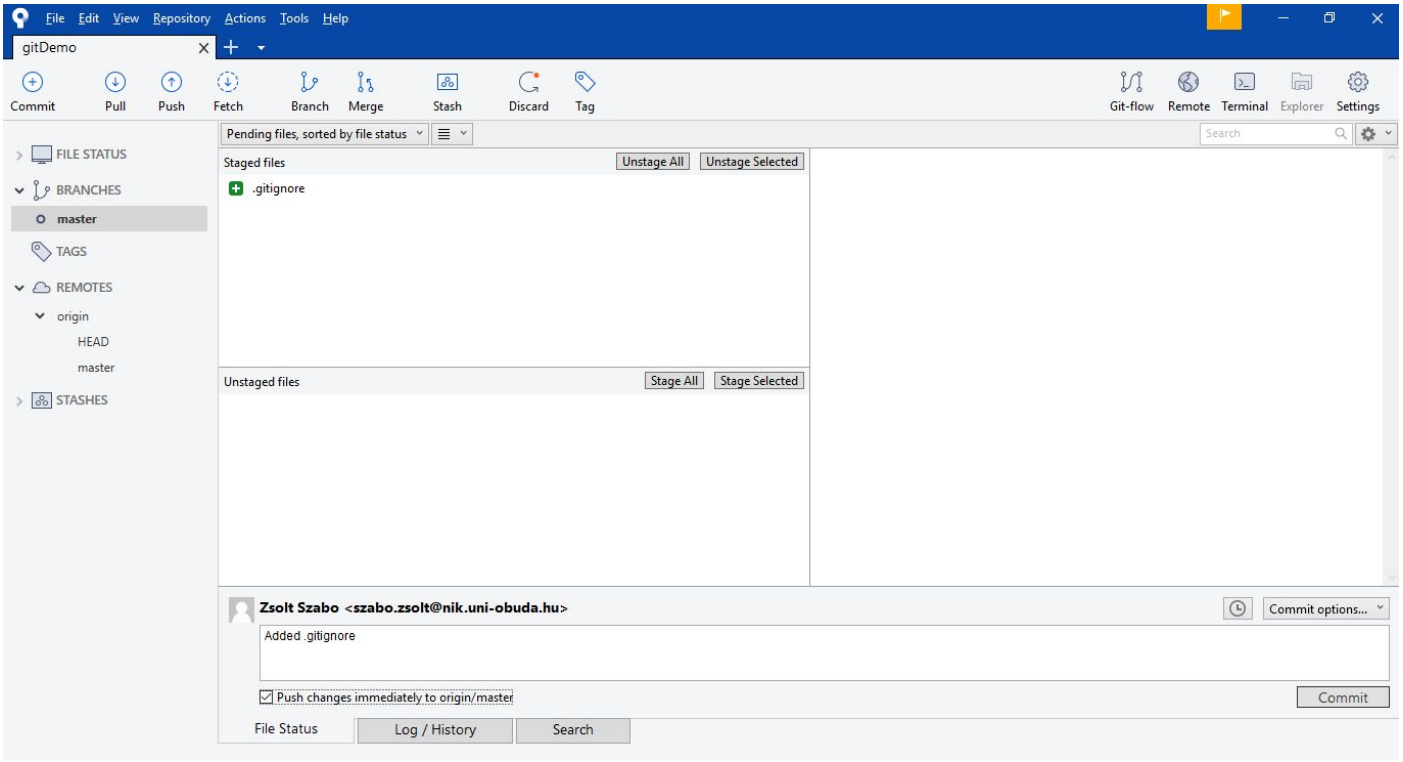
6. The staging area could be handled on this screen, but after clicking the COMMIT button it is more clear that we are actually selecting the modifications that will go into the next commit.

   Stage All/Unstage All = all files;
   Stage selected/Unstage selected = selected files;
   Stage hunk/Unstage hunk = modified section inside a file.

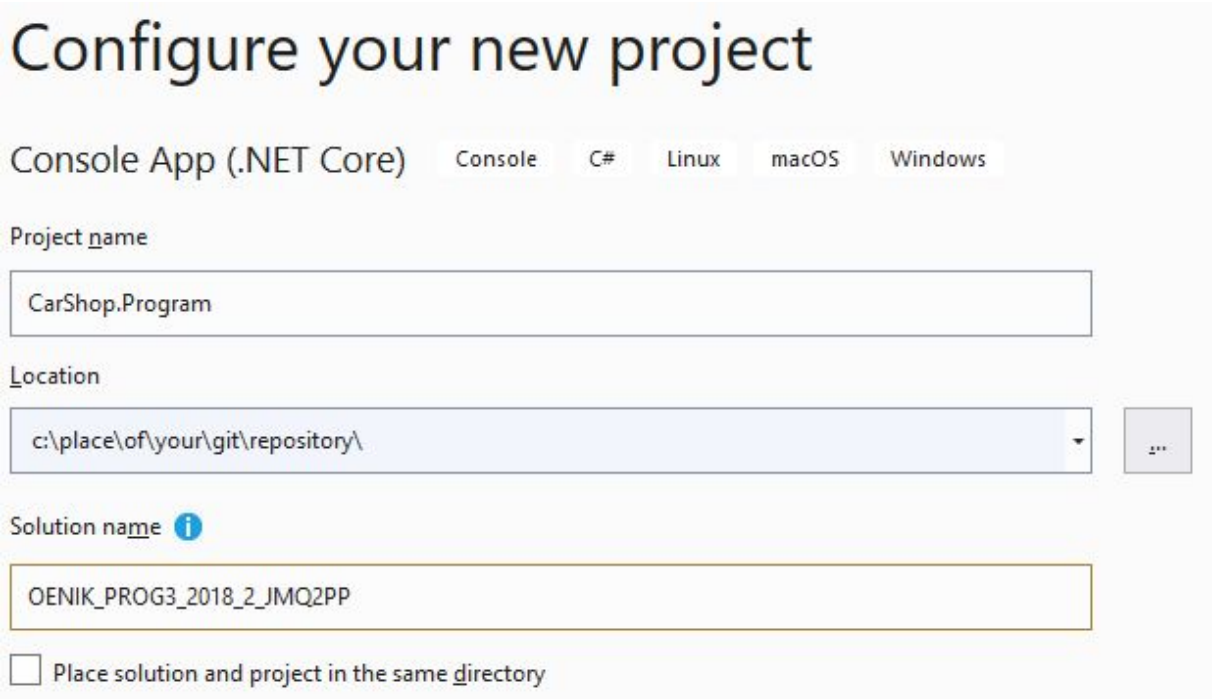
   Down there we can set the commit message, which should always be English (just like **all parts** of the source code)! If we have to use the AND/ALSO words, then split the modifications into multiple commits!

   Checking the "Push changes immediately…" is advised, if we don't check the checkbox, then after the commit we have to do a manual PUSH.

   After this, we can click on the COMMIT button in the lower-right corner (which will, in this case, also execute a PUSH).

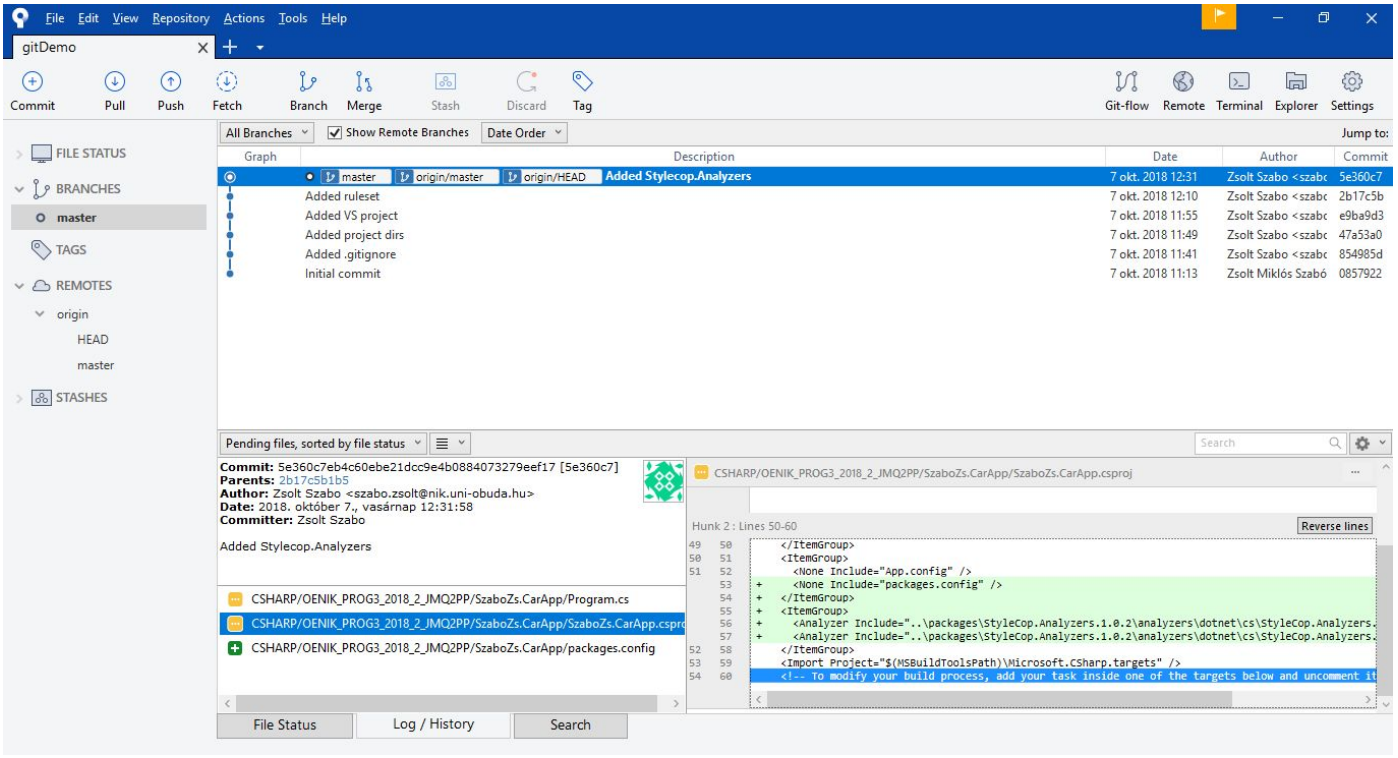


7. From the VS, we save a new solution with the first new project inside. The name of the solution is **obligatory: OENIK_SUBJECT_YEAR_SEMESTER_NEPTUN**, where SUBJECT must be PROG3 or PROG4; YEAR must be the year when the semester starts; SEMESTER means **1 = spring, 2 = autumn**. After this, you must put the neptun code / neptun codes, multiple neptun codes must be separated with underscores.



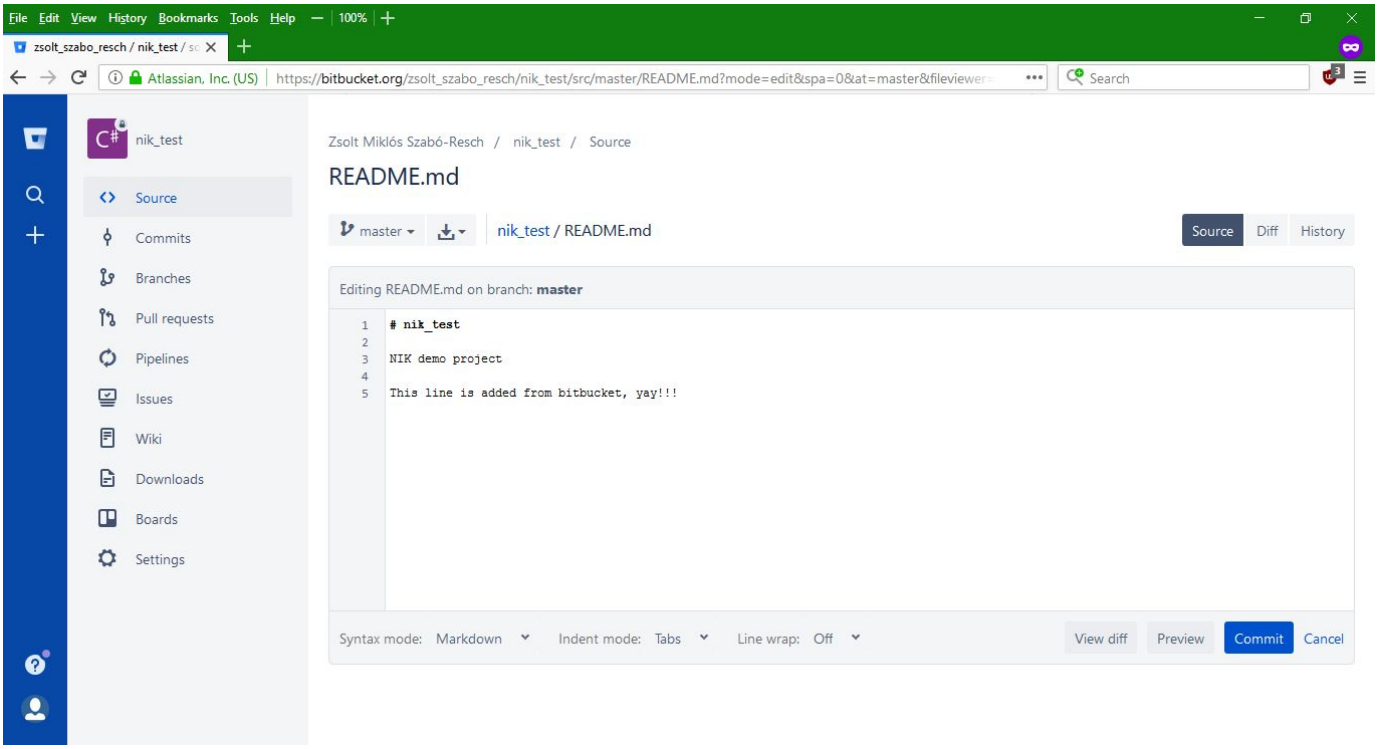
8. Then in SourceTree: COMMIT, STAGE ALL, COMMIT (+PUSH).

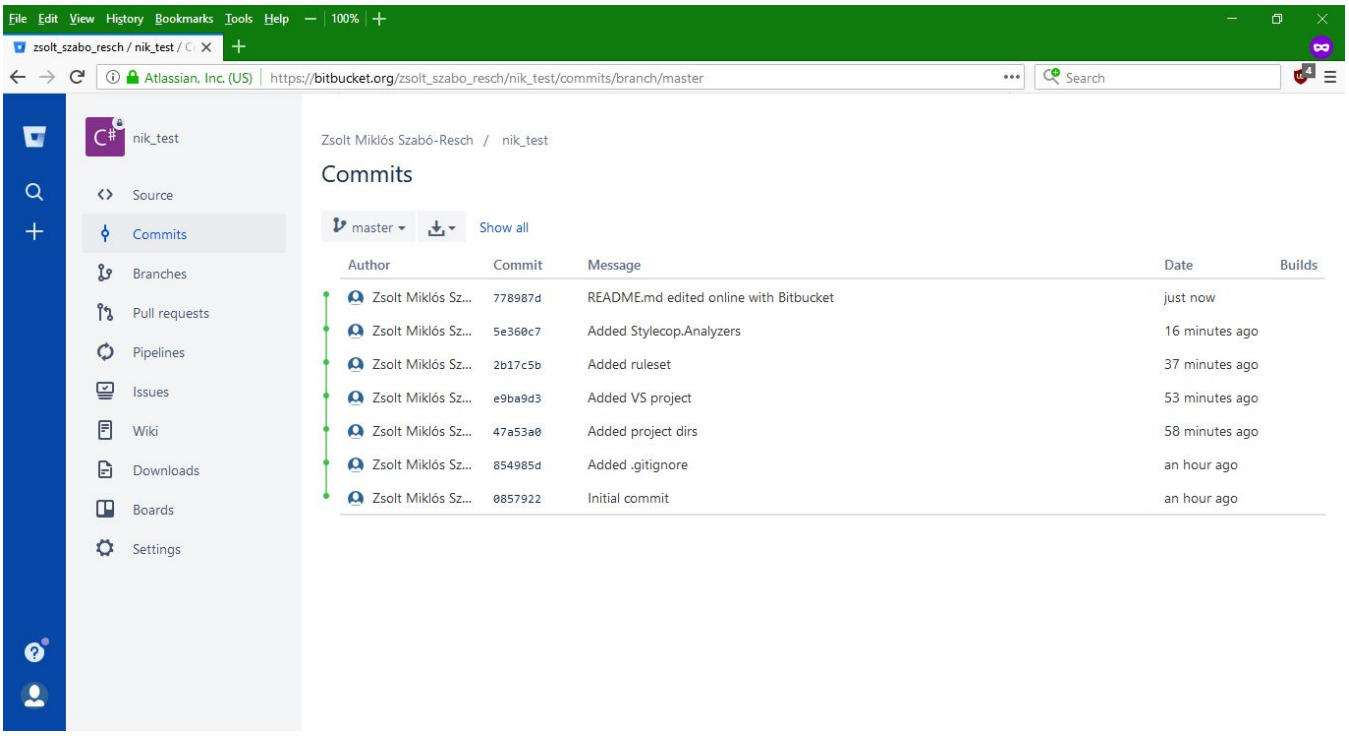
9. Install the Stylecop analyzers (for tutorial, see later)
10. Then again COMMIT, Stage All, COMMIT.



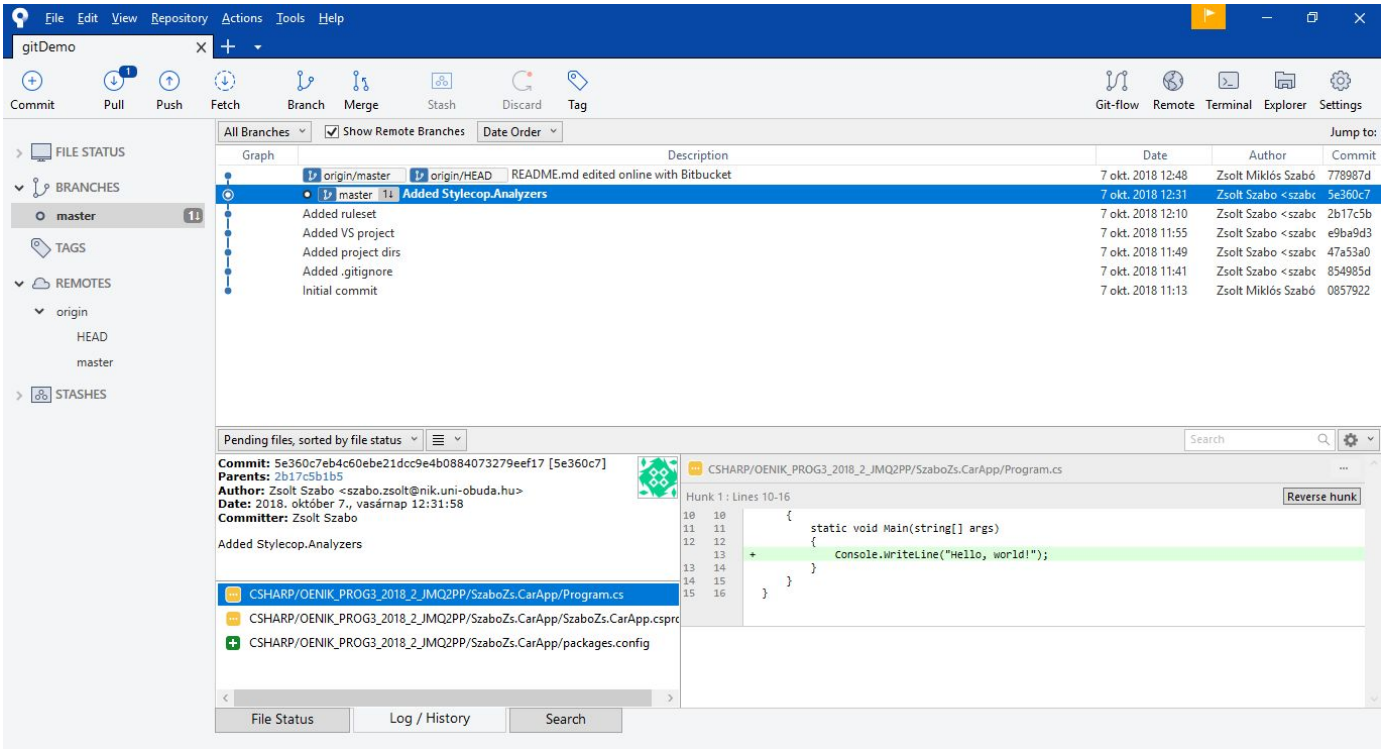
11. Changes can come from multiple sources: from our own VS version, or from external actors (other developers / even from the Bitbucket website). In case of external modification, the local repository has to be refreshed (PULL) before uploading the commits (COMMIT/PUSH); and if the external modification and our own modification changes the same file (CONFLICT), then this has to be taken care of (usually manually, with great care). For this, it is suggested to perform a FETCH/PULL before every COMMIT/PUSH.

12. We simulate an external modification: we modify the README file using the Bitbucket website!

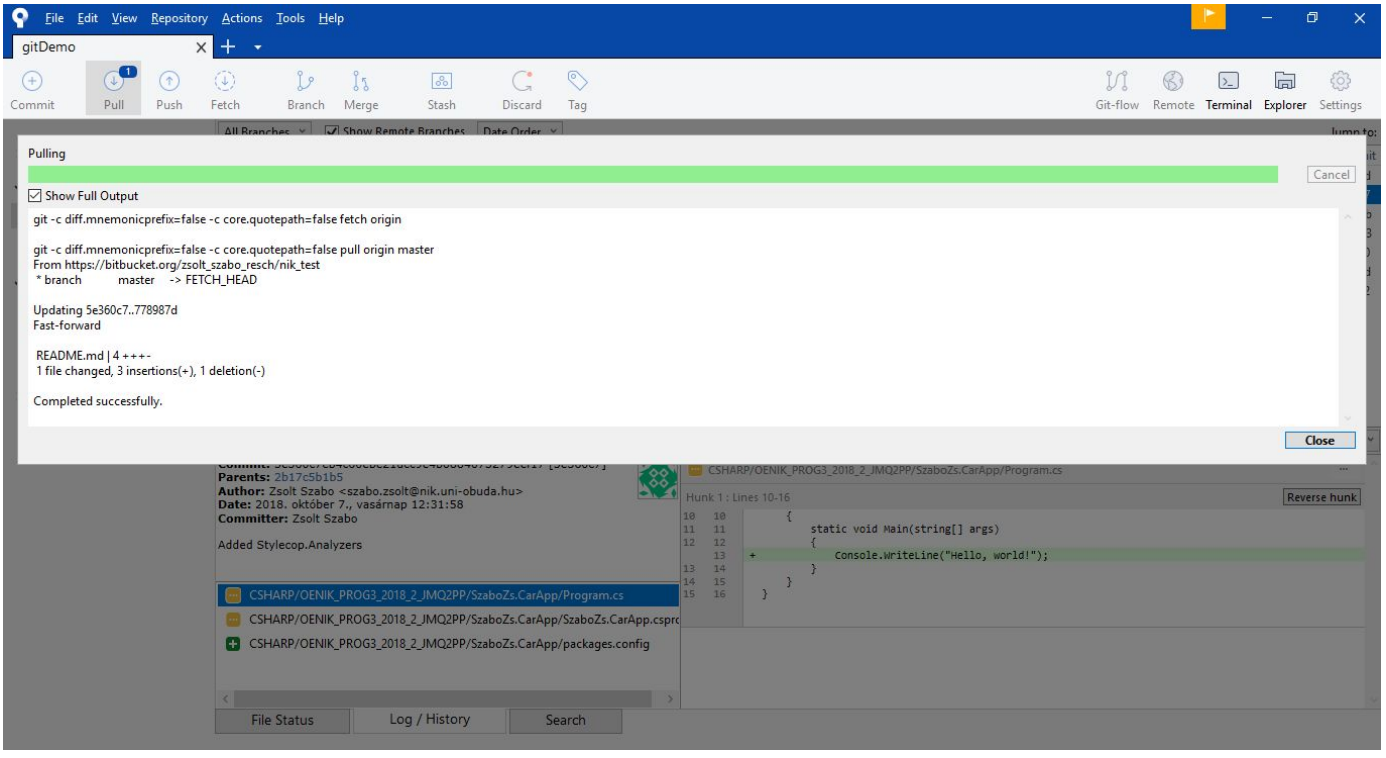


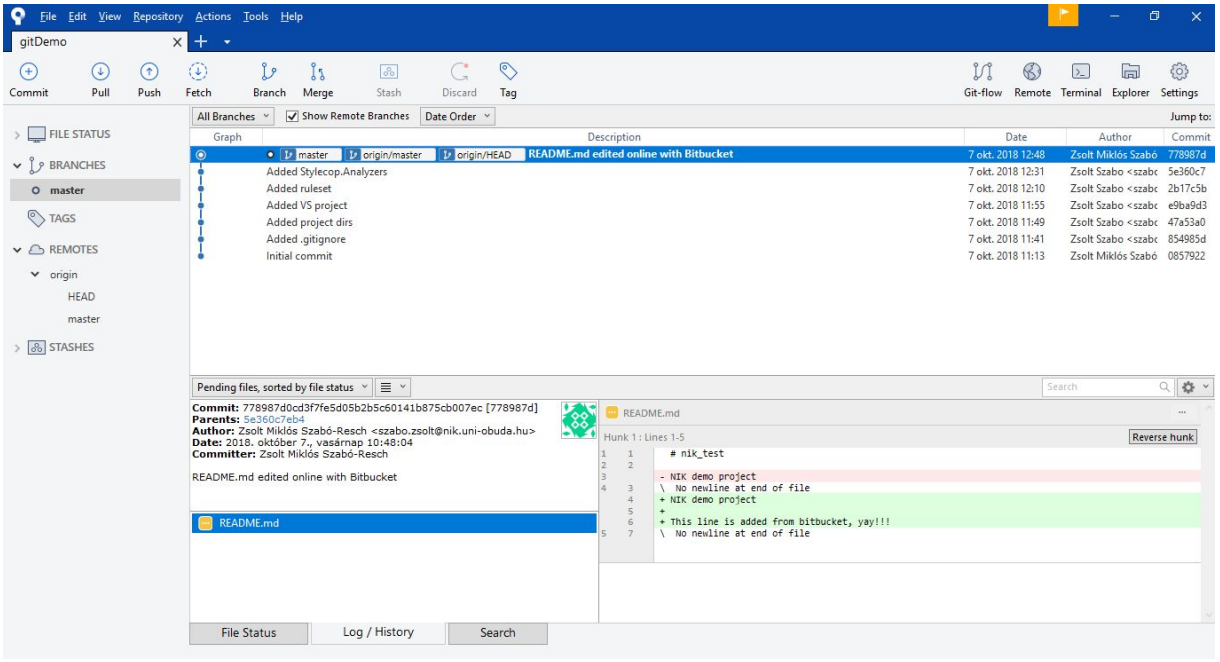


After Fetch (local tree = 1 behind):



After Pull:

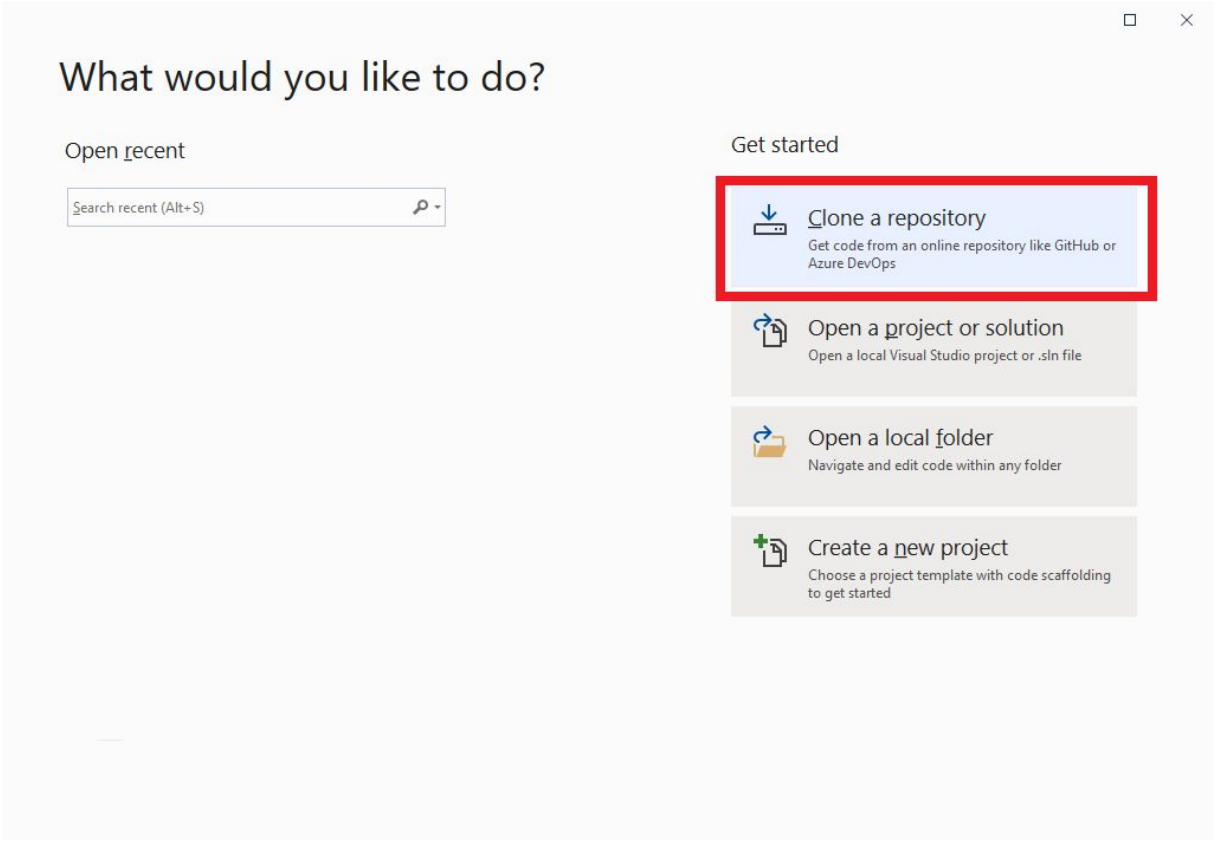


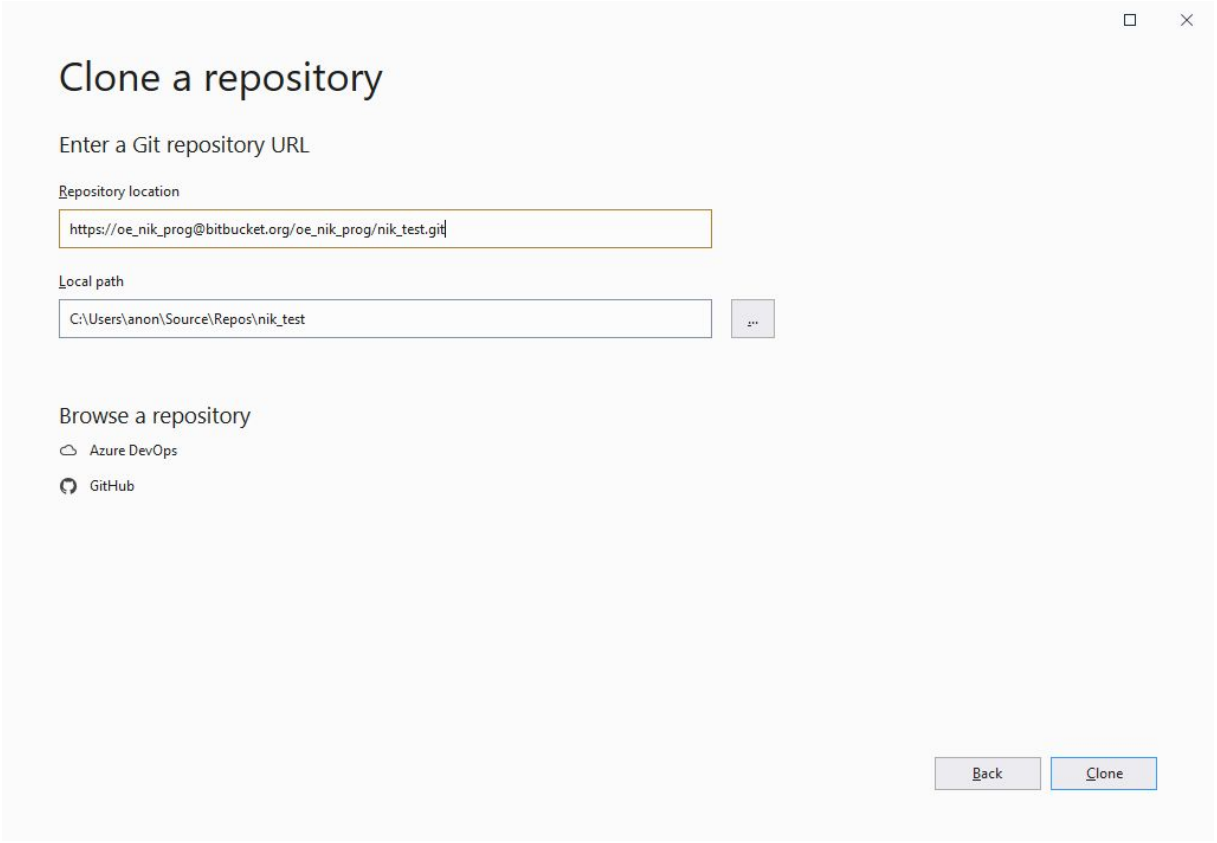


This document only shows the basic Git operations. The strength of the truly good version control systems (GIT, TFS) is that they are capable of handling multiple code sections (BRANCHes) and also they are capable of effectively handling the separation and the unification (FORK, MERGE) of these branches (including the handling of the commit conflicts that arise - this will be one of the topics in Prog4). From the students, the handling of branches is not expected in prog3, but in a typical company environment, only the experienced senior programmers have MERGE permission to the main (master) branch, every other developer works on his/her own developer branch.

Using Git from VS

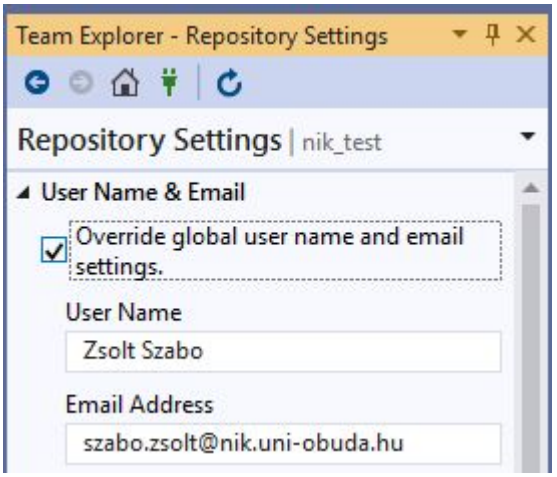
1. Cloning is possible directly from VS, but adding the Solution to the blank repository is not done the usual way, and there are two important problems: the normal Stage-management (and some other git features) are missing, and handling of files that are outside the solution can be problematic. A big plus on the VS side is the integrated conflict management interface.

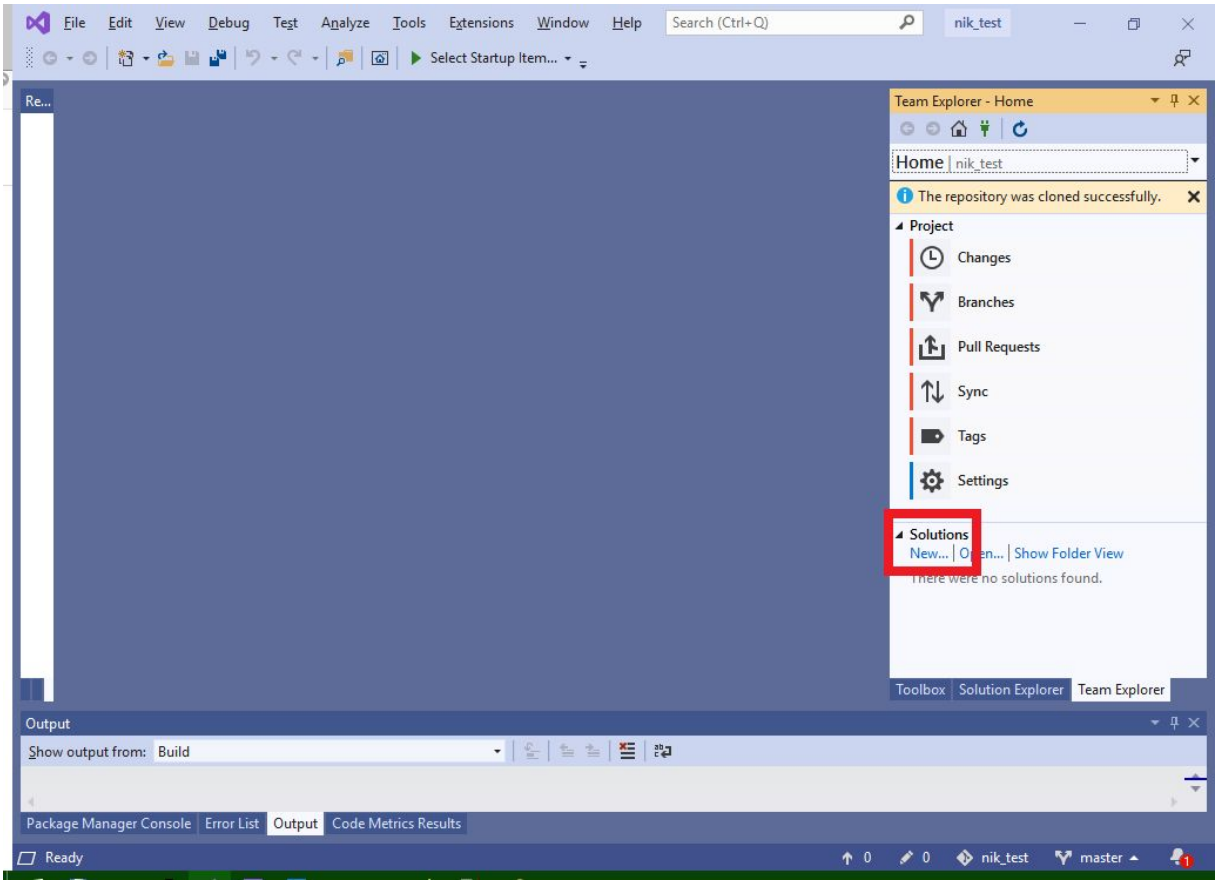


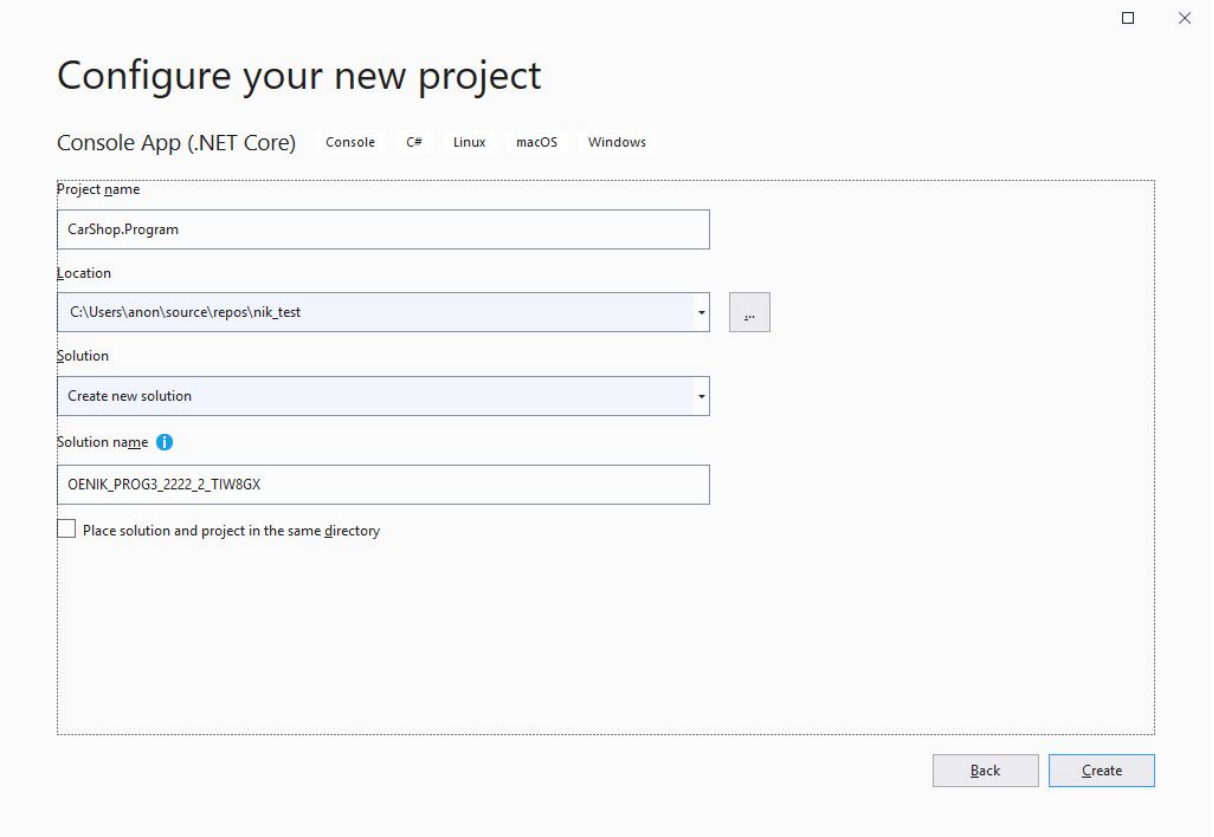


2. To use Git, we need the "Team Explorer" window. Here we can execute git operations, and we can add a new solution to the repository In my computer, when I added a new solution this way, the VS Git didn't notice the changes, I had to restart Visual Studio :)

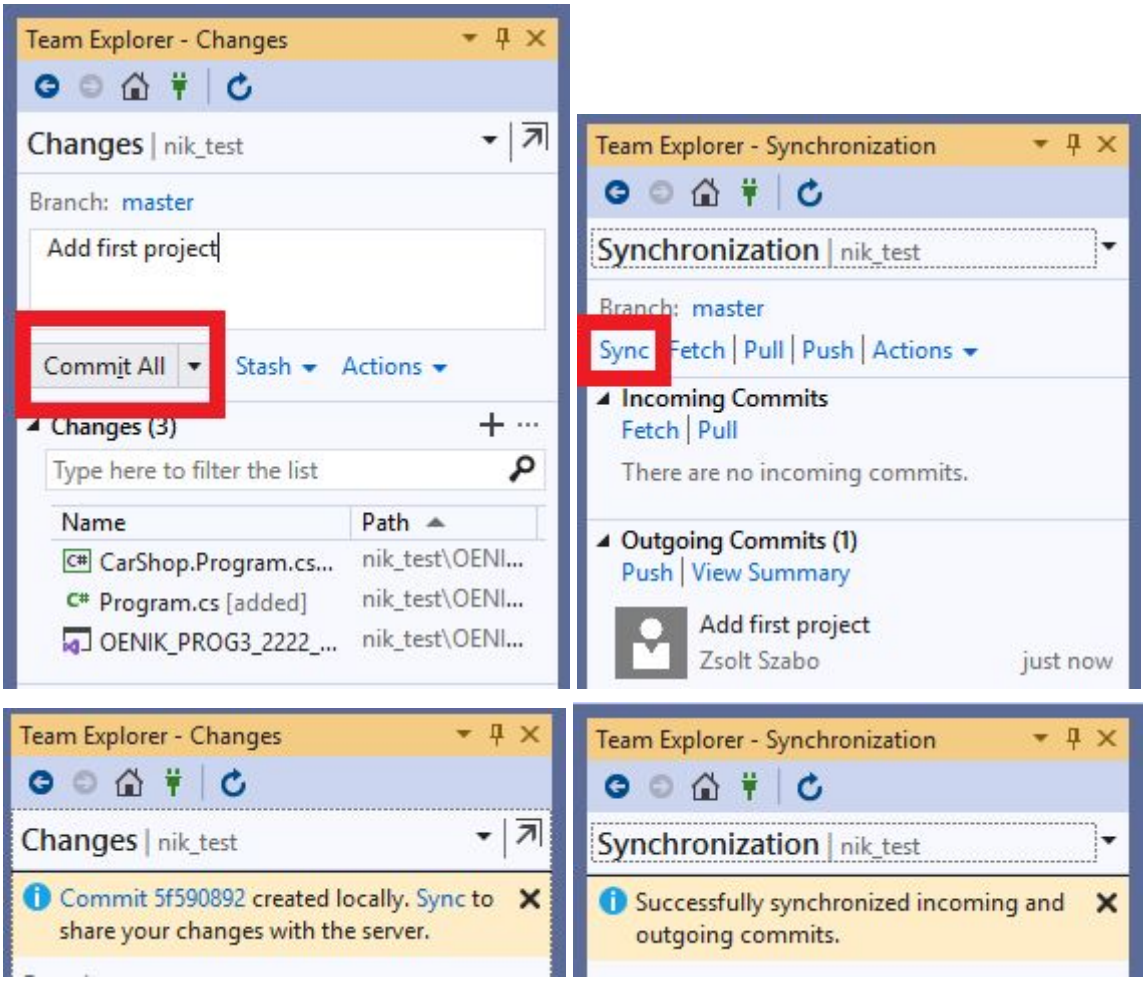
3. In the Settings window of the Team Explorer you can change Global and Repository-level settings. here you must set the author information that will be set in the Commit messages.







4. In the Team Explorer window first we will use the Changes and the Sync windows. In the Changes window we create the new Commit (I think the staging management is weird, I wouldn't recommend it; I think in VS it's difficult to split the modifications into multiple commits). Then you can execute a Push using the Sync window, but it is recommended to use the Sync operation, that is equivalent with first a PULL, and then a PUSH operation.



5. Then, you must create the new projects with enabled stylecop just like as it was detailed in the previous section. After changes, periodically create a new commit (using the Changes tab) and Sync (using the Sync tab).

Other resources:

- https://en.wikipedia.org/wiki/Version_control
- https://en.wikipedia.org/wiki/Git
- http://blog.osteele.com/posts/2008/05/my-git-workflow/
- http://nvie.com/posts/a-successful-git-branching-model/
- https://tanarurkerem.hu/git-suli/
- SourceTree (GIT GUI), Tortoise GIT (GIT GUI)
- https://confluence.atlassian.com/bitbucketserver/basic-git-commands-776639767.html (GIT parancssor: Actions / Open In Terminal)
- https://bitbucket.org/zsolt_szabo_resch/nik_test/src

Teamwork

The creation of the project is now divided into two parts:

1. One team member creates the common GIT repository with the common VS solution with one project, basically following the description above. Also, this one team member gives permission to the other repo users (plus the usual oe_nik_prog user).
2. The other team members should simply clone the repository.
3. When modifying things, we must take care of branches and conflicts.

When using the repo from VS, the following links can be useful:

- https://docs.microsoft.com/en-us/vsts/git/tutorial/creatingrepo?tabs=visual-studio
- https://docs.microsoft.com/en-us/vsts/git/tutorial/clone?tabs=visual-studio
- https://docs.microsoft.com/en-us/vsts/git/tutorial/commits?tabs=visual-studio
- https://docs.microsoft.com/en-us/vsts/git/tutorial/branches?tabs=visual-studio
- https://docs.microsoft.com/en-us/vsts/git/tutorial/pushing?tabs=visual-studio
- https://docs.microsoft.com/en-us/vsts/git/tutorial/pulling?tabs=visual-studio
- https://docs.microsoft.com/en-us/vsts/git/tutorial/merging?tabs=visual-studio
- https://docs.microsoft.com/en-us/vsts/git/tutorial/undo?tabs=visual-studio

This example shows a good branch structure (this can be simplified in our case, the usage of the dev+master branches are obligatory, the usage of feature branches are recommended):

- http://nvie.com/posts/a-successful-git-branching-model/

CODE ANALYSIS

1. You have to do two steps for ALL created c# projects: first, right click on the newly created project, then Properties, then in the Build section you must enable the XML documentation file, this will be needed because of the Doxygen documentation as well. Here, you **MUST USE** a relative path (simply CarShop.Data.xml), an absolute path (c:\xxxx, d:\xxxx, \xxxx) will cause build errors when compiling on a different computer (e.g. by NikGitStats)!!!
2. Secondly, Tools/NuGet package Manager/Manage NuGet Packages for Solution, here search for the **Stylecop.Analyzers** package, and add it to the project. These two steps (nuget stylecop.analyzers + enable XML documentation file) must be repeated with all subsequent projects!!! (note: the old FxCop analyzers and Stylecop classic packages are NOT used! Microsoft.CodeAnalysis.NetAnalyzers is enabled via the dotnet SDK/project file!)
3. After a build, you will have lots of Warnings, listing the errors in your coding style. You should first INTERPRET the warning message, and then you have to DECIDE if you want to follow that rule or not. Important philosophy: ALL RULES ARE GOOD, they warn about one specific code smell, but some of them are more important and more widespread than others. IF you think the rule is not good for you, AND the rule is not in the GitStats list of rules that cannot be suppressed (check the Gitstats website, [?] link of the StylecopSuppressions checker), then (and only then!) you can suppress the rule. Otherwise, you have to fix your code to get rid of the warning.
4. For the warnings, you **MUST NOT** use "right click / Suppress / In Source", and also using the <NoWarn> inside the CSPROJ is forbidden (exception: the default NoWarn value of 1701;1702)!
5. You can only use the project-level GlobalSuppressions.cs and the solution-level .editorconfig file to suppress warnings (the latter is the recommended, but since some assembly-level warnings are not suppressable with that, sometimes you need the GlobalSuppressions.cs file).

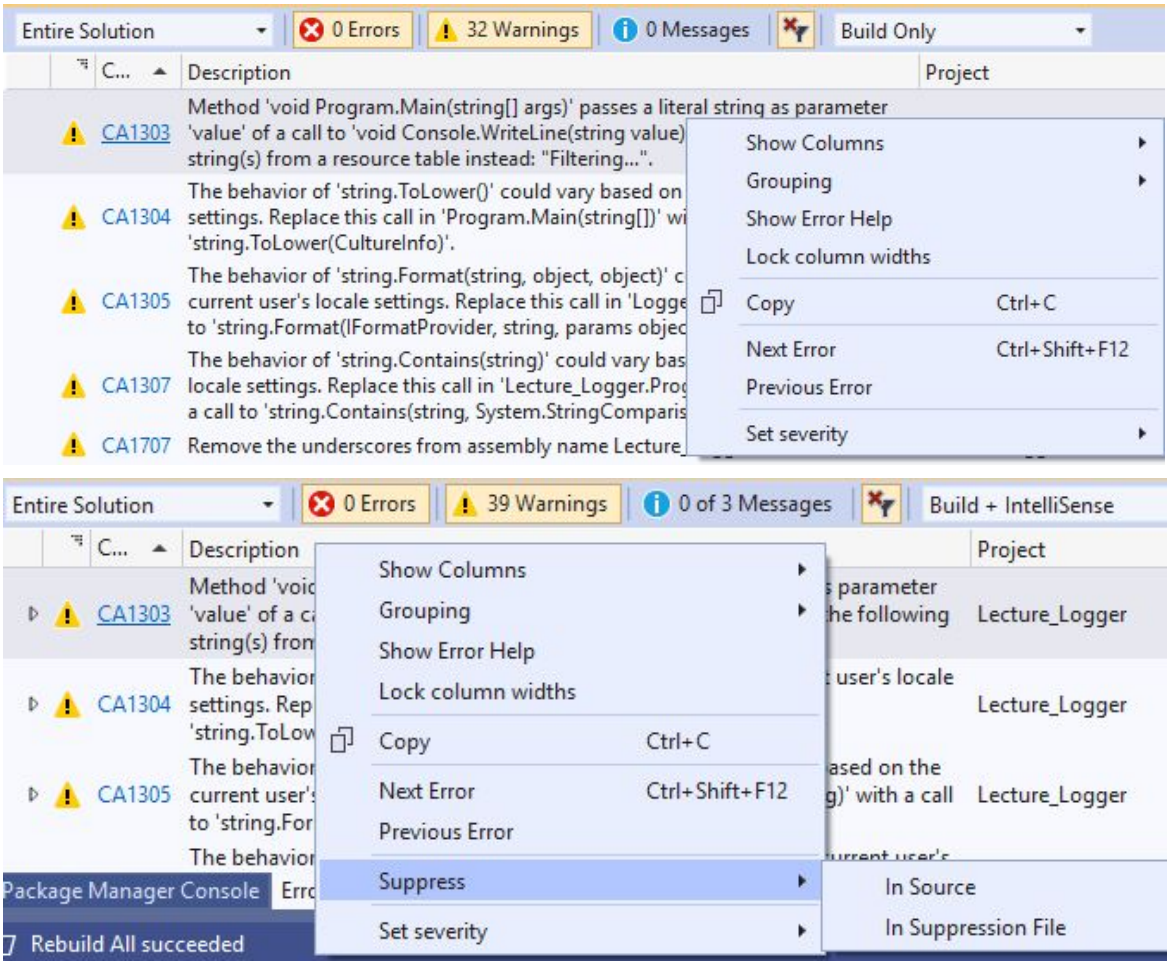
   .editorconfig

   ```
   [*]
   dotnet_diagnostic.CA1012.severity = none
   ```

   GlobalSuppressions.cs (assembly-level, specific overrides)

   ```
   using System.Diagnostics.CodeAnalysis;
   [assembly: SuppressMessage("", "CA1014", Justification = "<NikGitStats>", Scope = "module")]
   ```

6. Only use the meaningful Suppressions, and do not override warnings resulting from actual documentation / readability mistakes!
7. Visual Studio should help you in handling the GlobalSuppressions.cs file, but the "Suppress" menuitem in the context menu is not always shown as expected. Look out that the "Suppress" menu will not be shown with "Build" warnings, only with "Build + Intellisense" or "Intellisense" filters (to my experience, sometimes it is not displayed even if you use that setting). You must NOT use "In Source", only the "In Suppression File" is allowed.
8. The following entries must be added MANUALLY to all project files:
   <EnableNETAnalyzers>true</EnableNETAnalyzers>
   <AnalysisMode>AllEnabledByDefault</AnalysisMode>
   <EnforceCodeStyleInBuild>true</EnforceCodeStyleInBuild>




```
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <TargetFramework>net5.0</TargetFramework>
    <EnableNETAnalyzers>true</EnableNETAnalyzers>
    <AnalysisMode>AllEnabledByDefault</AnalysisMode>
    <EnforceCodeStyleInBuild>true</EnforceCodeStyleInBuild>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="StyleCop.Analyzers" Version="1.1.118">
      <PrivateAssets>all</PrivateAssets>
      <IncludeAssets>runtime; build; native; contentfiles; analyzers; buildtransitive</IncludeAssets>
    </PackageReference>
  </ItemGroup>

  <ItemGroup>
    <ProjectReference Include="..\CarShop.Data\CarShop.Data.csproj" />
    <ProjectReference Include="..\CarShop.Repository\CarShop.Repository.csproj" />
  </ItemGroup>

</Project>
```

You must not use too many suppressions and you must not use suppression to hide the very basic rules. You MUST HAVE zero warnings and zero build errors!

DoxyGen Quickstart

In addition to the manually written documentation and diagrams, an important part of the developer documentation is the automatically generated documents (using the source code). One of the tools for this task is the DoxyGen (https://www.doxygen.nl/download.html; " binary distribution for Windows" - this includes Doxygen and DoxyWizard too - if the Windows OS warns us because of this dangerous app, then its a false alarm :))

XML documentations

All the public parts of the source code (including classes, methods, variables, properties, events and everything else) must be documented with XML documentation. Also all documentation must be added that is requested by the StyleCop ruleset.

The XML documentation parts of the C# code can be generated into a real XML file: **Enable Project Properties / Build / XML Documentation file**.

For the texts, English language must be used, however we will not accept automatically generated „english-like" texts created by Resharper and other similar programs – projects that are handed in containing documentation like that will count as non-documented (e.g. not handed in).

The format of XML documentation is as follows:

```
/// <summary>
/// Moves the current shape, if possible.
/// </summary>
/// <param name="x">x component of movement vector</param>
/// <param name="y">y component of movement vector</param>
public void Move(int x, int y)
{
    // …
}
```

If „///" is typed over a finished method or class, Visual Studio will add the necessary template for this, which then just needs to be filled in. The completed XML documentation will appear in Intellisense which could help while writing code.

The main advantages of the DoxyGen are that it is open source, and it supports nearly all programming languages. It is much more customizable, however, the zillion of configuration parameters are hard to follow, and it is hard to wire all components together. In addition to this, it cannot be integrated into the VS as it is (I wouldn't trust the Nuget-distributed version).
A good alternative might be SandCastle, but in a real development project the software used for generating documentation is very likely already given.

DOXYGEN

Command line:

- https://sourceforge.net/projects/doxygen/files/ , download appropriate version
- doxygen -g doxygen.cfg (this command generates the config file)
- Edit the config file
- doxygen doxygen.cfg

GUI (doxywizard):

- https://www.doxygen.nl/download.html#srcbin , doxygen-1.x.y-setup.exe
- Working directory: the WRITEABLE directory where the doxygen.exe file is
- Project name, Synopsis, Version: fill them in.
- Source code directory: the directory that contains the .sln file for the project.
- Scan recursively: yes!
- Destination directory: specify a custom WRITEABLE directory.
- WRITEABLE directories = Important that those folders should be writeable by non-elevated users, without Windows UAC prompts

<Next, Mode tab>

- Documented Entities Only
- Include cross-referenced source code: yes!
- Optimize for Java or C# output

<Next, Output tab>

- HTML: With navigation panel
- LaTex: no (html tickbox stays ticked).

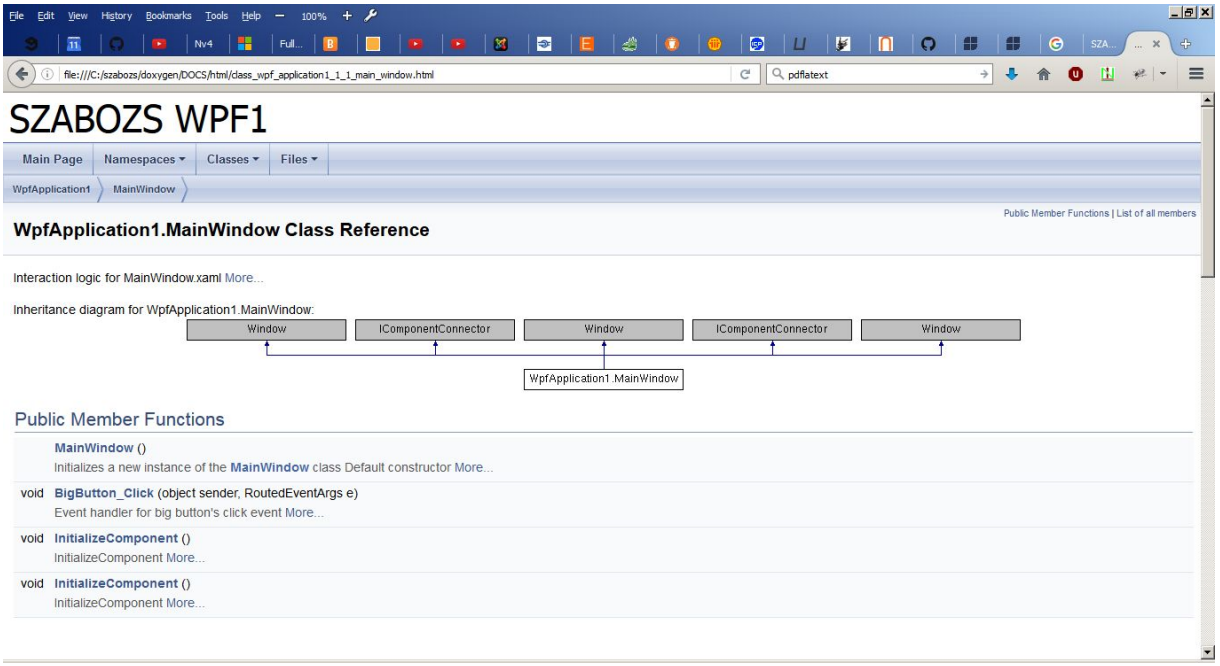
<Expert tab / Build section>

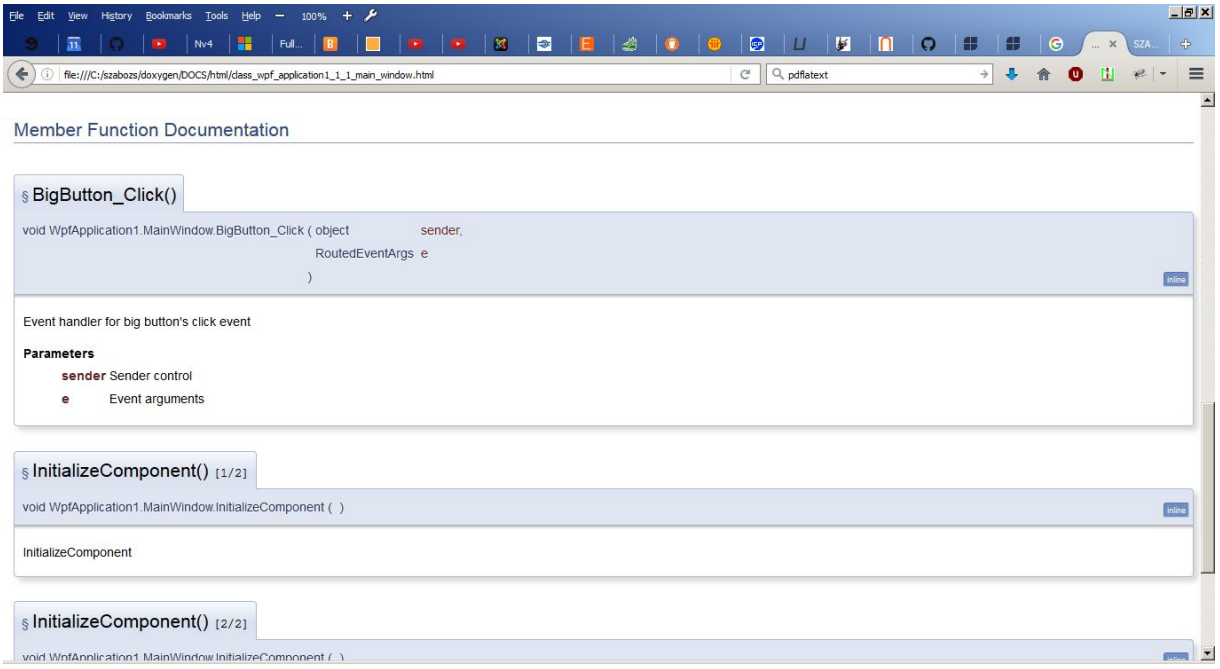
- EXTRACT_PACKAGE: yes!
- EXTRACT_STATIC: yes!

After some additional „Next”s (or after selecting the “Run” tab) press „Run doxygen”.

Check if the documentation has been generated correctly starting at index.html (or by clicking on the “Show HTML output” button): the documentation should contain the nice documentation for your classes and methods.

Put the results into the “Doxygen” folder that should be in the new directory called “Documentation” of the repository root (without an extra “html” folder, the directory Documentation\Doxygen should contain all the generated files).


SZABOZS WPF1
Main Page
Namespaces
Classes
Files
WpfApplication1
MainWindow
Public Member Functions | List of all members
WpfApplication1.MainWindow Class Reference
Interaction logic for MainWindow.xaml More...
Inheritance diagram for WpfApplication1.MainWindow:
Window
IComponentConnector
Window
IComponentConnector
Window
WpfApplication1.MainWindow
Public Member Functions
MainWindow ()
Initializes a new instance of the MainWindow class Default constructor More...
void BigButton_Click (object sender, RoutedEventArgs e)
Event handler for big button's click event More...
void InitializeComponent ()
InitializeComponent More...
void InitializeComponent ()
InitializeComponent More...


Member Function Documentation
§ BigButton_Click()
void WpfApplication1.MainWindow.BigButton_Click (object sender,
RoutedEventArgs e
)
inline
Event handler for big button's click event
Parameters
sender Sender control
e Event arguments
§ InitializeComponent() [1/2]
void WpfApplication1.MainWindow.InitializeComponent ( )
inline
InitializeComponent
§ InitializeComponent() [2/2]

FAQ

- The program must not die with an exception, under any circumstances! We should aim to handle all possible cases and user inputs; and no errors should be in the final code (including visible operational and non-visible code errors as well!)
- The required current dotnet version is .NET5.0.x, the VS 2019 version is 16.8.x (use start menu/Visual Studio Installer). No other earlier dotnet framework/dotnet core versions are expected to work, especially because the current test runner combination (nUnit runner + dotnet test + dotcover code coverage tool) is VERY sensitive to framework versions.
- Iinside VS, change all projects in solution to use net5 (right click on project/properties/Application/Target Framework)
- You must only have the Doxygen documentation ready at the end of the semester. Do not forget to RE-generate the doxygen HTML documentation whenever you have major changes in the code. Documentation\Doxygen\index.html must exist in the git root, and all the other documentation html/image/css files must be in the same folder!
- In the video you must show the application in use, as the menuitems (in case of prog4, the game) are running. Use any application (OBS Studio / Powerpoint / FastStone Capture / anything else) to create the recording. The maximum file size is 99MB, you have to upload the video to https://users.nik.uni-obuda.hu/ff/
- Be careful that Mocks MUST NOT use any implementation details, you always mock INTERFACES and you must NEVER call someMock.Object.XXXX()
- If you remove a project/file from your solution in Visual Studio, look out that files will not be deleted from the disk, only from the Solution Explorer, thus they might still remain in the git repository. You have to physically remove the project files using a file manager in order to remove the project from git as well (check the actual contents of the git repo in the bitbucket website OR by cloning the git repo into a blank folder).

Our primary aim is always to keep the industry standards and philosophy, and to solve the problem using the best possible code. We receive lots of "is it a problem if" questions, and always the same reply goes for all: if you do not contradict this document or the subject-specific requirements, then it is not an instant "Fail" grade. BUT if you already tought that something might cause problems in the future, and the code might not be the best, then probably the answer is: yes, this is a problem :) It might not mean an instant fail grade, but even the student knows that it is wrong; otherwise you would not ask it…
If the solution has no other problems, everything else is perfect, then of course you can finish the code with "small error" inside, awaiting the "yellow card" warning from the practice teacher; and hoping that there is no other errors in the code. However, two yellow card warnings will indeed result in a fail grade, so please try to write an error-less code.
On the other hand, please DO have some professional pride and self esteem: one should NEVER hand in any code where you know that it is not good!!!