Lab 01

Source Control Management using git

Hands on Lab



**Conditions and Terms of Use**

**Microsoft Confidential - For Internal Use Only**

This training package is proprietary and confidential, and is intended only for uses described in the training materials. Content and software is provided to you under a Non-Disclosure Agreement and cannot be distributed. Copying or disclosing all or any portion of the content and/or software included in such packages is strictly prohibited.

The contents of this package are for informational and training purposes only and are provided "as is" without warranty of any kind, whether express or implied, including but not limited to the implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

Training package content, including URLs and other Internet Web site references, is subject to change without notice. Because Microsoft must respond to changing market conditions, the content should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication. Unless otherwise noted, the companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred.

**Copyright and Trademarks**

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

For more information, see Use of Microsoft Copyrighted Content at  
[*http*://www.microsoft.com/about/legal/permissions/](http://www.microsoft.com/about/legal/permissions/)

Microsoft®, Internet Explorer®, and Windows® are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Other Microsoft products mentioned herein may be either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. All other trademarks are property of their respective owners.

© 2016 Microsoft Corporation. All rights reserved.

Contents

[Introduction 4](#_Toc449080308)

[Exercise 1 – Installing git locally 4](#_Toc449080309)

[Exercise 2 – Getting started with git – Command Line Interface 7](#_Toc449080310)

[Exercise 3 – Getting started with git – Visual Studio 9](#_Toc449080311)

[Exercise 4 – Creating an initial master Remote branch 13](#_Toc449080312)

[Exercise 5 – Creating a feature branch 16](#_Toc449080313)

[Exercise 6 – Pushing changes to a remote branch 17](#_Toc449080314)

[Exercise 7 – Approving Pull Requests 24](#_Toc449080315)

[Exercise 8 – Resolving Conflicts 26](#_Toc449080316)

# Introduction

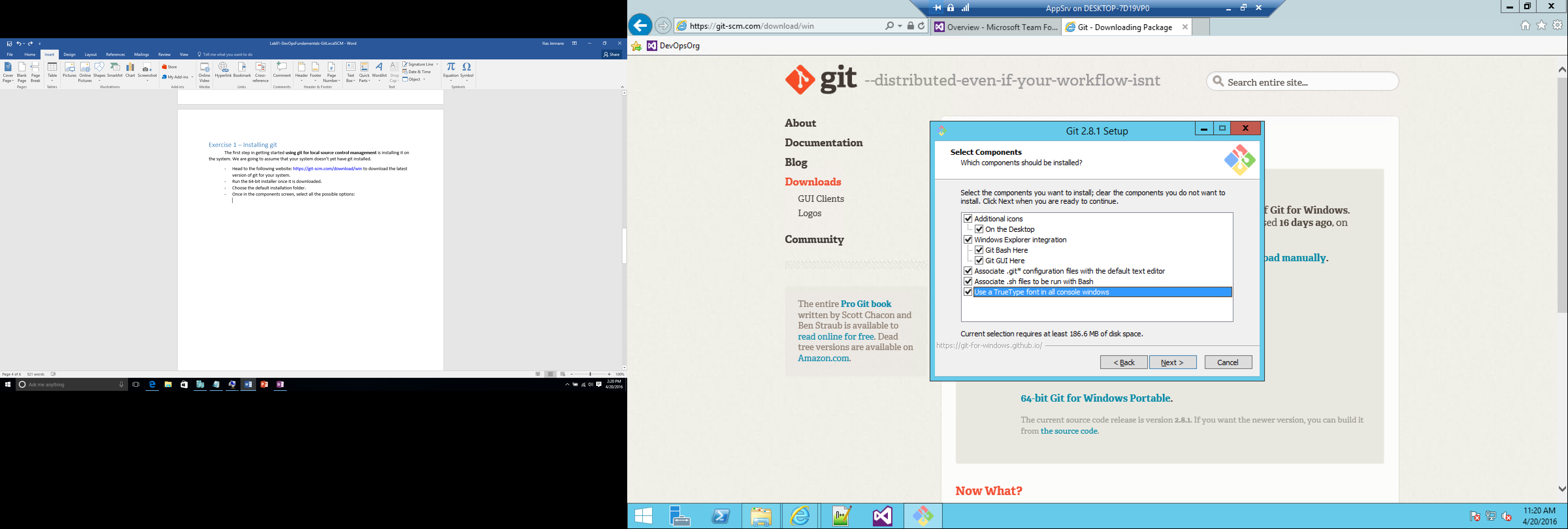
**Duration: 60 minutes**

Goal: Introduction to git for source control management

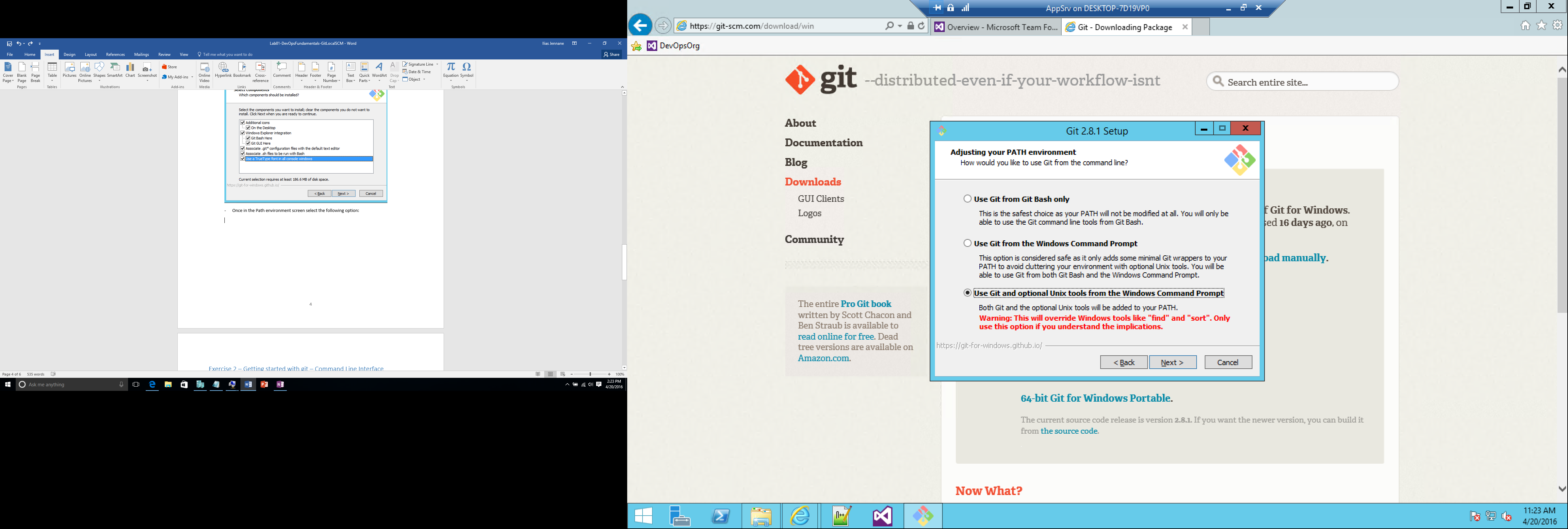
# Exercise 1 – Installing git locally

The first step in getting started **using git for local source control management** is installing it on the system. We are going to assume that your system doesn’t yet have git installed. If it already does, please scan through these instructions so you know for future reference then move on to the next section.

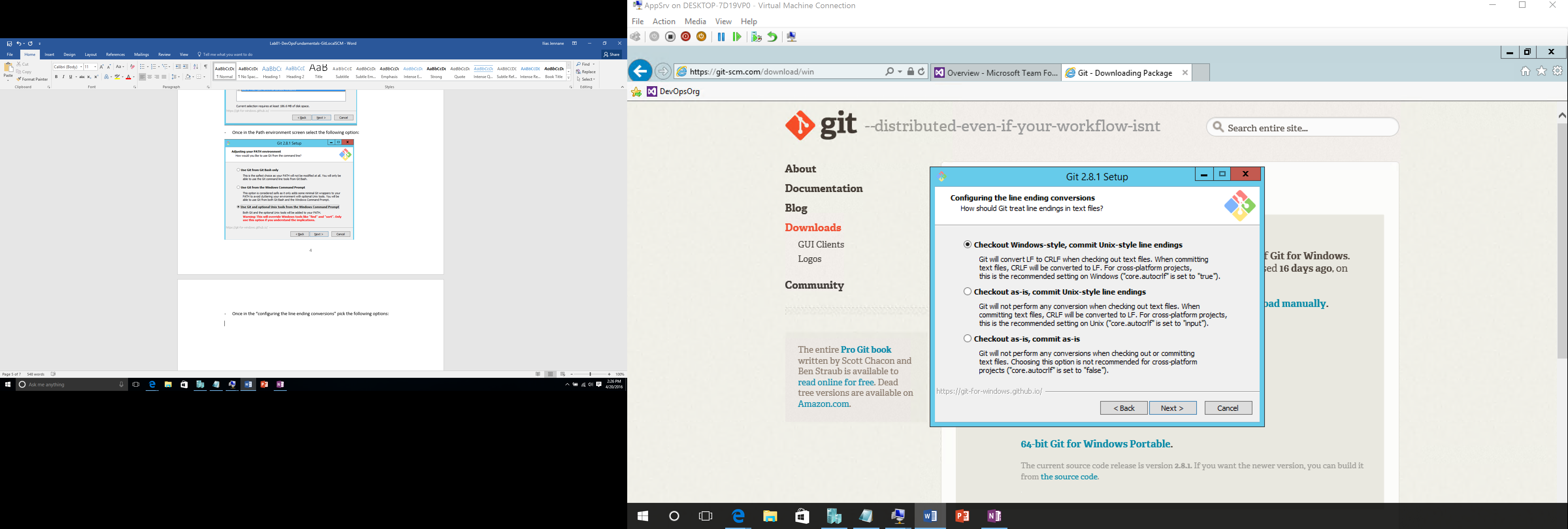
* Head to the following website: <https://git-scm.com/download/win> to download the latest version of git for your system. Run the 64-bit installer once it is downloaded. (If you’re facing connectivity issues, the installer is available locally in the C:\Tools\Git folder)
* Choose the default installation folder.
* Once in the components screen, select all the possible options, then click next:



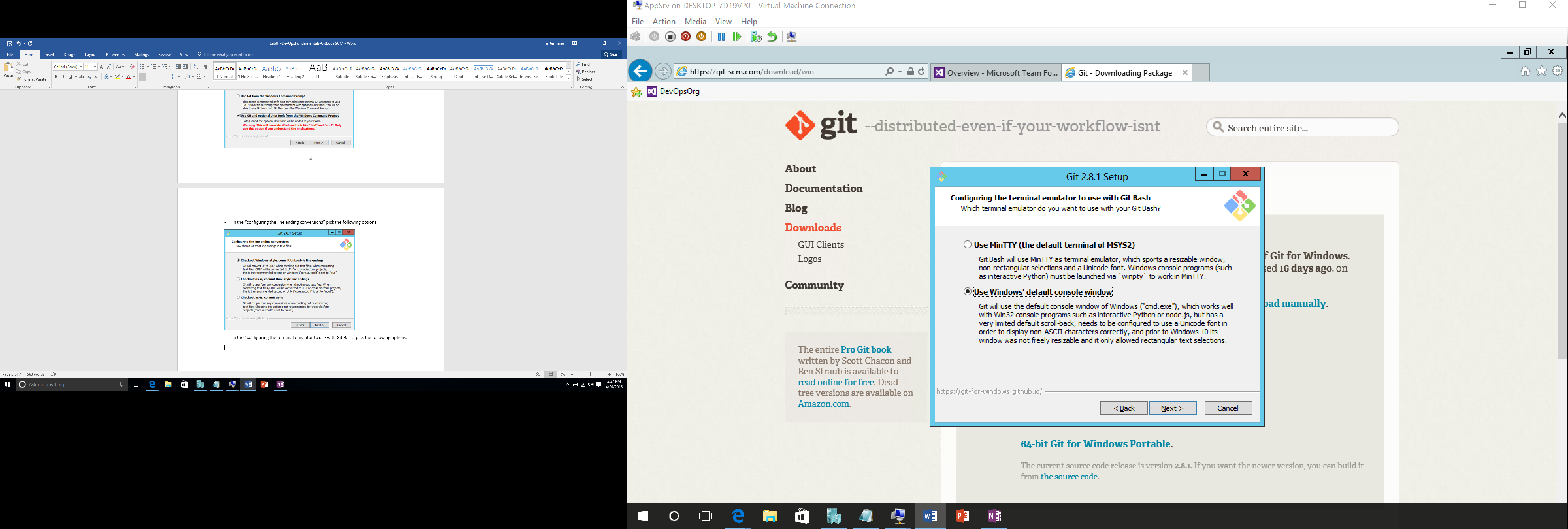
* Once in the Path environment screen select the following option:



* In the “configuring the line ending conversions” pick the following options:



* In the “configuring the terminal emulator to use with Git Bash” pick the following options:



* In the configuring extra options select both check boxes
* Uncheck “View Release notes” and check “Launch Git bash” in the very last step of the installation.
* Notice how you can resize the Git bash screen and use the mouse wheel with the CTRL key to make text bigger or smaller.

Next we are going to install a couple more utilities which will allow you to use Git directly from a PowerShell window is you wish to do so. The first tool is **PsGet** which allows installing packages directly PowerShell:

* Open a PowerShell window and type the following:
  + **Set-ExecutionPolicy RemoteSigned** (Choose Y)
  + (new-object Net.WebClient).DownloadString("http://psget.net/GetPsGet.ps1") | iex
  + Use the following command to install posh-git
    - Install-module posh-git

For the rest of the exercises, you can either use the Git Bash window or the PowerShell window.

# Exercise 2 – Getting started with git – Command Line Interface

Setting the environment (avoid copy paste, since the goal is to allow you to get familiar with the commands)

* Open the git bash window (the shortcut is placed on the desktop)
* Set the git environment information about your username and email
* git config --global user.name "John Doe"
* git config --global user.email johndoe@example.com
* Set core editor **if** **you have Notepad++ installed, otherwise skip this step**

git config --global core.editor "'C:/Program Files (x86)/Notepad++/notepad++.exe' -multiInst -nosession"

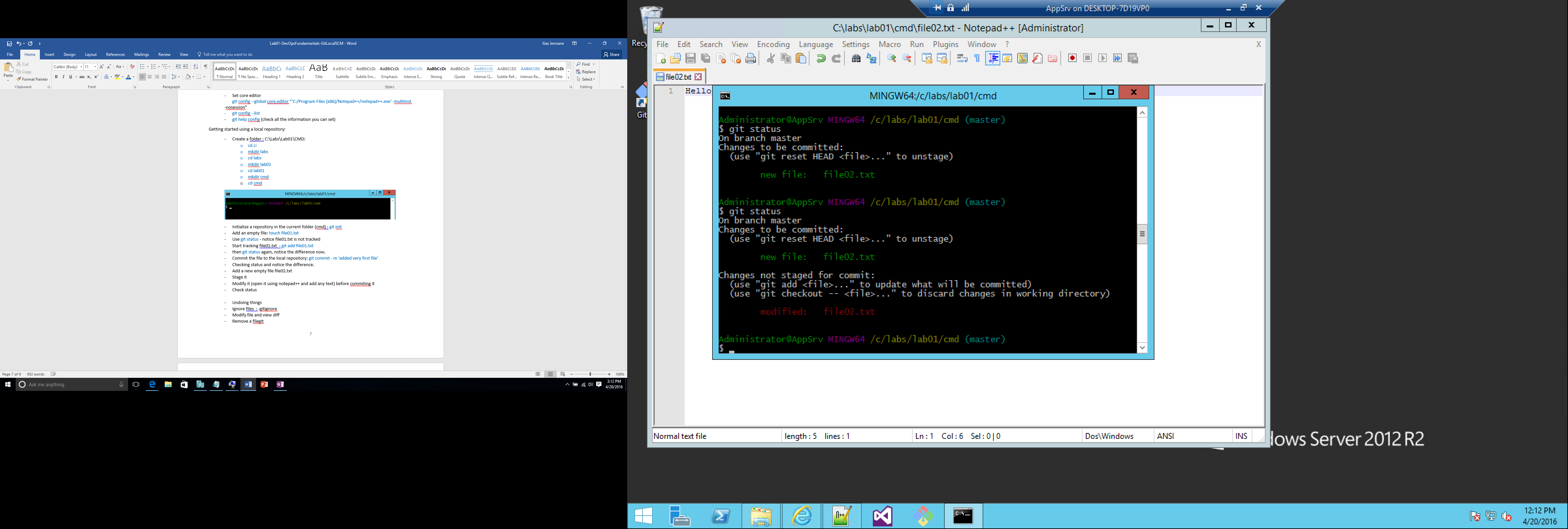
* git config --list
* git help config (check all the information you can set)

Getting started using a local repository:

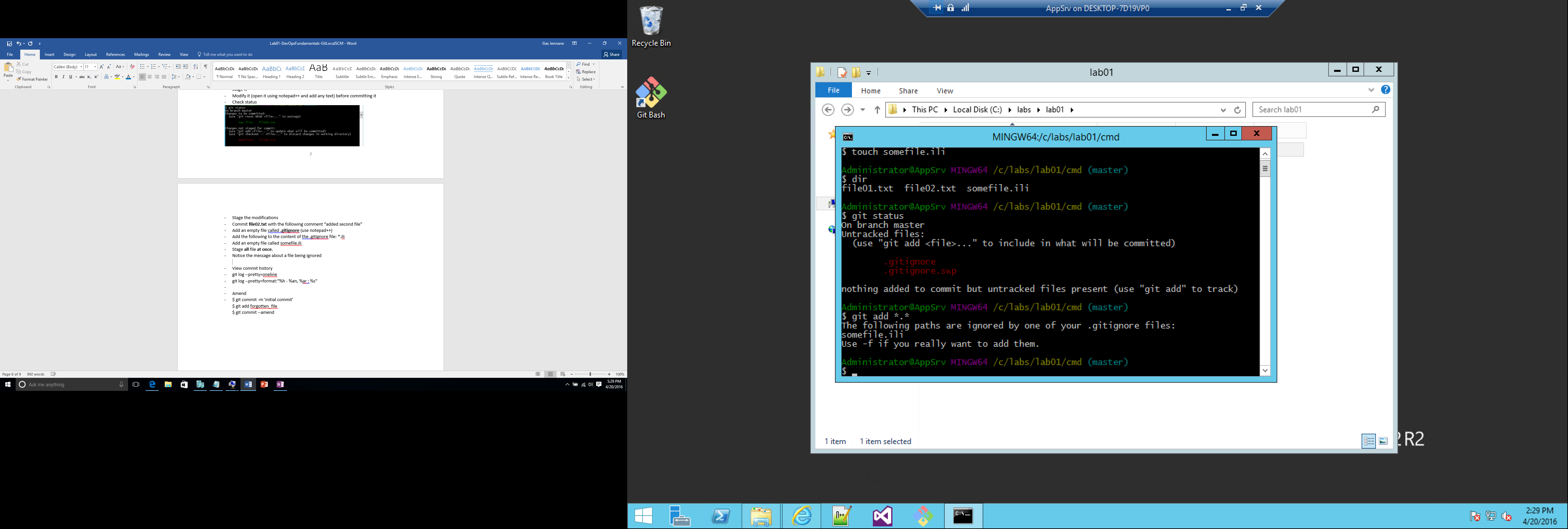
* Create a folder : C:\Labs\Lab01\CMD:
  + cd c:
  + mkdir labs
  + cd labs
  + mkdir lab01
  + cd lab01
  + mkdir cmd
  + cd cmd



* Initialize a repository in the current folder (cmd) : git init
* Add an empty file: touch file01.txt
* Use git status and notice how file01.txt is not tracked
* Stage file01.txt by using the command git add file01.txt
* then git status again, notice the difference now.
* Commit the file to the local repository: git commit - m 'added very first file'
* Checking status and notice the difference.
* Add a new empty file file02.txt
* Stage it
* Modify it (open it using notepad++ and add any text) and save your changes before committing it
* Check status



* Stage the modifications
* Commit **file02.txt** with the following comment “added second file”
* Add an empty file called **.gitignore** using command line
* Add the following to the content of the .gitignore file: \*.ili
* Add an empty file called somefile.ili
* Stage **all** files **at once** using git add \*.\*
* Use git status Notice the message about a file being ignored

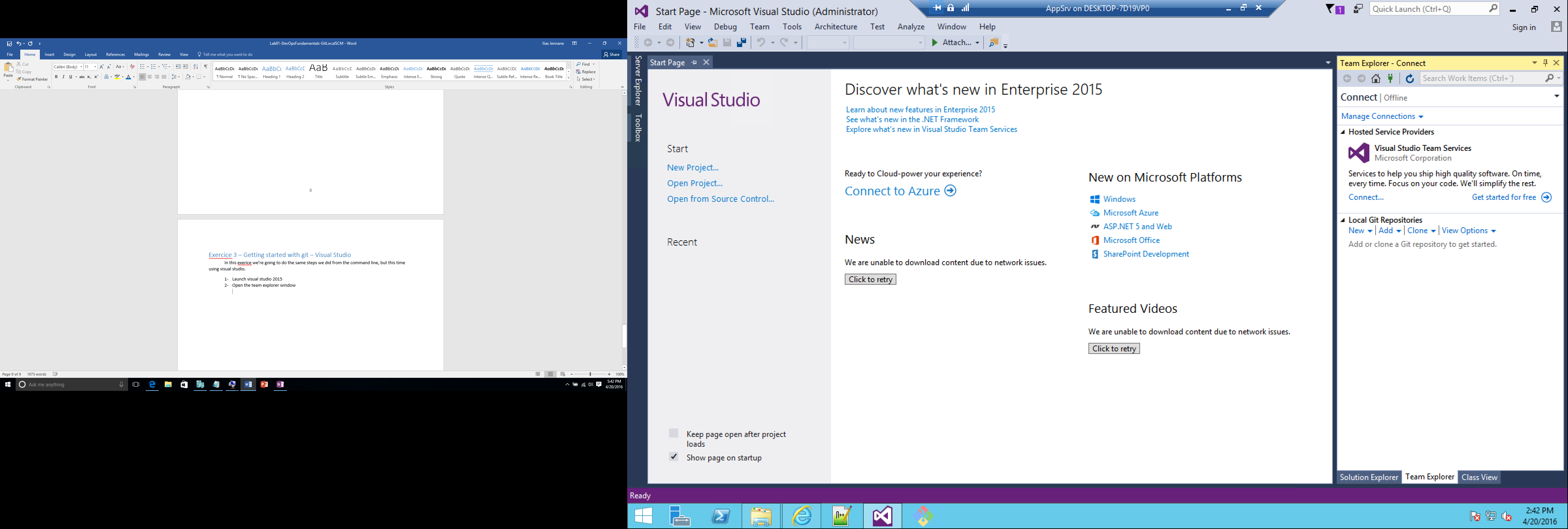


* Commit all files at once: git commit -a -m ‘added a gitignore file’
* View commit history git log --pretty=oneline
* Or git log --pretty=format:"%h - %an, %ar : %s"
* Add an empty file file03.txt, stage it then commit it.
* Open the file using notepad++ and add some text and save your changes.
* Stage the file once more
* Amend the previous commit using git commit --amend (You will be prompted to modify your comment)
* View history again, notice how the last two changes are recorded as one entry.
* Close the bash command line.

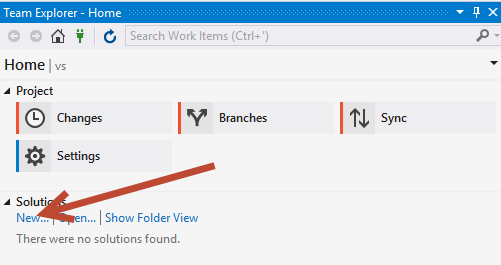
# Exercise 3 – Getting started with git – Visual Studio

In this exercise we’re going to do the same steps we did from the command line, but this time using visual studio.

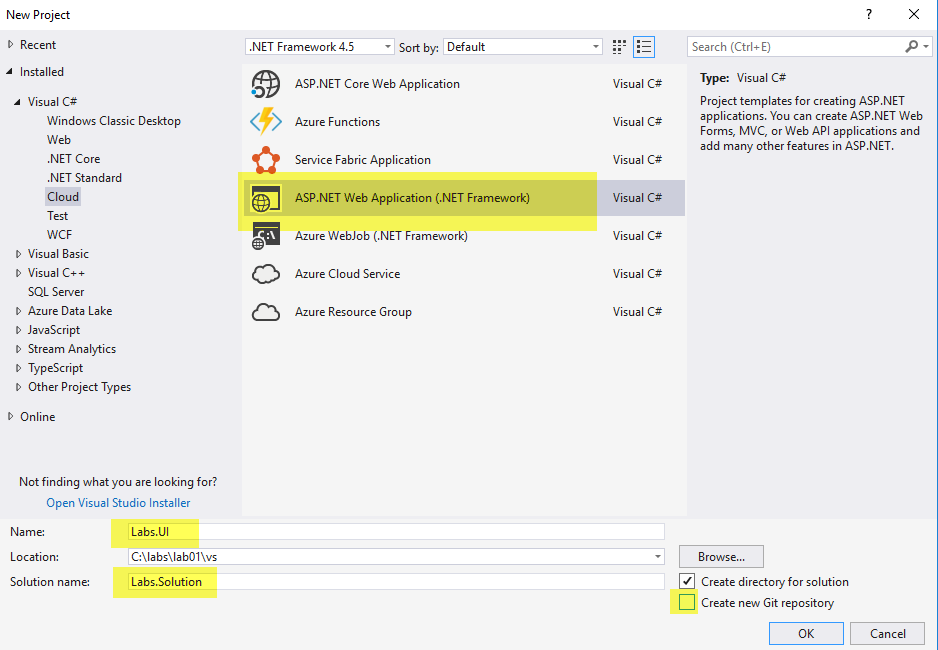
* Launch visual studio 2015 or higher
* Open the team explorer window. Click on the green plug icon to go to **Manage Connections.**
* Click New link under local git repositories.



* Use the following path to create a repository: **C:\labs\lab01\vs**
* Double click on the newly created repository
* Use the New link under solutions, it allows you to create a solution under the currently selected branch in the selected repository

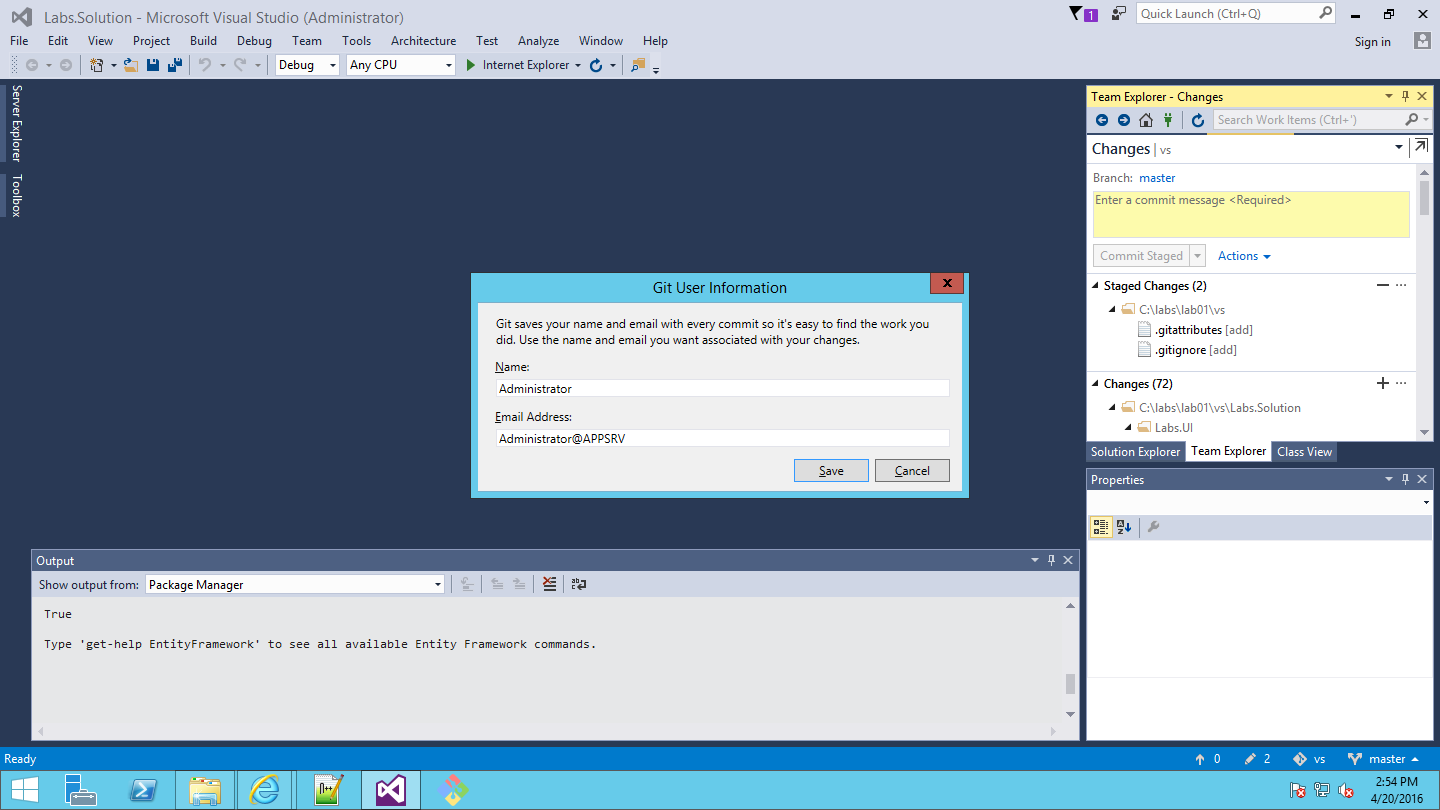


* Pick an asp.net web application
* Name the solution **Labs.Solution** and the project **Lab.UI**
* Uncheck Add Application Insights if that is an option
* Uncheck Create new Git Repository



* In the next screen pick MVC.

Under team explorer, click on the Changes button. (You may get promoted to enter your user information Enter a valid name and email (it will be used later to receive notifications):



* Notice that so far only the .gitignore files are already staged
* Stage the other solution files by clicking on the + button under changes
* Enter a message ‘Added initial UI project’ and commit locally using the Commit Stage button.
* Go back to Solution explorer
* Under Models add an **Employee.cs** class with the following properties:

namespace Labs.UI.Models

{

public class Employee

{

public int Id { get; set; }

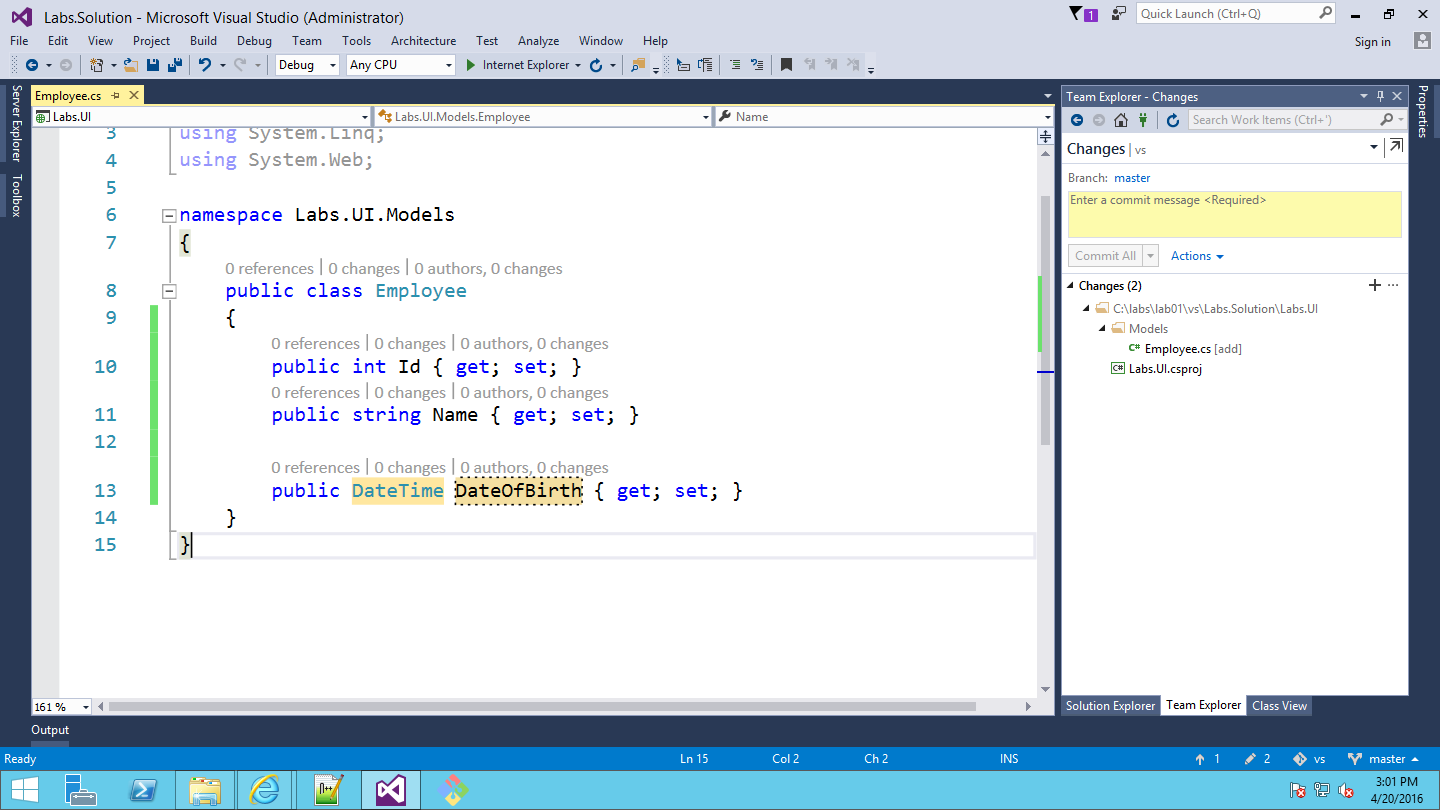
public string Name { get; set; }

public DateTime DateOfBirth { get; set; }

}

}

* Save the file.
* Notice how visual studio is indicating to you that there are uncommitted changes on the very bottom right of your screen. It also indicated which branch you are currently working on. (We will cover branches later):



* Click on the pencil icon showing two changes, stage them, then commit them using the following message ‘Added an employee model’
* Add a property called Age of type double to the class:

public double Age { get; set; }

* Stage and commit this change with the message **‘added an age field’**

Realizing that Age field must be computed you decide to modify it, but you want those changes to show as one single commit, so that if you push this code to your team they see it as once change.

* Add the following static method to the employee class:

public static int getAge(DateTime birthDate, DateTime laterDate)

{

int age;

age = laterDate.Year - birthDate.Year;

if (age > 0)

{

age -= Convert.ToInt32(laterDate.Date < birthDate.Date.AddYears(age));

}

else

{

age = 0;

}

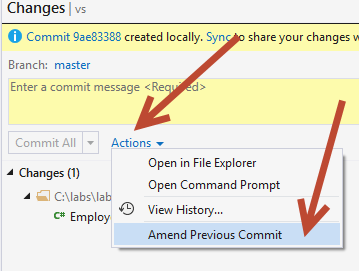
return age;

}

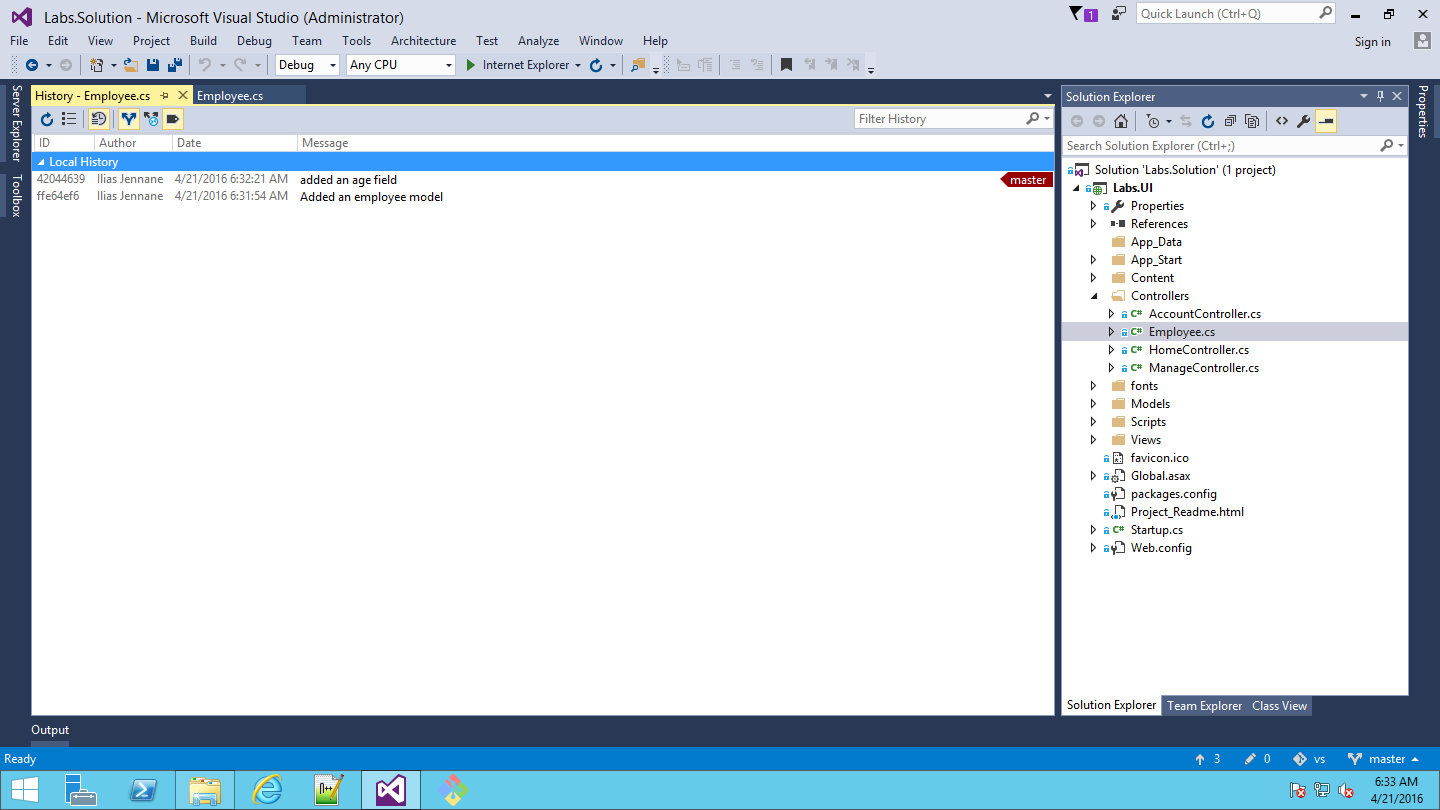
* Modify the Age property to be:

public double Age { get { return getAge(this.DateOfBirth, DateTime.Today); } }

* Click on the changes icon and in the commit screen click on Actions, then click on Amend previous commit (see screenshot below).



* Go back to the solution, right click on the employee class and view the history of changes for this class and see how the commit is represented as one change even though it was committed in two steps.



* Add a comment about the class describing what it does. Go the **Employee.cs** class and right above the class declaration type ///, then enter the following description:

/// <summary>

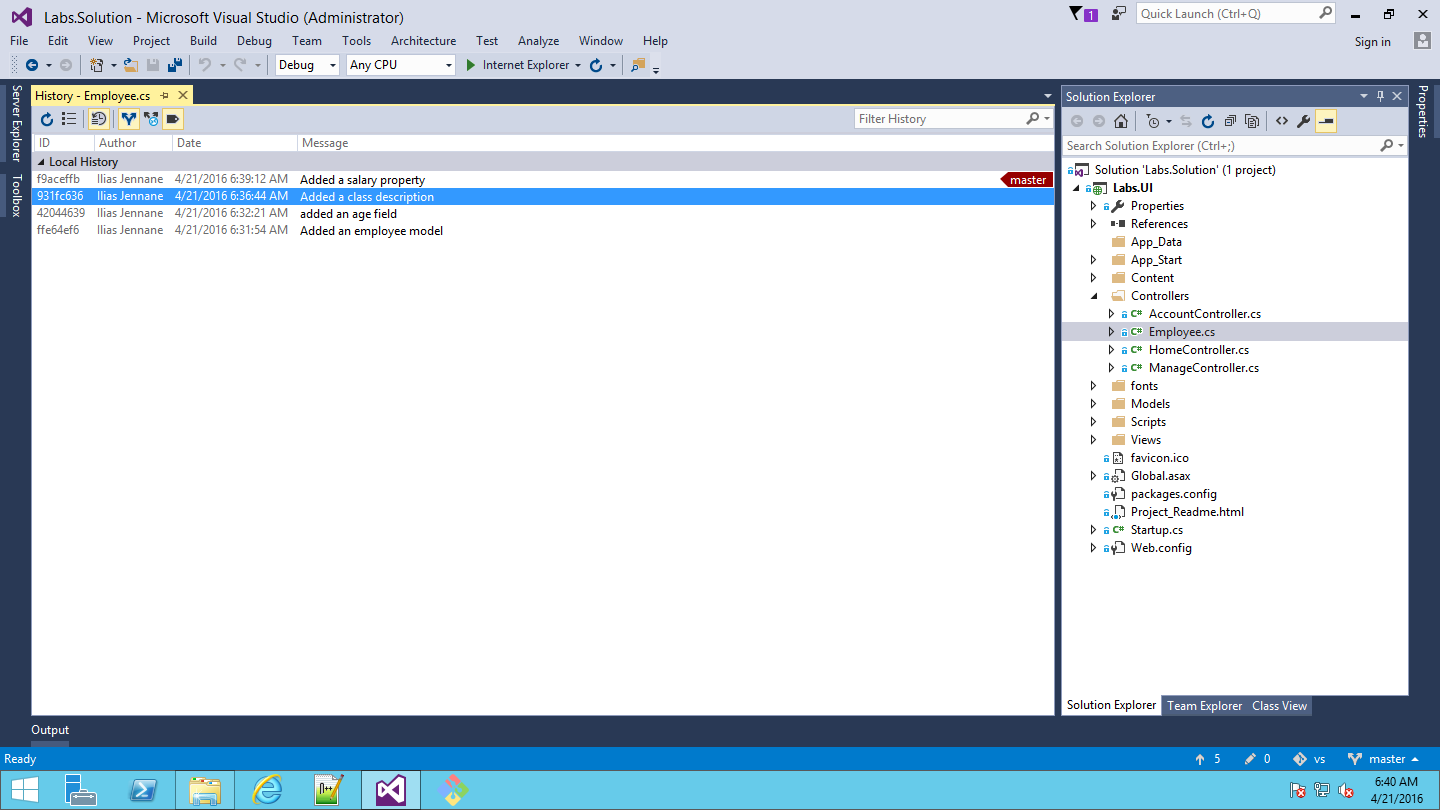
/// Represents an employee model

/// </summary>

* Commit this change, with the following comment ‘Added a class description’
* Add another property called Salary of type double and commit it with a comment ‘Added a salary property’

public double Salary { get; set; }

* Close the Employee.cs file.
* View the history of the employee class:



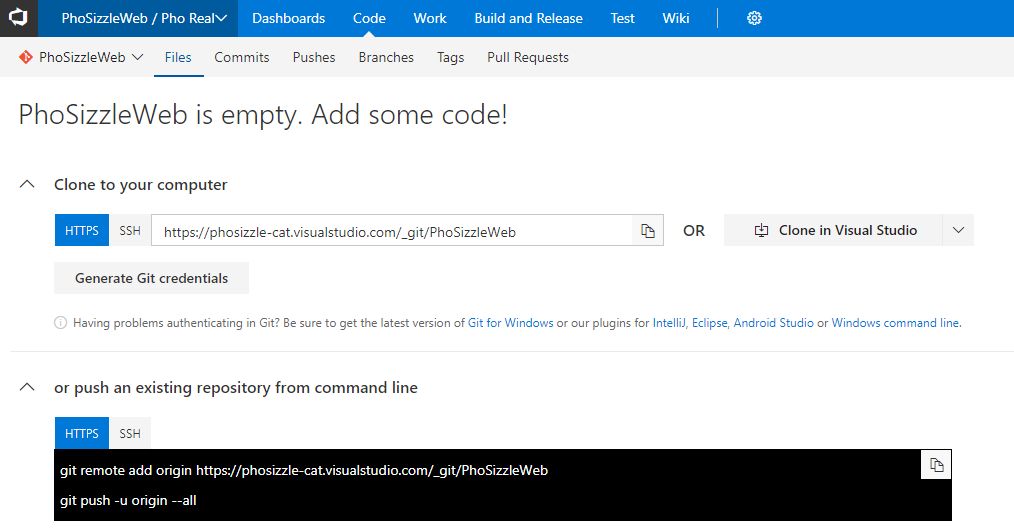
* Right click on the Added a class description and revert it
* Open the employee class now, you will notice that the comments are removed from the code.
* View history again and notice how the revert shows there as well.
* Committing all these small changes directly to a **remote** master branch and at this granular level is probably not a great idea once you collaborate with many teammates simply because you might be working on the Employee model class as a single deliverable. In the next tasks we will start team collaboration and evaluate a good branching strategy.

# Exercise 4 – Creating an initial master Remote branch

Jim is the solution lead and wants to structure how the team will collaborate. First stop is to create a Feature work item that will be use to track the work done for this specific requirement the team is working on.

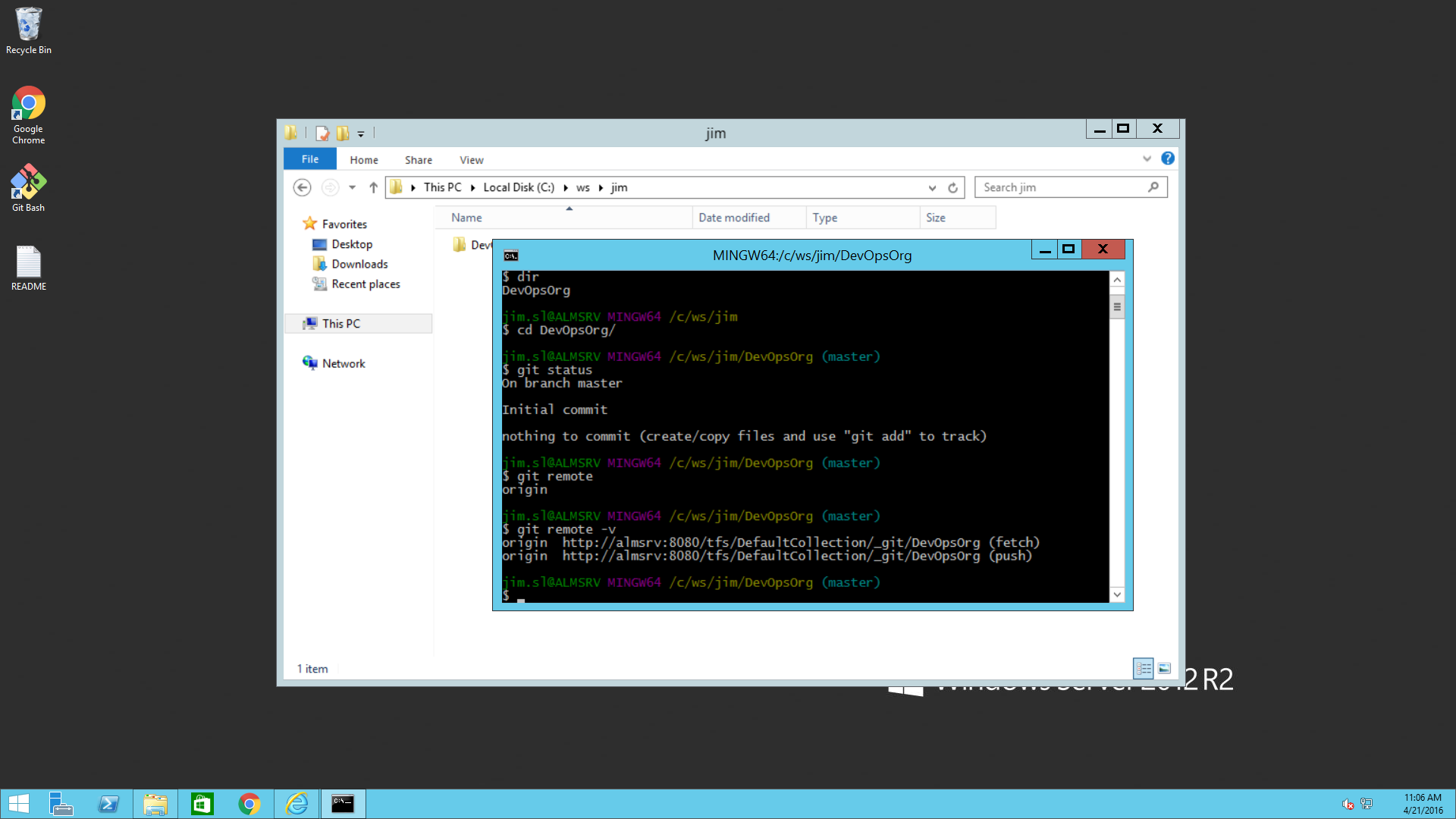
(This lab is not meant to show how to structure work items, but we will use few of them to show traceability and some source control management proven practices)

* Open a browser and go to your existing VSTS PhoSizzle account created in the previous labs.
* Choose the PhoSizzleWeb / PhoReal Team’s Code tab

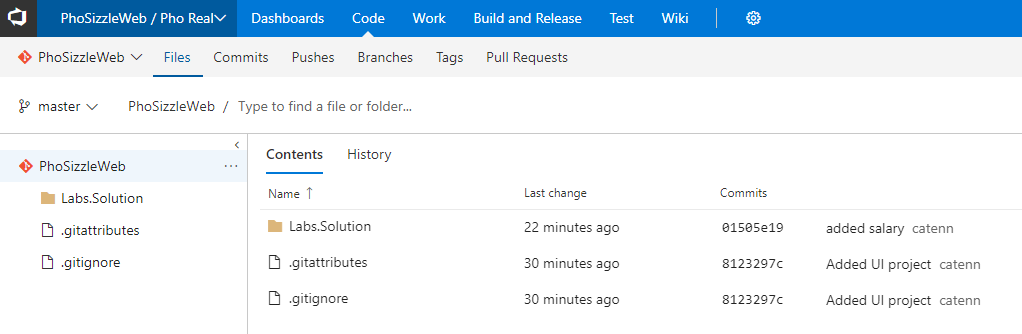


* In your Git command prompt or PowerShell, navigate to the Labs.Solution project you just created. cd C:/labs/lab01/vs
* Enter the first of two commands that are on your VSTS page to “push an existing repository from command line’ that starts with git remote add origin…
* Type git remote and hit enter
* To see which remote servers you have configured, you can run the git remote command. It lists the shortnames of each remote handle you’ve specified. You should see origin – that is the default name Git gives to the server you cloned from or default specified in the command from VSTS

You can also specify -v, which shows you the URLs that Git has stored for the shortname to be used when reading and writing to that remote, try running the command git remote -v:

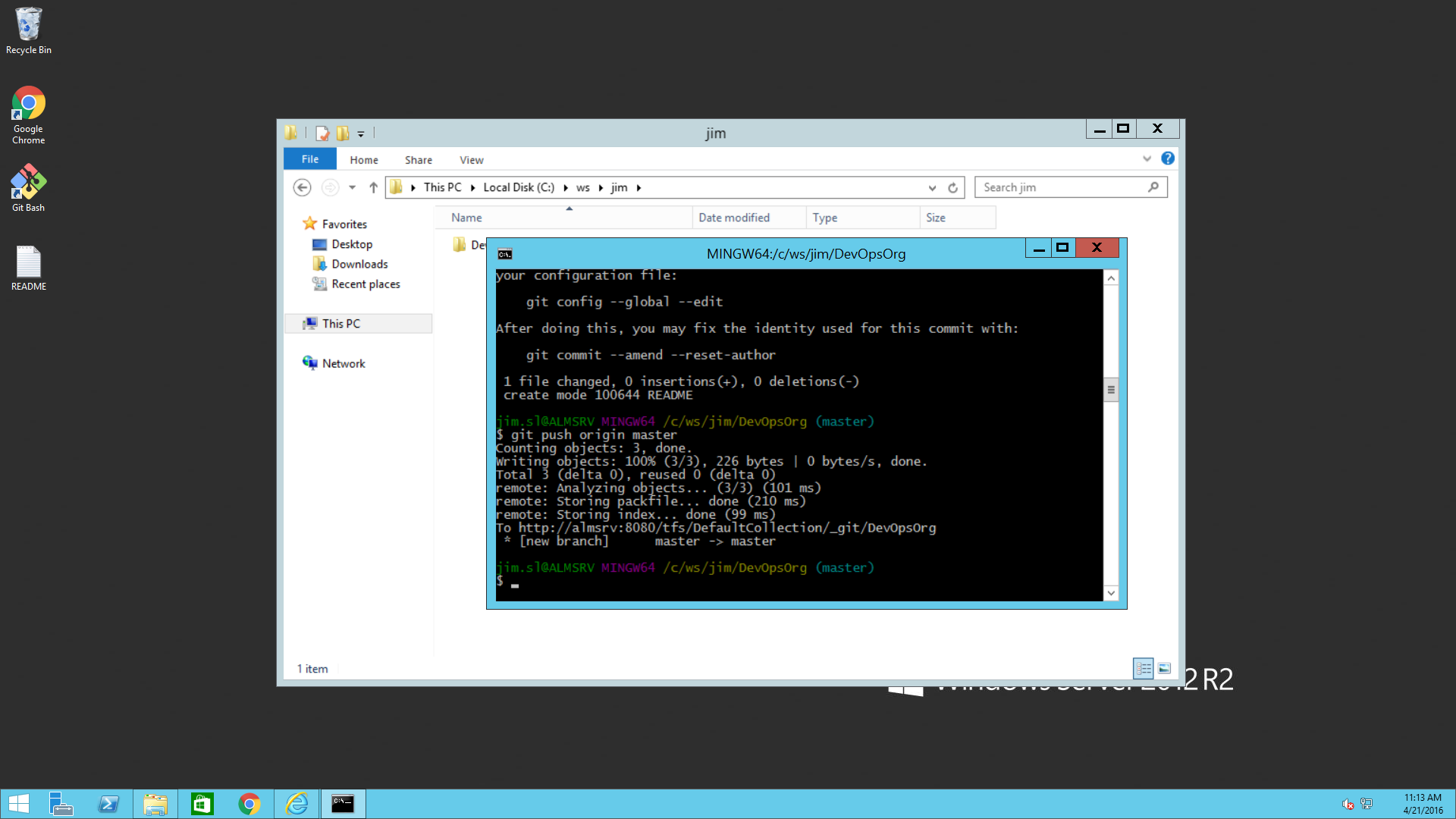


* Now, let’s actually push our code to VSTS. Run the second command from your VSTS page that says: git push -u origin –all
* Refresh your VSTS page and it should look like this now:

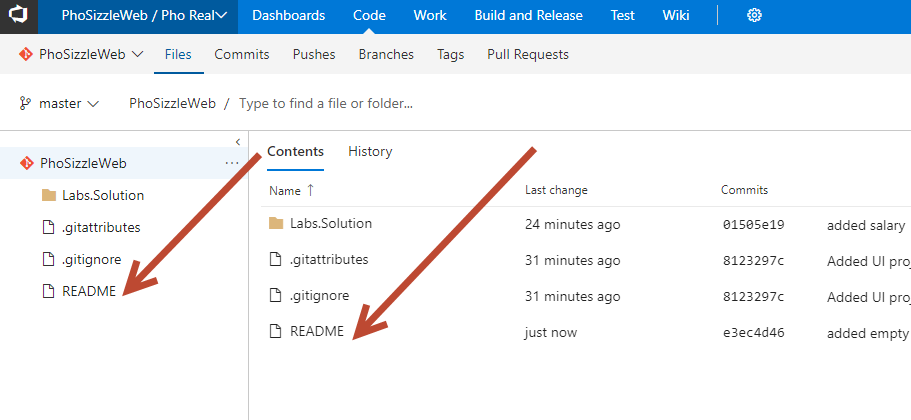


The first file the team leads wants to push to the repository is a readme file, use the following command to push it from the command line:

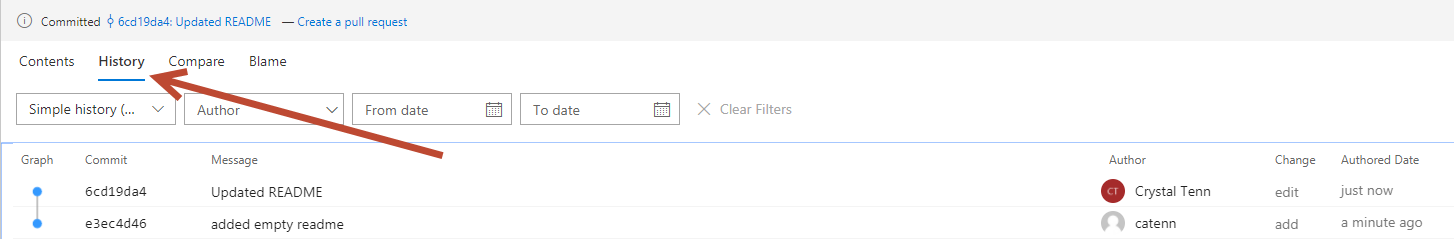
* touch README
* git add README
* git commit -m ‘Added an empty read me file’
* git push origin master



* Refresh VSTS.
* Open the VSTS code tab, you will notice that the readme file is now added to the repository in the master branch:

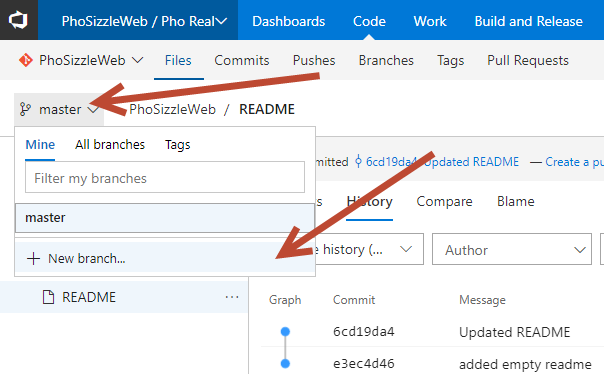


* Edit the file directly from the browser by using the “… “ link menu next to the README file when you hover over it, and select Edit (this is use for some lightweight editing). Add the following text “Add a feature branch for every feature you are working on and make sure it mapped to the feature work item.” Then hit commit.
* Click on the history tab:

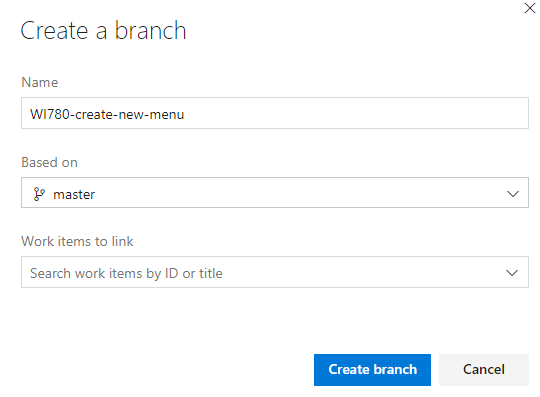


# Exercise 5 – Creating a feature branch

* Click on Master branch, on the dropdown, hit New branch.



* This will create a source code management branch for it. **Name the branch using a name that will make it easy to map to the work item**. Examples:
  + FB780
    1. FB = feature branch, where 780 is the ID of the feature work item that will be contained in this branch
  + WI780-create-new-menu
    1. WI = work item, 780 = ID of feature work item, plus name of work item

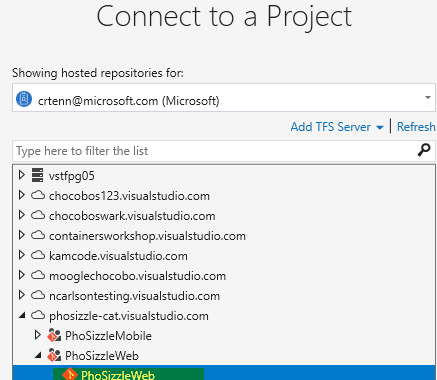


# Exercise 6 – Pushing changes to a remote branch using Visual Studio

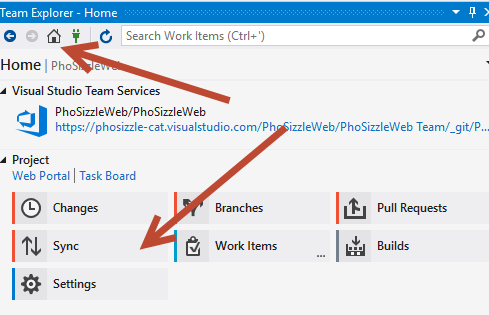
* Go to Visual Studio
* Go to Team Explorer
* Click on the manage connections icon



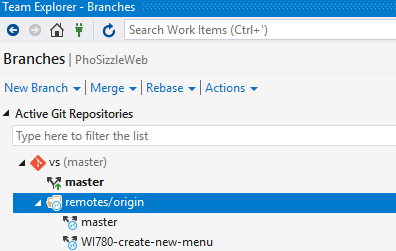
* Choose your PhoSizzle > PhosizzleWeb > PhoSizzleWeb project:



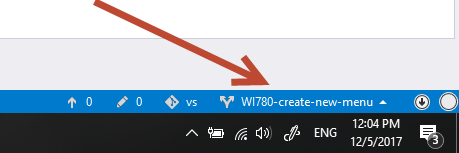
* Leave your local path as C:\labs\lab01\vs
* Hit Connect
* Now your Team Explorer will be connected to your VSTS account.
* Click on the Home button for your Team Explorer.



* Click Sync
* Click Fetch (this will get all remote branches)
* Click Home
* Now choose Branches
* Click on the down arrow on remotes/origin
* Click on Branches it should show you the follow branches:



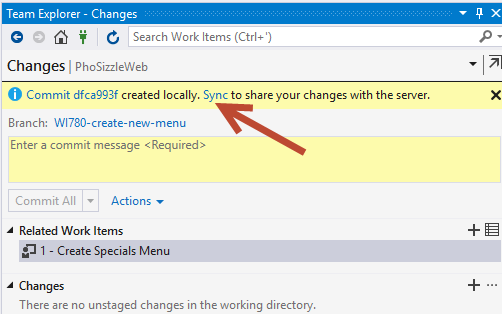
* Double click on WI780 (The name might be different based on what you used to create the feature branch), notice how the bottom indicator has changed to reflect the current branch. (It might not look like anything changed since this branch is a clone of master branch, but you can see the branch you are in on the bottom right of Visual Studio)



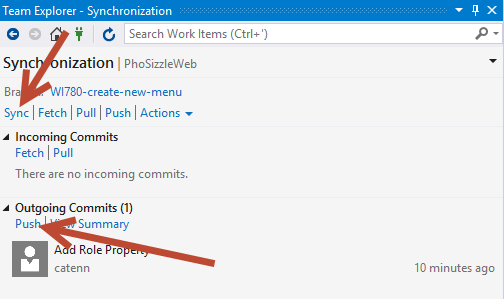
* Let’s add a new line to your Employee.cs class.

public double Role { get; set; }

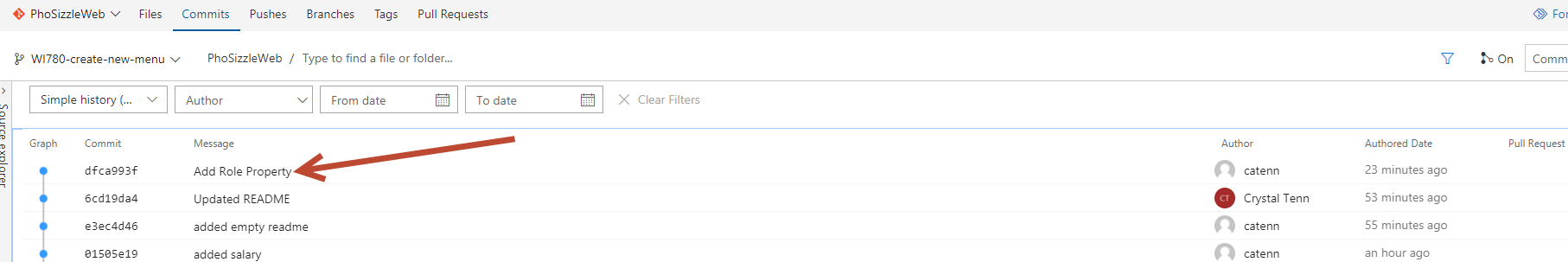
* Click on the changes indicator at the bottom
* Stage all changes
* In the related work items, enter the work item ID “1” to show how to add work items (Note the ID isn’t going to actually match the ID of your work item in this scenario, but realistically at work they will match. You should have a work ID 1 from the previous lab module). If you would like the Work ID to match what you are doing, create a new Work ID in VSTS for your work and reference that ID.
* Enter the following commit comment: “**Added Role property**”
* Hit commit
* Click on the Sync text above your commit to navigate easily to the Sync page, or hit Home and Sync.



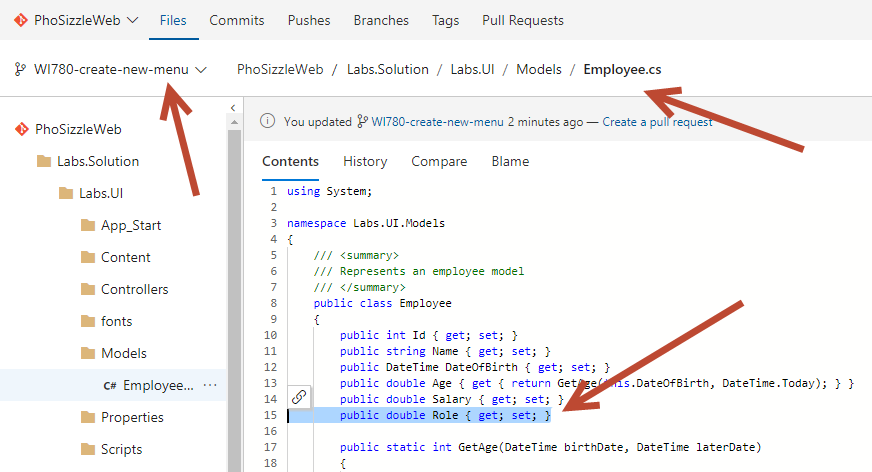
* You have some choices here.
  + Sync: will do a fetch, then pull, then push if no merge conflicts
  + Fetch: will get any remote commits and show them to you (but not put them into your branch) and will get any remote branches and show them in your Team Explorer
  + Pull: Will do a fetch first then automatically pull in any code to your branch if no merge conflicts
  + Push: Will push your code up to the VSTS account assuming no merge conflicts. Will not allow you to do this if there are pending commits to pull in.
* Since there are no other changes (since no one else is in your VSTS account), you can either hit Sync or Push.



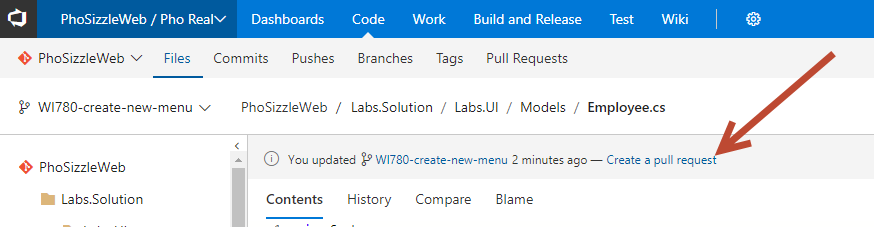
* Go to the Code section in the web portal in VSTS.
* Make sure that your changes are pushed to the Feature branch.
* You can check your Commits to see if the commit is there:



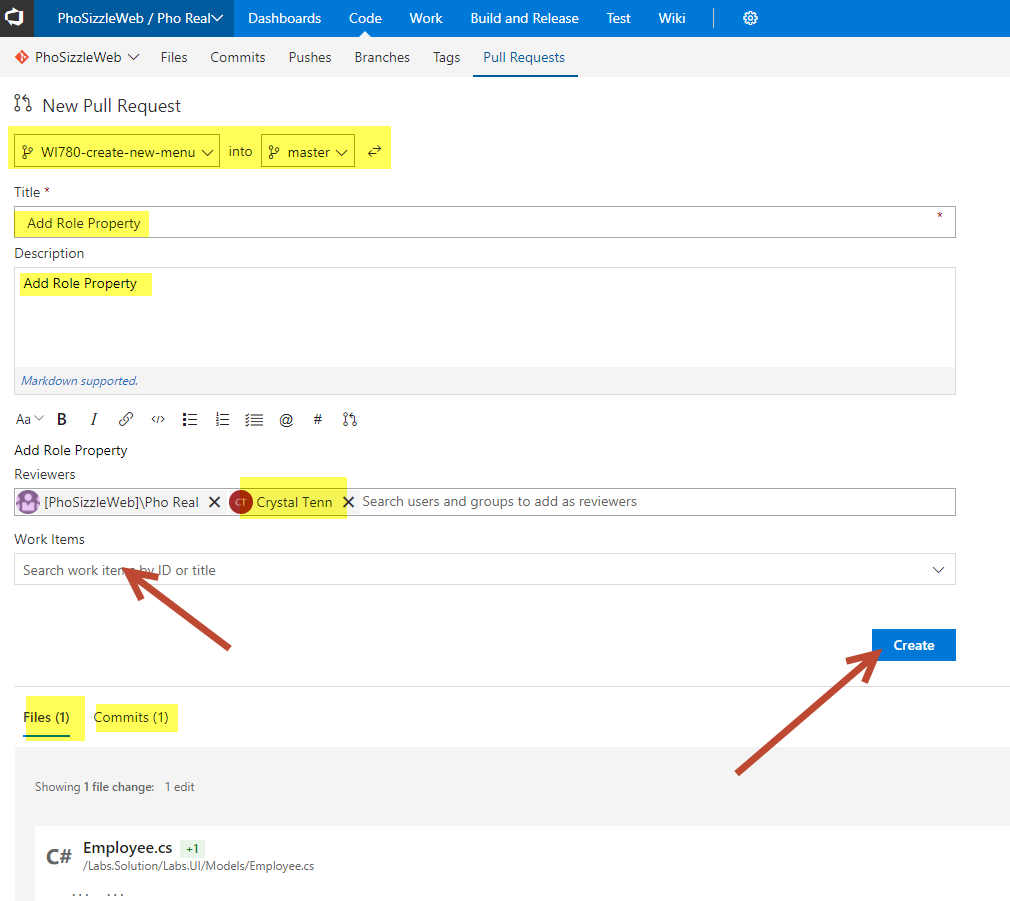
* You can also check your actual source control File. Make sure you are in your feature branch and go to your file:



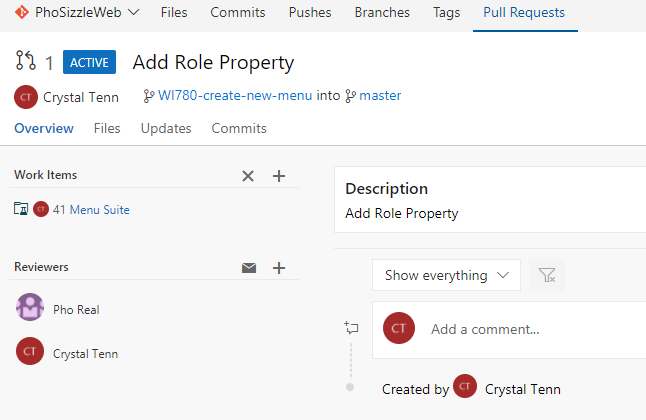
* The portal indicates that you can create a Pull Request to send your changes to the master branch after they go through a review. Click “New Pull Request”:



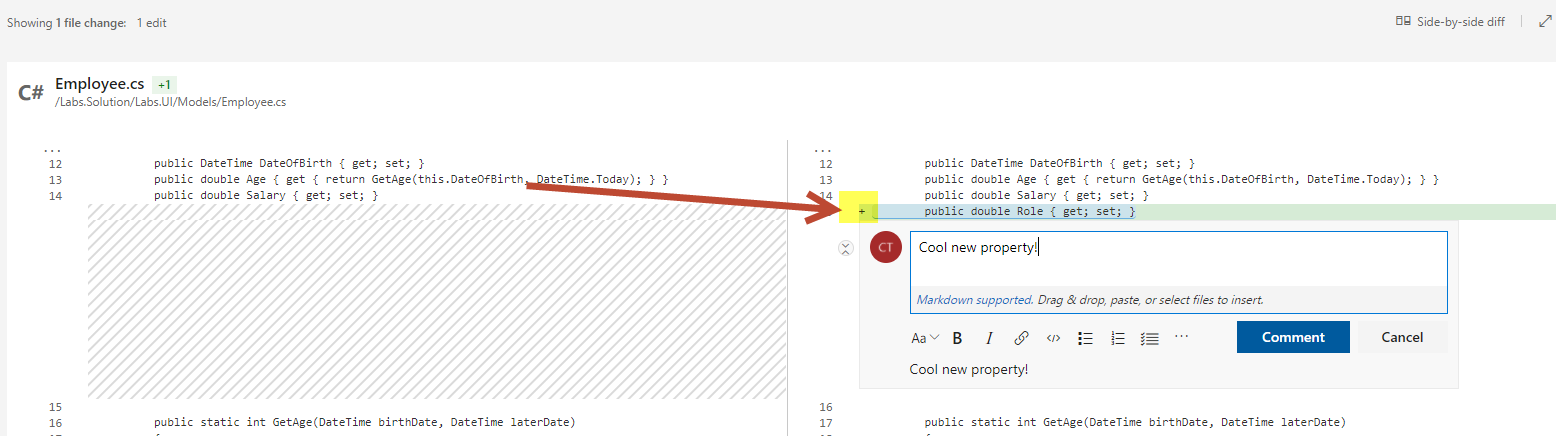
* There is a lot on this page! Make sure that you are asking the Pull request to put your Feature branch INTO master. You can change your title or description. Add yourself as a Reviewer. Add associated work items (can choose random ones). Check the files and Commits summary on the bottom of the page.



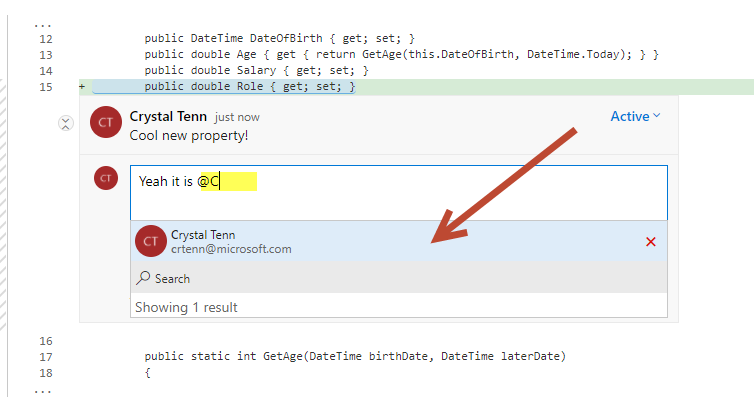
* Click Create when you are done.
* From this page, you can see who your reviewers are, files involved, updates, commits, and comments.



* Click on Files.
* Add a comment by clicking the “+” sign to the left of the line you want to add a comment to.



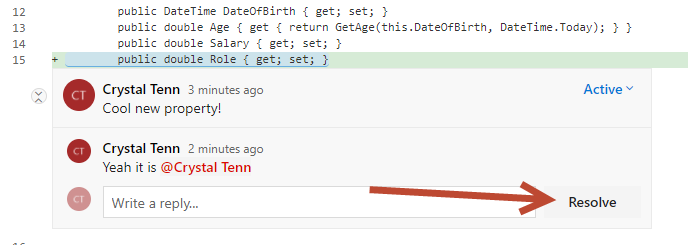
* Add a reply and use the @ sign to add yourself. This will send a notification out to your email in a minute that you have been called out in a post so you can go see it. This is how to make sure someone looks at your comment!



* Notice at the top, you will see that 0/1 comments are resolved.



* Click Resolve on the comment to resolve. This is a great way to keep track of which comments you have fixed.



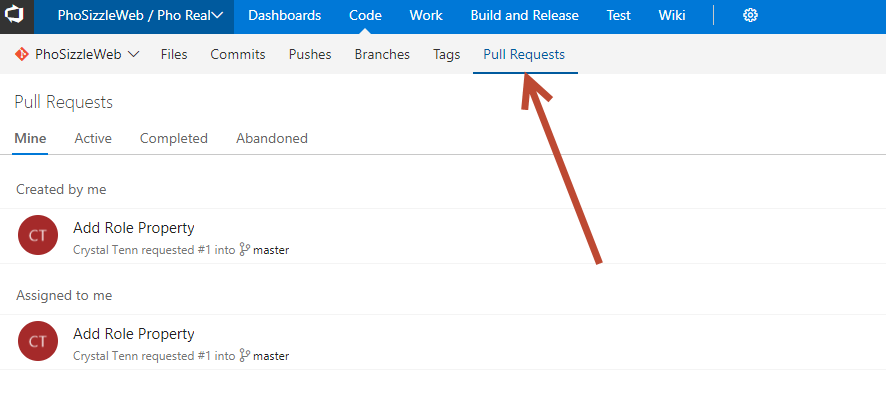
* It should say All Resolved at the top when you are done.
* Click on the Updates and Commits tab to see what is in it
* Go back to Visual Studio, add another property.

public double Role { get; set; }

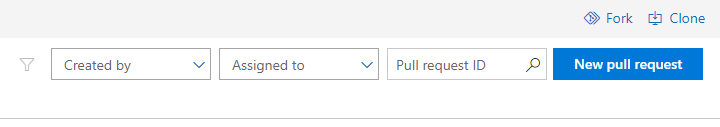
* Commit and Push or Sync that into your VSTS feature branch.
* Refresh / go back to your pull request. You should see two commits now and both changes.
* The pull request is for your whole feature branch, so any changes go into here.
* If someone asks you to fix something in your feature branch before completing your pull request, then you should make the changes and push it into the feature branch.. and then you can have them check the changes again before it gets completed. Once the pull request is completed, then the changes will be merged into Master.

# Exercise 7 – Approving Pull Requests

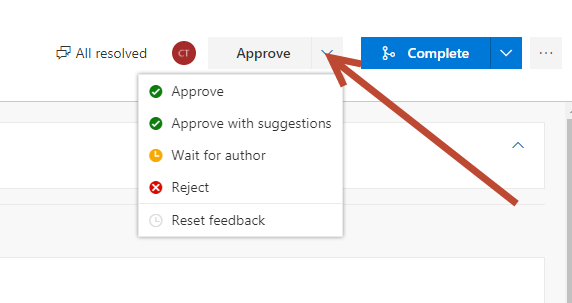
* Click on the Pull request Tab



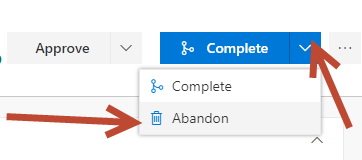
* To the right is how you can create new Pull Request or filter through existing ones.



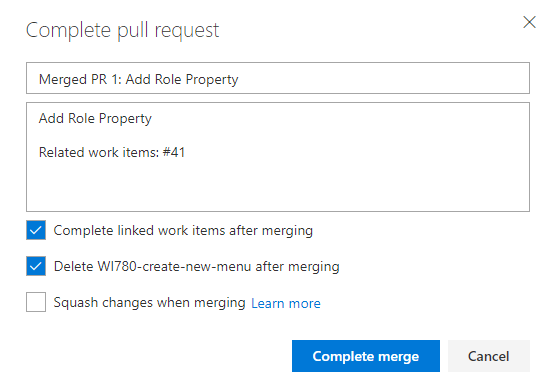
* Pull Requested **Created by you** are the ones you made for your branches that are ready to be merged in and reviewed by your team. **Assigned to Me** are pull requests that you need to review (in this scenario you both created it and assigned it to you, but typically you wouldn’t do that for your own requests when you have many people in your team).
* Click on your Pull Request.
* Review the changes.
* If you click on the down arrow, you can Approve now or see the other options. Approve with suggestions means that you Approve, pending that the person reviews the comments you made.



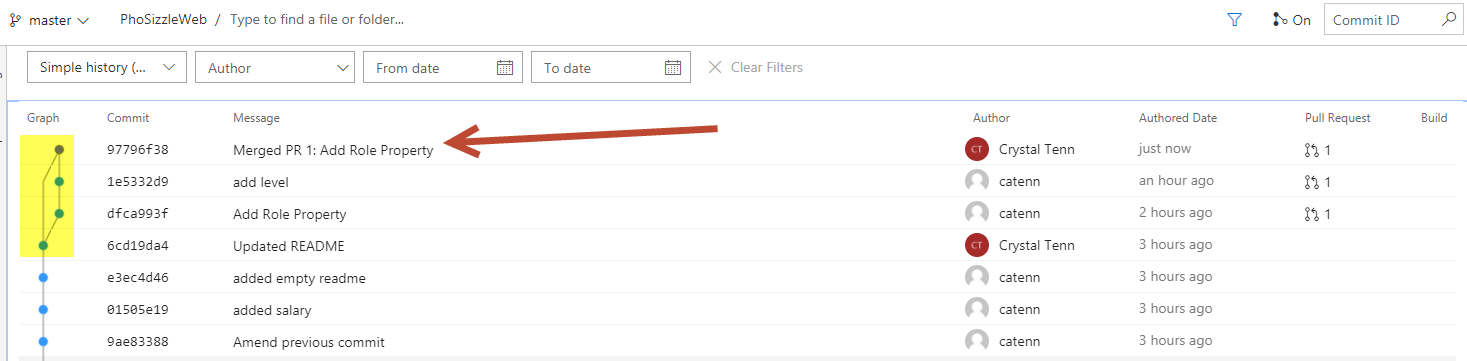
* Choose Approve.
* For your reference, there is an option to Abandon the pull request which will just remove this request. The feature branch will still exist, and no code will be affected whatsoever. This request will just be removed from VSTS. You do not need to select this, it is just for your knowledge.



* Click Complete. Typically the person who created the Pull Request to begin with will Complete it, but others on the team can if it’s very urgent of if the team lead / architect wants to be responsible for doing this.
* Leave the defaults.
* Hit Complete Merge.



* Click complete merge.
* Verify that the merge was successful
* View the **Commits** for the **master** branch, it should look as follows:

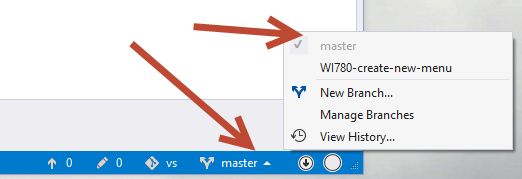


# Exercise 8 – Resolving Conflicts

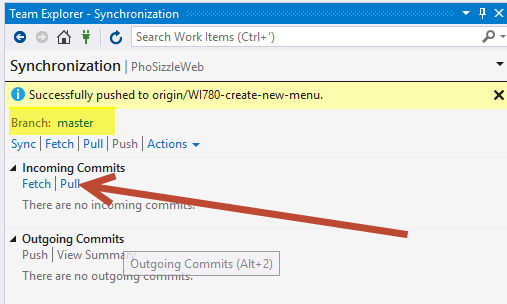
You will get a conflict if two different branches have changes to the same file.

Let’s create a new branch (we will learn how to do this in Visual Studio now instead of VSTS. Note you can also do this using the Git command line.), and edit the HomeController in our feature. Then let’s edit the HomeController online in VSTS to practice editing code on VSTS itself. This will trigger a merge conflict when we try to complete our Pull Request.

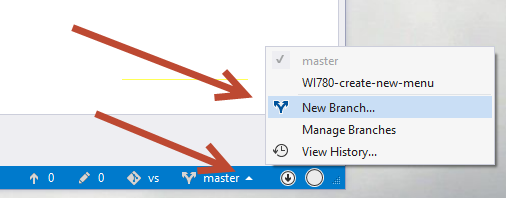
* Open visual studio
* Click on the up arrow on the blue toolbar in VS. Change to your master branch



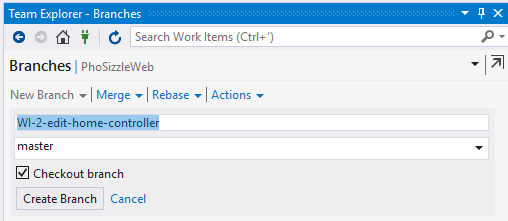
* Check your Employee.cs code, note the new changes are not here locally yet (no Role or Level property)
* Pull on master branch



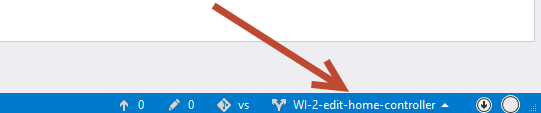
* Check your Employee.cs code, note new changes should be here with Role and Level property
* Click New Branch on your bottom blue VS menu



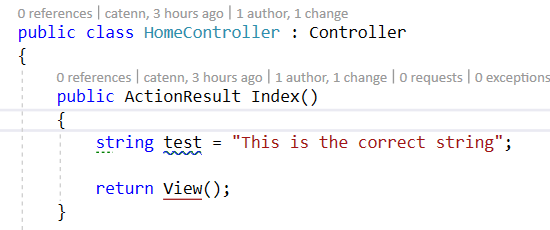
* Call it “WI-2-edit-home-controller”. Base it off master. Checkout the branch means that once you create it, it will automatically change you to that new branch. Hit Create Branch.



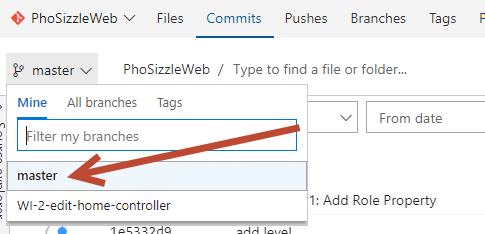
* Check the bottom VS toolbar. Verify you are in the new branch you made.



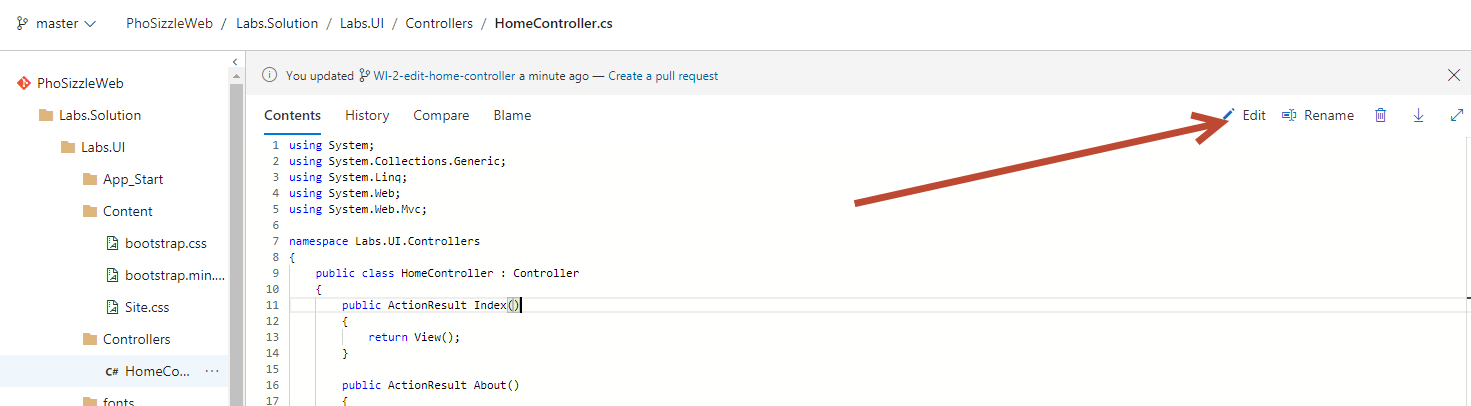
* Go to the HomeController. Edit the Index method so it looks like this:



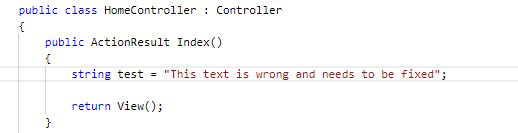
* Commit with a message.
* Sync or Push this change.
* Verify it shows up in VSTS in your new branch.
* In VSTS, swap to your Master branch now.



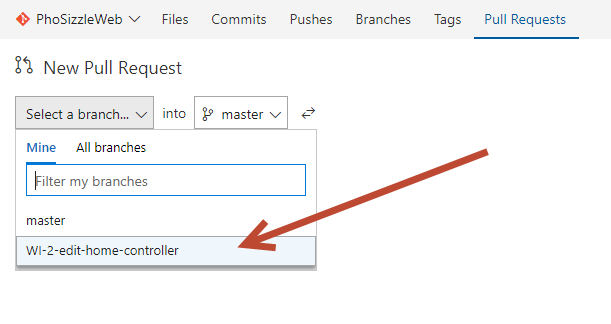
* Click Files > Labs.Solution > Labs.UI > Controllers > HomeController. Click Edit



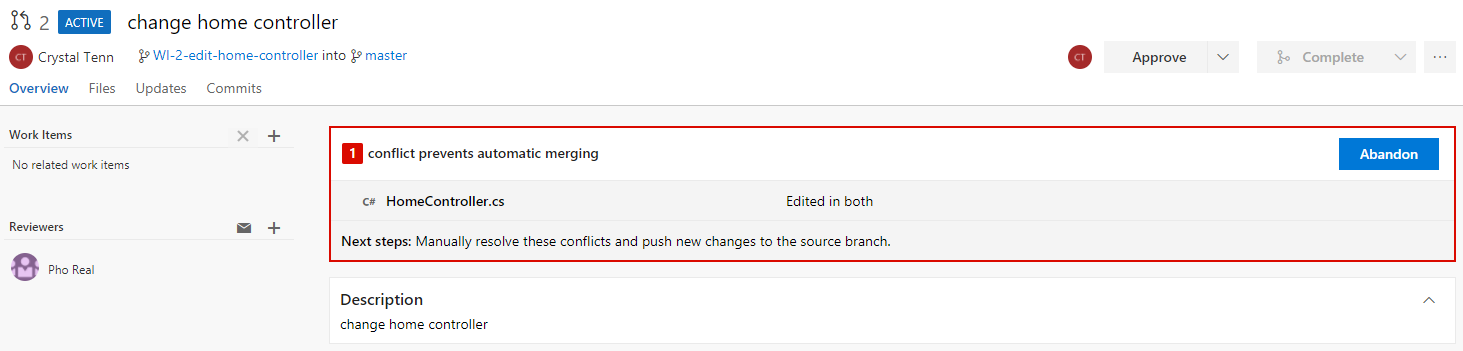
* Edit the Index method to look like this:



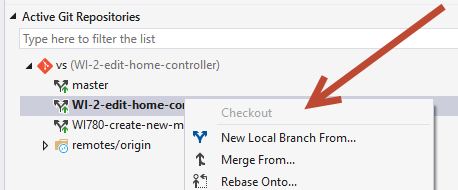
* Hit Commit
* Edit description if desired on popup, then hit Commit again.
* Click on Pull Requests tab
* Click New Pull Request.
* Select to move your feature branch into master



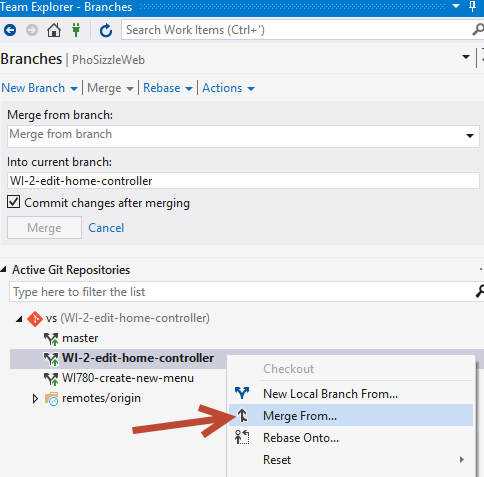
* Hit Create
* See merge conflicts:

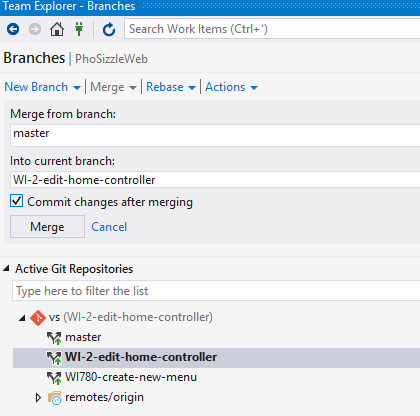


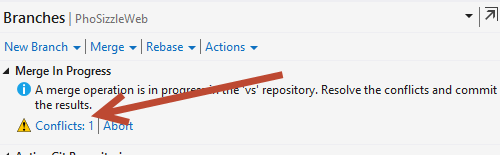
* All this means is that you need to resolve the conflict in your local VS solution first, commit the change, and then the branches will be synced up.
* Go back to Visual Studio.
* (You can skip this, but in a real scenario pull latest on your current branch)
* Using the VS blue bottom toolbar: Swap to master branch and pull latest there too. You **have to do this step** otherwise your local master branch will not have the changes you made in VSTS.
* Go to Team Explorer and hit Home. Hit Branches.
* Swap back to your feature branch by right clicking the branch, hitting Checkout, then it will be greyed out and bolded when it has swapped:



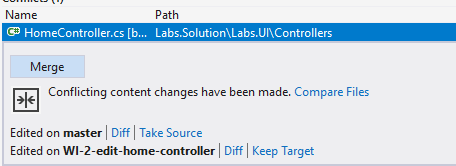
* Right click on your feature branch and hit Merge from:



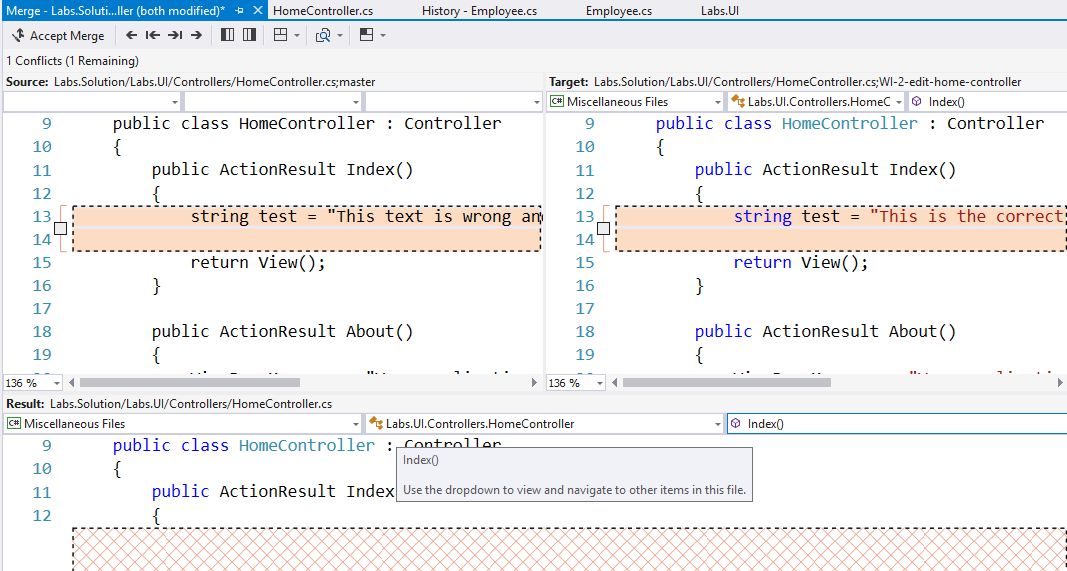
* We will merge from MASTER INTO our current branch. We want to get the most up to date version of the branch we will merge to later into our branch now with our changes, that way there will be no merge conflicts. The merge will say it is “up to date” if you did not get the latest from the master branch a couple of steps ago.
* 
* Click Merge
* We will get a note about a Merge Conflict.
* Click on the Conflict text



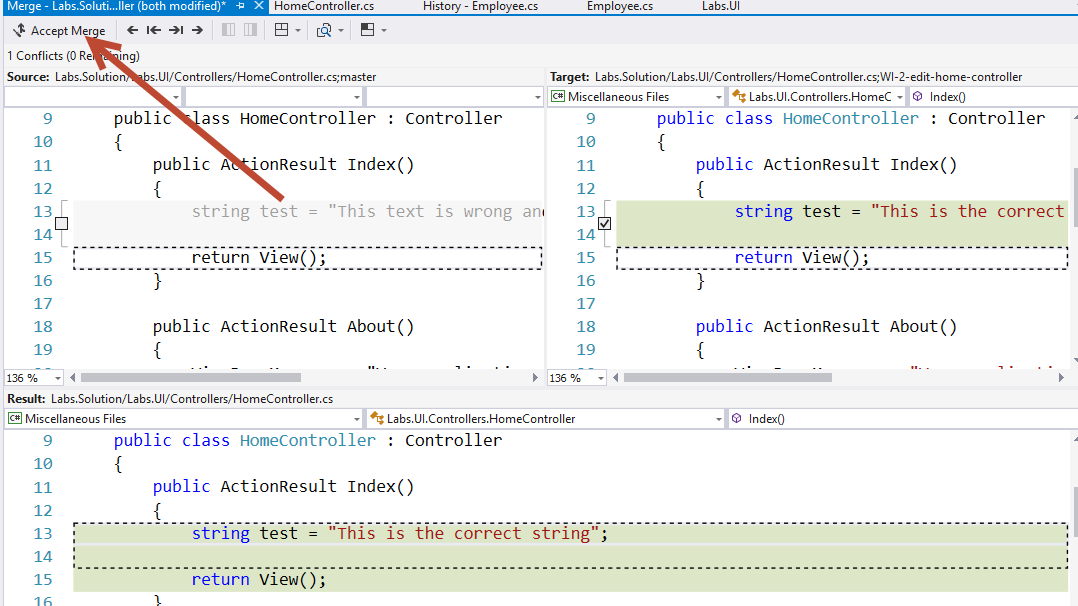
* Click on the file to select it



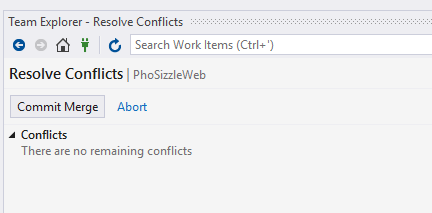
* Hit Merge
* You will see a screen like below. Left side is your master branch. Right side is your feature. We want the updated feature change in this case. The bottom pane is what will be the final change. You can also manually type / delete things in the bottom pane if you need to change anything else specific.



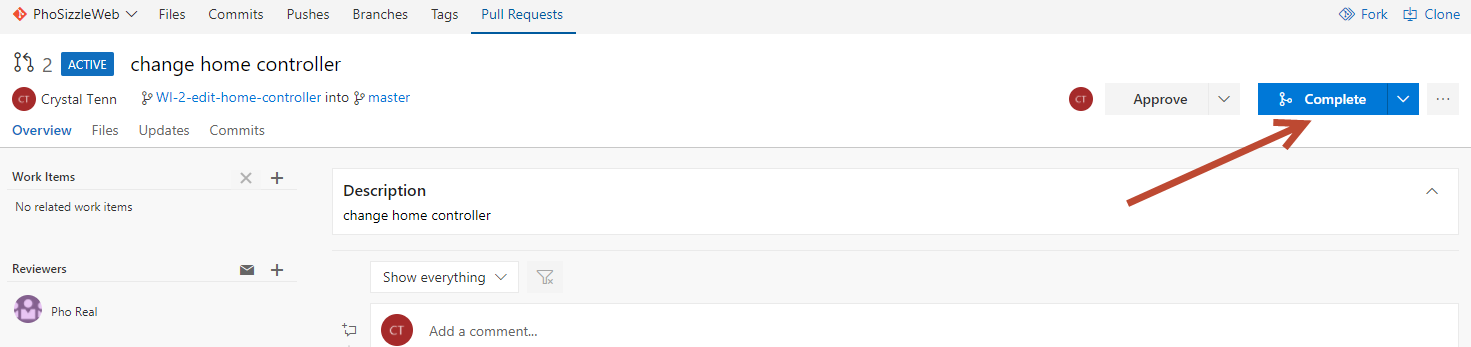
* Check the right box, your screen should look like this. Then hit Accept Merge.



* On Team Explorer, hit Commit Merge



* Write a message, hit Commit All.
* Sync your changes to VSTS by Pushing or Syncing.
* Go back to VSTS.
* Refresh your Pull Request Page. The merge conflict is now gone. You can now Complete the Pull Request.



* Check to make sure your changes are in the master branch (the commit and the code).