1 Lab Exercise: Git basics with the Calculator

In this exercise you will use and extend your previous Calculator-exercise to practice using Git. Thus, the focus here is *not* on the software, but on using Git properly!

Remember! Whenever you have done a change (implemented a new test or a working part of a method), commit your work with a decent commit comment. When you have finished a suitable amount of work, do a Pull – Test – Push cycle, as explained in the lecture slides.

1.1 Exercise 1:

All students: Install Git Tools – as described on Brightspace

1.2 Exercise 2:

Team up in teams of 2-4 persons via Brightspace – use "Gruppetilmelding". This should be your groups for handing in the mandatory exercise.

1.3 Exercise 3:

1. For this class, each team should have an account on GitHub, which you will use to store remote repositories. Agree on a name and a password. Create it on Github.com. This should just be a normal, public, free account. Github will send an e-mail to the entered e-mail address with request for a confirmation. This confirmation must be done, before the next steps. Optionally, each team member can make a personal account, and they are invited as Collaborators to the common account.

2. One (1) team member shall take his/her solution for the Calculator from last lecture, including unit tests made with NUnit (or use the provided solution from Brightspace).

3. Then, this team member shall verify that he can build the solution and run the tests from within Visual Studio.

4. Then, this team member shall create a repository. This can be done from several places – see the video on Brightspace
    a. Use the "Add to Source Control" at the bottom right of your VS window
    b. Open the Git Change window – click Create Git Repository
    c. Right click on the Solution in the Solution Explorer and select the Create Git Repository command

5. You can choose to create both a local repository and creating a GitHub at the same time. Don't worry if you only make a local repository, it can be "Published" to GitHub later on.

6. To make sure a .gitignore file has been created, Select the Git menu, and selected the Settings command. Go to the Source Control/Git Repository Settings/General page, and check that there is an Ignore File.

7. Open the Git Changes view. From here Commit/Push/Pull/and Sync can be executed. For the first user, the solution's repository was already Pushed, or a Publish or a Push can executed here.

8. Then, all other team members shall clone the repository. There are several ways to do this, see also the video on Brightspace. If you have several Github accounts involved it is most easily done from VS.
    a. When starting VS, there is an option to clone a project. It has several ways to do this
    b. There are other ways to clone from VS, the Git menu Clone Repository command is one of them
    c. Use TortoiseGit to make a clone
    d. Use the command line to call the `git clone <url>` command

9. After cloning, all other team members shall try to build the solution on their own PC.

1.4 Exercise 4:

Add the following functionality to the Calculator or other relevant features you think could be interesting – let your imagination run.

Make decisions on details of operations and exceptions as necessary.

Make the relevant unit tests for the class for operations and exceptions.

Be sure to divide the work between you, so you work in parallel!

`public double Divide(` `  double dividend, double divisor)`	Return the result of the division dividend/divisor. Consider what should happen when a division by zero is attempted. Test it.
`public double Accumulator {get; private set; }`	Add the C# property Accumulator to the Calculator class. It should always contain the result of the latest operation. Consider what it should contain, initially and when errors happen. Test it accordingly.
`public void Clear()`	This method should clear the accumulator to contain zero.

`public double Add(double addend)`
`public double Subtract(double subtractor)`
`public double Multiply(double multiplier)`
`public double Divide(double divisor)`
`public double Power(double exponent)`
	Make overloads of the calculation functions with one parameter less than the original. The missing operand is always taken from the accumulator, thus making it possible to make chained calculations like on a real calculator.
	The methods returns the result **and** changes the accumulator.
	Consider error situations.
	Test everything.

`(no new function)`	Check the result of your Power function. Are you satisfied with the results from strange combinations of operands, e.g. negative x with non-integer exponent? Do you need another exception? Test the implementation of your decisions.

Remember!

- Whenever you have done a change (implemented a new test or a working (part of a) method), commit your work with a decent commit comment.
- Push your changes to the distributed Git repository when you have something to share. Remember, before you push, pull the repository and solve any merge conflicts (due to previous commits from other team members), and test, if any changes were pulled or had to be merged. Merge is most easily done using Visual Studio's built-in three-way merge.