git Workshop - Basic

Lab Version: 2.1

Last Updated: 6/28/2019

Approximate Time Required: 90 Minutes



Table of Contents

[Overview 4](#_Toc12606297)

[Prerequisites 4](#_Toc12606298)

[About the Developer Scenario 5](#_Toc12606299)

[Authentication 5](#_Toc12606300)

[Exercise 1: Configure git 6](#_Toc12606301)

[Task 1: Set global git configuration. 6](#_Toc12606302)

[Task 2: Change editor used for interactive commit messages, squash, etc 6](#_Toc12606303)

[Task 3: Ensure set up for Windows Secure Channel Library connection to DevOps Server 7](#_Toc12606304)

[Task 4: Visually edit global and system configuration 8](#_Toc12606305)

[Exercise 2: Configure Visual Studio to access remote git repo 9](#_Toc12606306)

[Task 1: Manage Connections in Visual Studio 9](#_Toc12606307)

[Exercise 3: Set up to do work on your local machine 10](#_Toc12606308)

[Task 1: Clone the gitWorkshop repo 10](#_Toc12606309)

[Exercise 4: Get ready to work 11](#_Toc12606310)

[Task 1: Create a local feature branch in Visual Studio 11](#_Toc12606311)

[Task 2: Push your feature branch to DevOps Server 13](#_Toc12606312)

[Exercise 5: Make some changes to your feature 14](#_Toc12606313)

[Task 1: Add a c# class file. 14](#_Toc12606314)

[Task 2: Add another file. 14](#_Toc12606315)

[Task 3: Modify a file that other developers may be modifying 14](#_Toc12606316)

[Task 4: Use git stash to create new repo and move pending changes there. 14](#_Toc12606317)

[Task 5: Add the changed files to the index of files to be committed. 16](#_Toc12606318)

[Task 6: Commit your changes. 17](#_Toc12606319)

[Task 7: Push your feature branch to DevOps Server 18](#_Toc12606320)

[Exercise 6: Resolve merge conflicts in your feature 19](#_Toc12606321)

[Task 1: Check out the develop branch. 19](#_Toc12606322)

[Exercise 7: Finish your feature 21](#_Toc12606323)

[Task 1: Create Pull request 21](#_Toc12606324)

[Task 2: Complete the Pull Request setup 21](#_Toc12606325)

[Exercise 8: Perform a code review 23](#_Toc12606326)

[Task 1: Locate pending pull requests for your repo 23](#_Toc12606327)

[Task 2: Review code and comment when needed 23](#_Toc12606328)

[Task 3: Complete the pull request 24](#_Toc12606329)

[Exercise 9: After the pull request is completed 26](#_Toc12606330)

[Task 1: Refresh DevOps Server and delete local feature branch 26](#_Toc12606331)

# Overview

In this lab, you will learn how to use git to manage your local source code management and how to promote your changes to DevOps Server. You will learn how to work with git in Visual Studio 2017 and at the command line. We will also look at ways to troubleshoot and fix some common problems.

## Prerequisites

In order to participate in this workshop you will need a computer running Windows 10 and Visual Studio 2019. You will also need the latest version of git installed. The gitWorkshop repo must exist in the DevOps Server development environment.  
  
For Visual Studio, contact your manager.

For git, install from <https://git-scm.com/download/win>. Current version on Windows as of this document version is **2.21**, but be sure to get the latest maintained version. For the most part, select default settings unless otherwise indicated below.

If you wish, select a new default editor for git. Also, be sure to select the “Use the native Windows Secure Channel library” option during installation. This will allow you to work over the VPN over HTTPS.

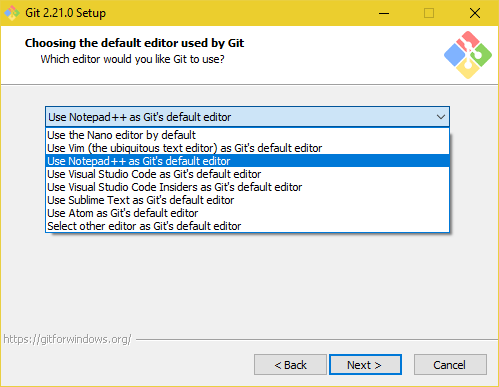


Figure 1- Set git default editor

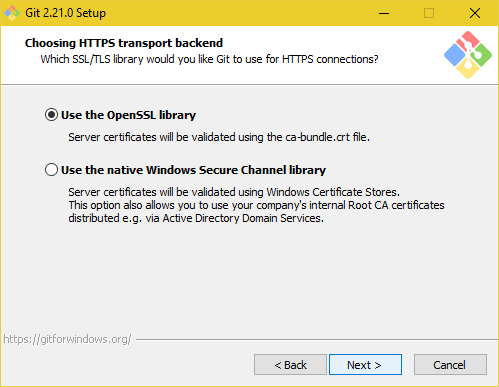


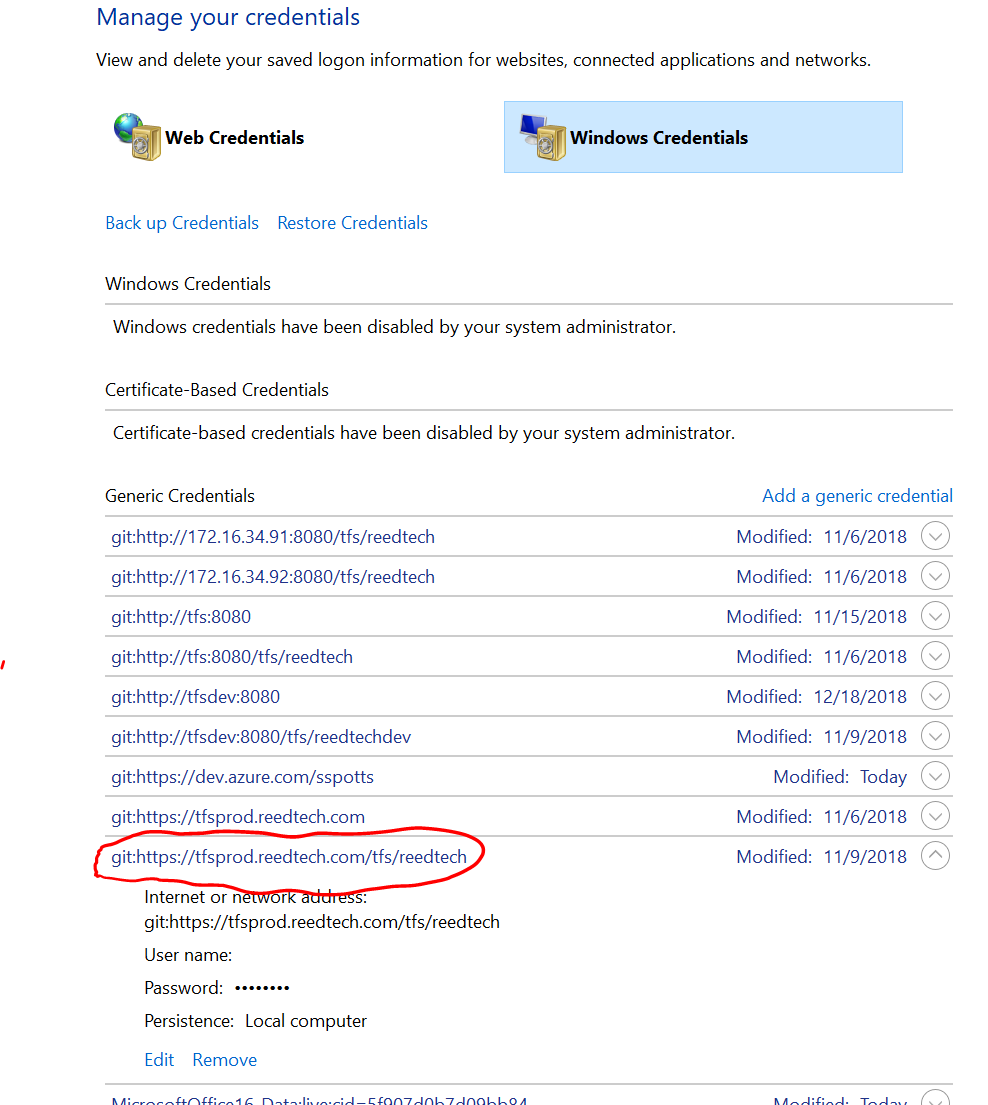
Figure 2- git Setup for Windows Secure Channel Library Connections

## About the Developer Scenario

The premise for this workshop starts with creating a branch from a workshop specific repository (repo) on DevOps Server. You will be creating a “feature” branch that will be what you will be making changes in, learning how to work with your feature branch, and how to push it to the remote DevOps Server for data retention and sharing. You will also learn how to have your feature branch merged with the develop (shared working) branch, and how to perform code reviews and complete pull requests.

## Authentication

If at any time you get an authentication error, such as:   
  
fatal: unable to access 'https://tfsprod.reedtech.com/tfs/ReedTech/ReedTech/\_git/gitWorkshop/': OpenSSL SSL\_connect: SSL\_ERROR\_SYSCALL in connection to tfsprod.reedtech.com:443  
  
you may need to modify your credentials in Windows Credentials Manager. The easiest way to open this in Windows 10 is to click on the search icon on the Task Bar and type “cred” and then click on Credential Manager from the returned list.  
  
Click on Windows Credentials, then expand the setting for git:https://tfsprod.reedtech.com and click Edit. Sometimes the problem is the communication library when working over the VPN, so you may need to change to update your sslbackend setting in the git system config as shown later in Exercise 1.

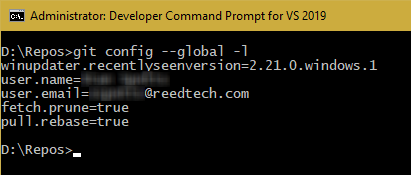


If the User name is not set, set it to your network username (without the domain). Set the password to your network password and save the entry. Then try to clone again.  
  
If this does not fix the issue, your configuration may be set to OpenSSL. The load balancer seems to have an issue with this configuration, so use the Windows secure channel protocol instead. Follow the instructions in Exercise 1, Task 3 to do this.

# Exercise 1: Configure git

You will learn how to set your user information into git, which git uses for the commit history, and additional items needed for VPN access to DevOps Server/git.

## Task 1: Set global git configuration.

1. Open a command prompt using Run as Administrator.
2. Type:  
   git config --global user.name “Firstname Lastname”
3. Type:  
   git config --global user.email “youremail@reedtech.com”
4. Verify your configuration settings. Type:  
   git config --global -l  
     
   You should see something similar to this:  
   
5. Set default pull to local branch from remote to do a rebase. This is so that when you are working on a feature and other developers have been making changes to the remote develop branch, when you pull to update your local develop branch it will fast forward the remotes changes to your local branch then apply your commits. Otherwise, you will see mixed commits and creating a pull request will be harder.  
   git config --global pull.rebase “true”
6. Set fetches from remote to prune local remote branches. Otherwise you’ll need remember to use command line to clean up your remotes/origin branches when feature branches are deleted after pull requests are completed.   
   git config --global fetch.prune “true”

**Note:** You may set these global configuration settings from Visual Studio. Open Team Explorer, click **Settings**, and then click **Global Settings** under Git. However, not all global settings are available in the UI, nor are any system or local repo settings. These settings require the command line.

1. Since we are running on Windows, tell git to ignore case in file system. Git, by its origins in Linux, supports case sensitive filenames. This can cause issues if you rename folders or files by just changing case.

git config --global core.ignorecase “true”

Task 2: Change editor used for interactive commit messages, squash, etc.

1. By default, git uses VIM as the text editor for things like interactive commit editing, squash merge editing, and other editing needs. If you do not wish to use VIM, you may change it.  
     
   Change to use 64-bit Notepad++, type:  
   git config --system core.editor “’c:\program files\Notepad++\notepad++.exe’ –multiInst –noPlugins –nosession -notabbar”

For 32-bit Notepad++, type:

git config --system core.editor “’c:\program files (x86)\Notepad++\notepad++.exe’ –multiInst –noPlugins –nosession -notabbar”

These include some useful parameters to simplify the editing experience. Feel free to include the ones you prefer.

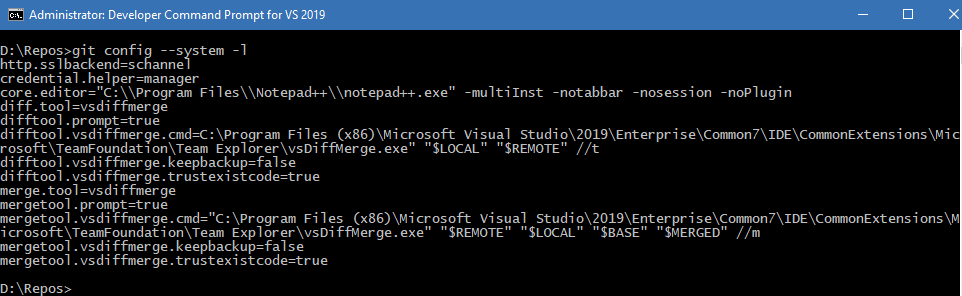
|  |  |
| --- | --- |
| -multiInst | Notepad++ is single-instance by default. This option allows you to launch several instances of Notepad++. |
| -noPlugins | Use this parameter to launch Notepad++ without loading the plugins. This parameter is useful when you want to determine where bad behavior or crash is coming from (from the plugins or from Notepad++ itself). |
| -nosession | Use this parameter to launch Notepad++ without loading the previous session (the files opened in Notepad++ at the last time). On shutdown, Notepad++ will not record currently opened files in session.xml. |
| -notabbar | This parameter allows users to turn off tab interface. It may be useful if users want Notepad++ behavior to be similar to MS notepad. |

For Windows notepad, type:  
git config --system core.editor “notepad”

## Task 3: Ensure set up for Windows Secure Channel Library connection to DevOps Server

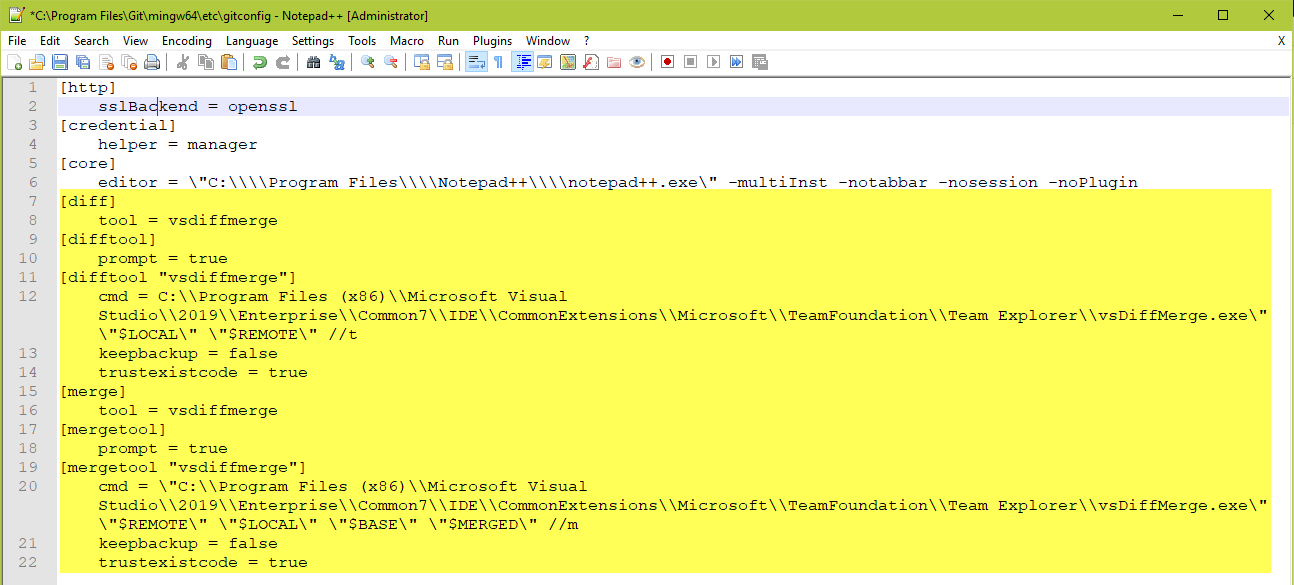
1. Now we will verify that the git system configuration is set to use WSCL connections to DevOps Server.  
   Type:  
   git config --system -l

You should see something similar to this:



1. If the entries for http.sslbackend does not exist or is set to “openssl”, we need to fix it. Type:  
   git config --system http.sslbackend schannel

## Task 4: Visually edit global and system configuration

1. git has built-in support for editing configuration *en masse* using your defined editor. We will see how this looks.  
   Type:  
   git config --system –e
2. Ensure the difftool and mergetool settings are as specified below, and change if not.  
   
3. If you made changes, simply Save and exit to update the system settings.
4. Look at the contents of the system configuration file. It is stored in c:\Program Files\git\mingw64\etc\gitconfig. These configuration entries affect all workstation users and all repos.
5. Do this for the global config next.

Type:  
git config --global –e

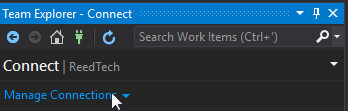
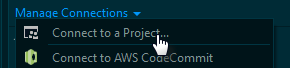
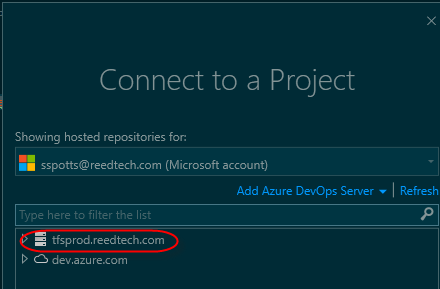
1. Look at the contents of the global configuration file. This is stored in c:\Users\{youraccountname}\.gitconfig. These configuration entries affect all repos, just for you.

**Note:** If you change to a root folder of a git repo and type **git config -e**, you will open the repos’ config file in the repo’s **.git\config** file. This affects only the particular repo, for all users.

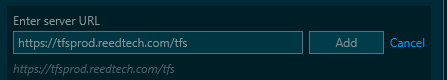
# Exercise 2: Configure Visual Studio to access remote git repo

We will manage a connection in Visual Studio that allows us to access our repo.

## Task 1: Manage Connections in Visual Studio

1. Open Visual Studio 2019 as an Administrator.
2. Go to Team Explorer and click on the Manage Connections icon (C:\Users\sspotts\Pictures\Saved Pictures\ManageConnectionsIcon.png).  
   
3. Click on Connect to a Project.  
   
4. Check if DevOps production server is in your list of servers.  
   
5. If it is not, click Add Azure DevOps Server.

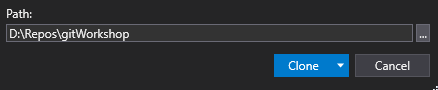


1. Then type **https:/tfsprod.reedtech.com/tfs** and click *Add*.  
   

# Exercise 3: Set up to do work on your local machine

Once we have a connection to DevOps Server, we will clone the repository used for this workshop. Clone makes a local copy of the repository, including commit history, and sets up a link between the local repo and the remote repo. Having the commit history means that you can see all changes, from all developers, even when disconnected from the network.

## Task 1: Clone the gitWorkshop repo

1. Expand the entry for the DevOps Server production environment, then expand **ReedTech**, and then expand the next level **ReedTech**
2. Click the **gitWorkshop** repository.
3. Set the Path to **c:\repos\gitWorkshop**, or **d:\repos\gitWorkshop** if you have a D: drive. You must click the ellipses to the right of the field where you enter the path, browse to the drive, and make new folder to create the folder structure.  
   
4. Click the **Clone** button. You should see an indication that you successfully cloned the repository.  
   

# Exercise 4: Get ready to work

For all development, we work with feature, bug, or hotfix branches. Branching is different than it was with TFVC; it is a frequent activity when using git. Before working on a sprint user story or bug, you will create a new branch from the develop branch, just for the scope of work for a story or bug.

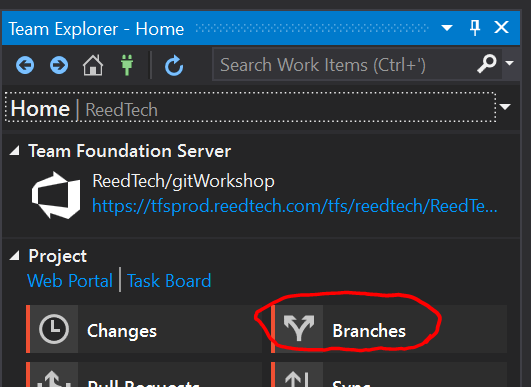
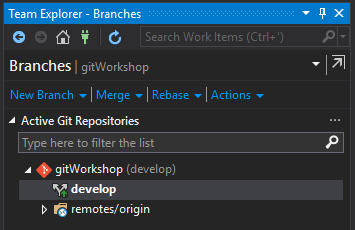
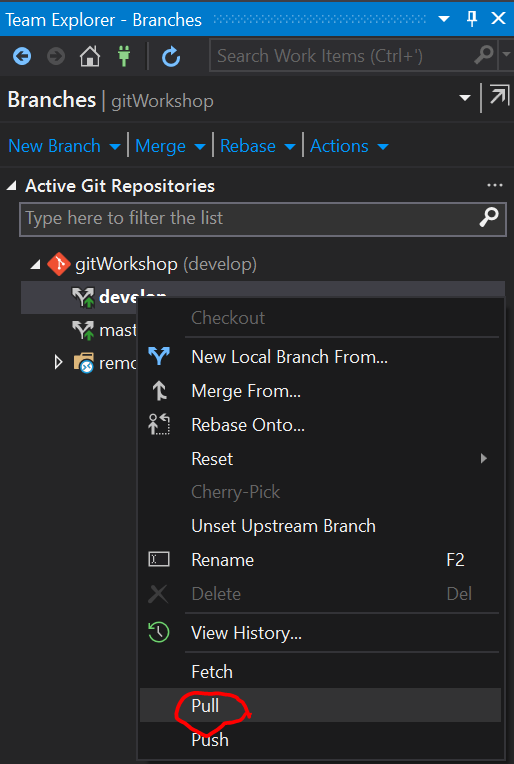
You will use the prefix “feature/” or “bug/” in the name, depending on what kind of work it will be. This will make it easier to organize your work. You will include the work item number and title of the story, and even your name, in the branch name. Again, this is an organizational need. It will also help at times when you have other people working with you on the same story.

When you cloned the gitWorkshop repo, the develop branch was downloaded and the solution in the root of the repo was opened in Solution Explorer.

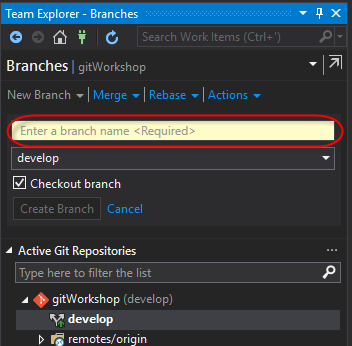
For the purposes of this demo, replace {yourusername} with your actual network user name (without domain).

**Note:** If you do not see the buttons or menu options specified in the exercises, you have connected only to your local repo and must reconnect to your remote repo. Click the Manage Connections icon (C:\Users\sspotts\Pictures\Saved Pictures\ManageConnectionsIcon.png) and open the dropdown menu for Manage Connections. Click **Connect to Team Project**. This should show more options under **Project**. Under the **Solutions** section, click on the solution file to open the solution if it is not open.

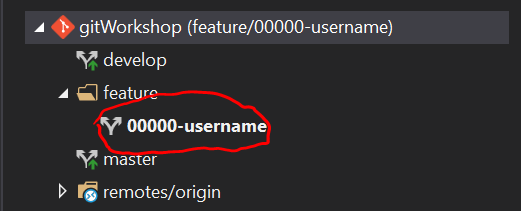
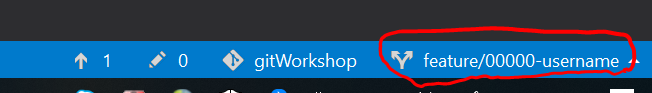
## Task 1: Create a local feature branch in Visual Studio

1. In Team Explorer, click on the Branches button.  
   
2. Double-Click on develop to select it.  
     
   Command line: **git checkout develop**
3. A good practice is to always pull from develop before you create a new branch. This ensures that you are branching from the latest code committed to the server and not starting from outdated code.   
     
   Right click on the develop branch, and click Pull from the context menu.  
   

Command-line: **git pull origin develop**

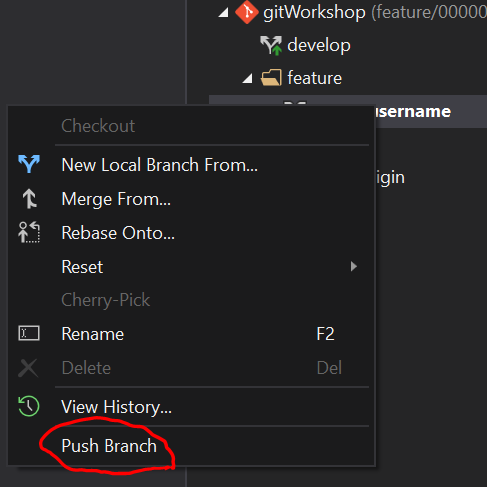
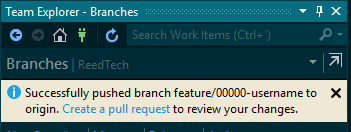
1. Click New Local Branch From… (Make sure *Checkout branch* is checked).  
   
2. Type **feature/00000-{*yourusername}*** as the branch name. For the purposes of this workshop, we will use {*yourusername}* instead of the title of a story. Make sure the **Checkout branch** box is checked.

**Note:** You may not use spaces in branch names. Use hyphen or underscores instead.

1. Click the **Create Branch** button.  
   Command line (3-5): **git checkout –b feature/00000-{yourusername}**
2. You will see the new branch, and it will be the current branch. You can tell by looking inside the parentheses after the repo name, the bold font on the branch name, and the bottom right of the footer in Visual Studio.  
     
     
   

## Task 2: Push your feature branch to DevOps Server

You will push your feature branch regularly, at least daily, to guard against loss in case your local drive fails.

1. Right click on the feature, and from the context menu click Push Branch.  
     
     
   Command line: **git push –u origin feature/00000-{yourusername}**
2. You will get a message saying that you pushed the branch to origin (the remote git on DevOps Server).  
   

# Exercise 5: Make some changes to your feature

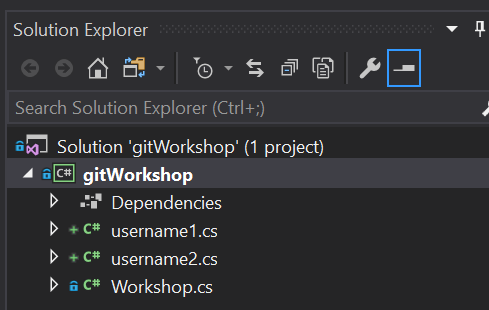
So far, we have used Visual Studio to connect to DevOps Server repos and to create branches. Now we need to work on changes to implement the requirements for the sprint user story. We will change a few files, make local commits, and rebase against the develop branch to synchronize changes there.

## Task 1: Add a c# class file.

1. Switch to Solution Explorer view in Visual Studio.
2. Right click on the gitWorkshop project and click Add from the context menu.
3. Click Add New Item…
4. Create a new Visual C# Class file called {yourusername}1.cs.

## Task 2: Add another file.

1. Switch to Solution Explorer view in Visual Studio.
2. Right click on the gitWorkshop project and click Add from the context menu.
3. Click Add New Item…
4. Create a new Visual C# Class file called {yourusername}2.cs.

Your solution will look similar to this:  


## Task 3: Modify a file that other developers may be modifying

1. In Solution Explorer, open the Workshop.cs file for editing.
2. In the Workshop class, add the following method.

public string HelloYourusername()

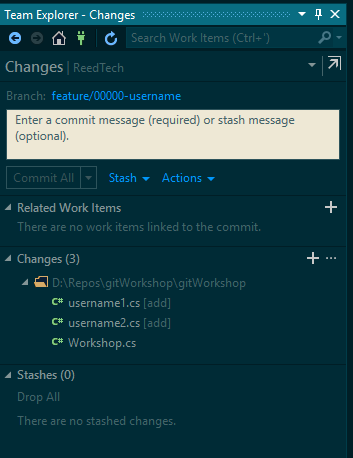
{

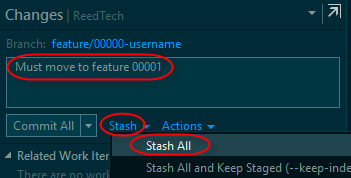
return "Hello, {yourusername}";

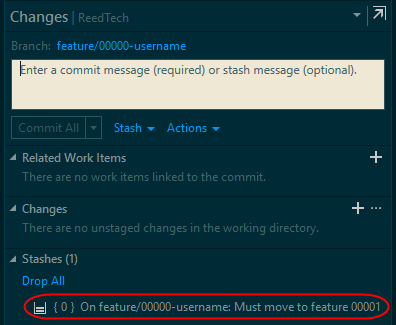
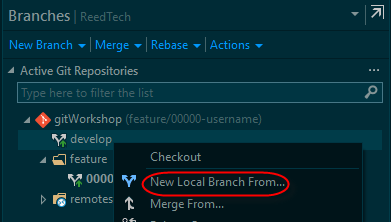
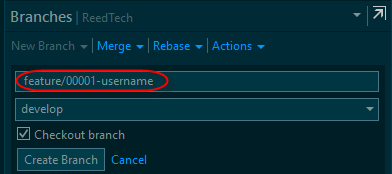
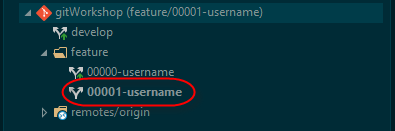
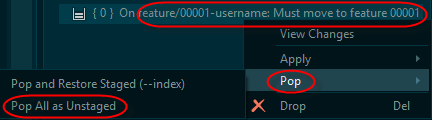
}

1. Save the file.

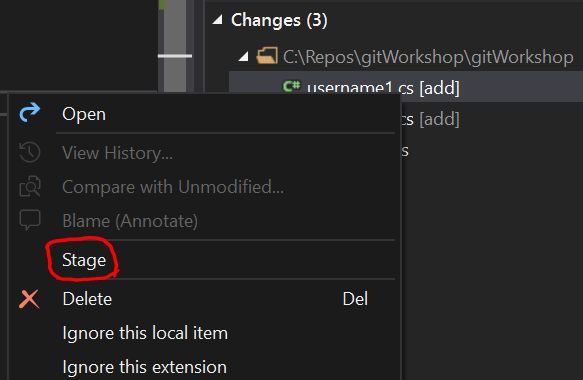
## Task 4: Use git stash to create new repo and move pending changes there.

1. Switch to Team Explorer view in Visual Studio.
2. Click on the Home icon.  
   
3. Click on **Changes**.
4. See the files that were changed. These are not yet in the index for the next commit.  
   
5. Enter a stash message.
6. Click Stash then Stash All.



1. See the new stash in the list of stashes.  
   
2. Click on the Home icon.  
   
3. Click on **Branches**.
4. Right click on the local Develop branch and click on *New Local Branch From…* (make sure *Checkout branch* is checked).  
   
5. Type **feature/00000-{*yourusername}*** as the branch name.  
   
6. Click the **Create Branch** button.  
   Command line (3-5): **git checkout –b feature/00001-{yourusername}**
7. You will see the new branch, and it will be the current branch. You can tell by looking inside the parentheses after the repo name, the bold font on the branch name, and the bottom right of the footer in Visual Studio.  
     
   
8. Click on the Home icon.  
   
9. Click on **Changes**.
10. Right click on the stash you just created and click *Pop*, then *Pop All as Unstaged*. This will apply the changes in the stack and remove them from the stash list.  
    

## Task 5: Add the changed files to the index of files to be committed.

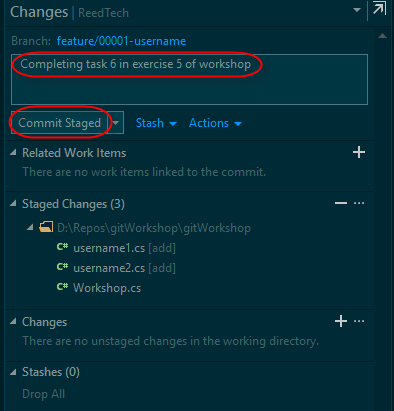
1. See the changed files. These are not yet in the index for the next commit.
2. Click on the “+” to the right of **Changes (3)**. This stages all the changed files for the next commit. You may instead stage only specific changed files by right clicking on them and clicking Stage from the context menu. However, since we want to stage all of the changes, click the “+” icon.  
   

Command Line:  
All (new, modified, deleted) files: **git add -A** or **git add –all** or **git add .**

Modified and new (not deleted): **git add –ignore-removal**  
Modified and deleted (not new): **git add -u** or **git add –update**

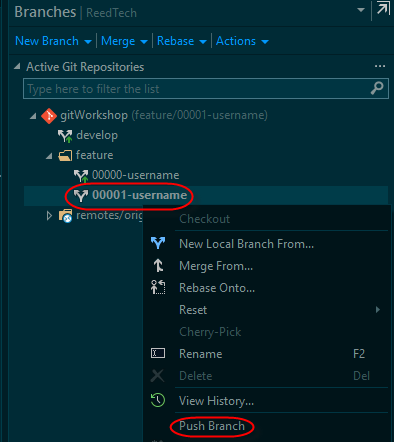
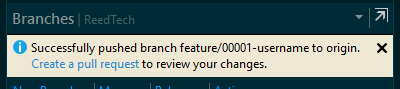
Specific files: **git add {yourusername}1 {yourusername}2**

## Task 6: Commit your changes.

1. Notice that now we have three staged changes and no unstaged changes.  
   Command Line: **git status**
2. Enter a descriptive commit message.
3. Click Commit Staged.  
     
   Command Line: git commit –m”Completing task 5 in exercise 5 of workshop”
4. You will see a message telling you that you created a commit.  
   

## Task 7: Push your feature branch to DevOps Server

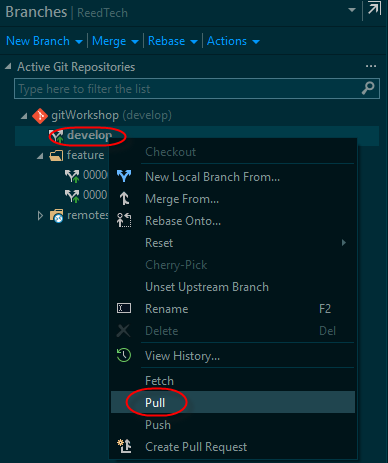
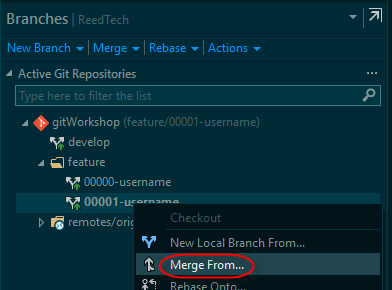
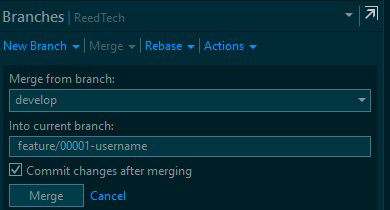
You will push your feature branch regularly, at least daily, to guard against loss in case your local drive fails5

1. Click the Home icon in Team Explorer.
2. Click **Branches**.
3. Right click on the feature, and from the context menu click **Push Branch**.  
     
     
   Command line: **git push –u origin feature/00001-{yourusername}**
4. You will get a message saying that you pushed the branch to origin.  
   

# Exercise 6: Resolve merge conflicts in your feature

Other developers may have made changes and merged them, via pull requests, into the develop branch. Periodically, you should merge changes into your feature branch and resolve any conflicts. Not only will this keep your feature branch up to date, it will make it easier to complete your pull request.

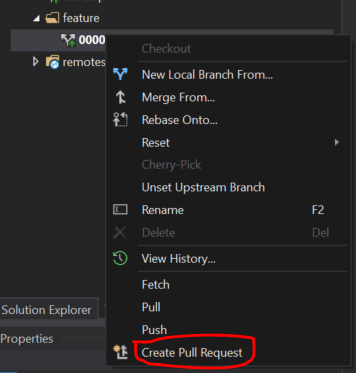
## Task 1: Check out the develop branch.

1. Open the Team Explorer window.
2. Click **Branches**.
3. Double click the local **develop** branch to check it out.  
   Command Line: **git checkout develop**
4. Right click on **develop** and click **Pull** from the context menu.  
     
   Command Line: **git pull --rebase**
5. Double click the feature branch to check it out.  
   Command Line: **git checkout feature/00001-{yourusername**}
6. Right click the feature branch and click **Merge from…** from the context menu.  
   
7. You will merge from your local **develop** branch into your current local feature branch.  
   
8. If you have merge conflicts, fix them and repeat from step 6.
9. When you have no merge conflicts, you may commit the changes (if you did not have the “Commit changes after merging” box checked.
10. It is good practice to push the feature branch to DevOps Server to ensure it is up to date after the merge from develop. Right click on the feature branch and click **Push** from the context menu now.
11. If you like, look at the Workshop.cs file to see changes that others may have made.

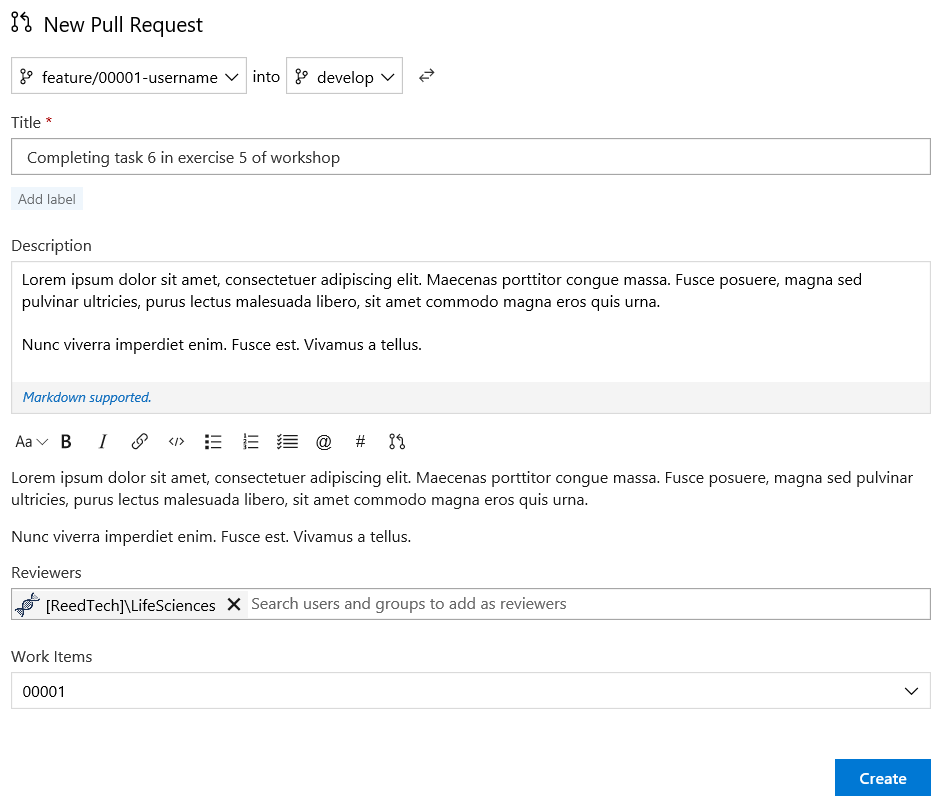
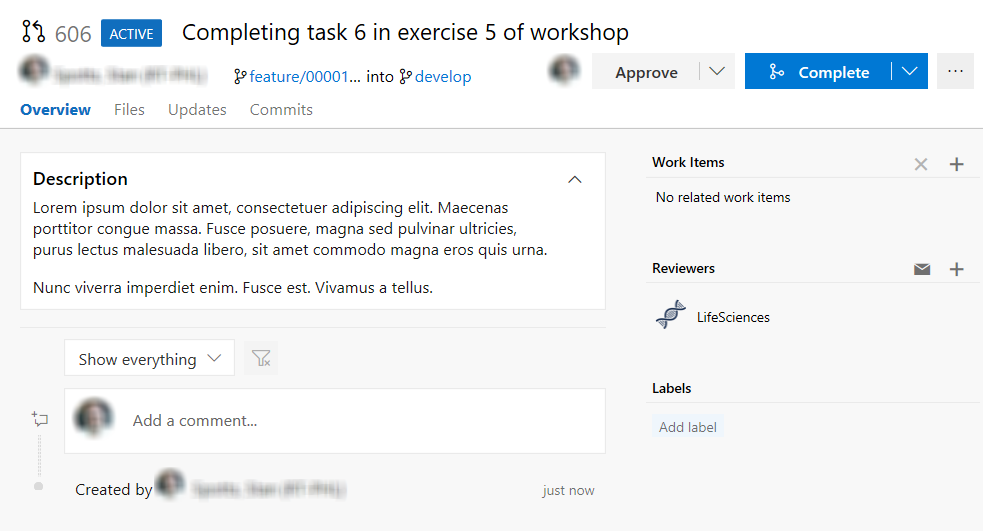
# Exercise 7: Finish your feature

When you have completed your coding, unit testing, refactoring, and anything else needed to consider the feature complete, you will push the feature branch to the server and create a pull request to the develop branch. This makes a request for review and for the code to be merged into the develop branch. You will not process the request yourself, someone else will perform a code review and either ask you to address some issues and try again, or will complete the request and merge the code.

## Task 1: Create Pull request

1. Open the Team Explorer window.
2. Click the Branches button.
3. Right click on your feature and click Create Pull Request from the context menu.  
   
4. A browser window requiring the information for the completion of the pull request will open.

## Task 2: Complete the Pull Request setup

1. Give the pull request a title. The default will be the last commit request comment, which you must replace.
2. Add a description of the changes you have completed for this feature. The default will be the last request comment, which you must replace.
3. You may optionally modify the code reviewers. Set the reviewer to yourself.
4. Add the work items to associate with this pull request. Always do this, as branch policies require it.
5. Scroll down the page and review the files changed and the commits made to ensure this is what you want to be merged into the develop branch.
6. Click the Create button.  
   
7. The browser shows the pull request, ready for completion. The review may approve it, approve it but include suggestions, or reject it. The reviewer may also set it to wait for the author to make changes before approving.   
   

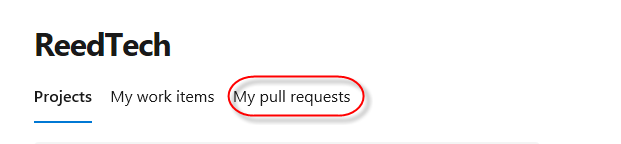
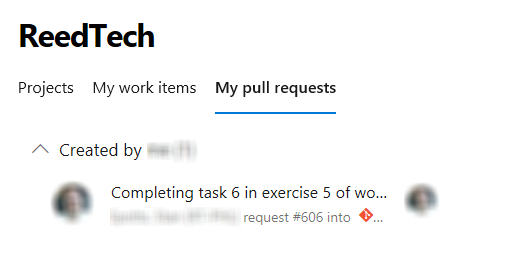
**Note:** If you had to make changes, or forgot to add something, just push to the feature branch again and the pull request will update with the changes. However, if you have made major changes and the comments are inaccurate, abandon the pull request and create a new one for the feature branch.

# Exercise 8: Perform a code review

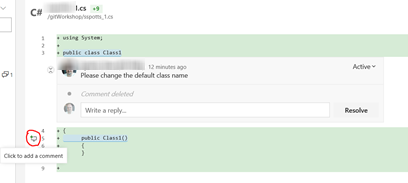
Developers perform code reviews for other developers during the development cycle. This is an important task, and after an approval, DevOps Server merges the code merged into the development branch. The policy for the develop branch is set to do a “squash” merge, which takes all the commits from the pull request and creates just one to apply to the develop branch. This allows developers to make as many small commits as they wish in their local feature branches and not worry that they will add non-critical comments to the commit log on origin.

Before beginning the first task, ensure you have an assigned developer whose pull requests you will process.

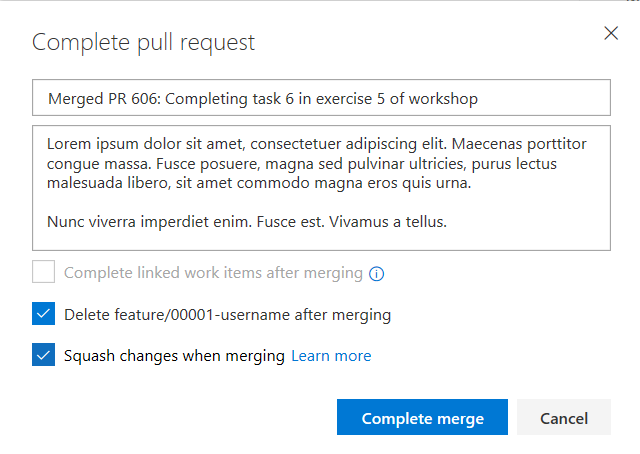
## Task 1: Locate pending pull requests for your repo

1. Open your browser and navigate to the dev DevOps Server site at <https://tfsprod.reedtech.com/tfs/ReedTech>.
2. Click on *My pull requests*. This is where you will see all pull requests created by you, and those that you are set as a reviewer.  
   
3. Click on the title of the pull request that is assigned to you, by you, for the workshop.  
   

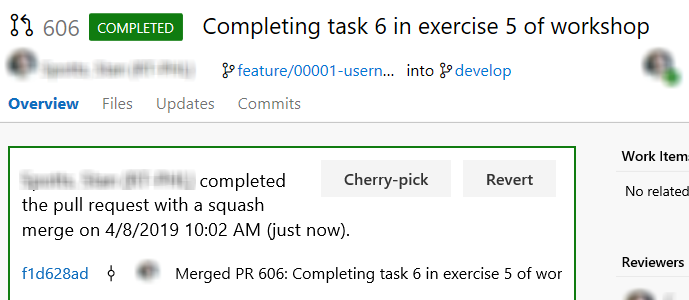
## Task 2: Review code and comment when needed

1. Click Files.  
   You will see the folder structure and the files included in this pull request.
2. Review the code. Options display to toggle between inline and side-by-side differences, and to display a less cluttered “full screen” view of the files.
3. When you move your mouse to the left of the line number of any code line, you will see an icon/prompt to add a comment. Click it to open a dialog box to add comments. You can also past text and even drop file on it. The UI keeps comments, and replies to them, in line with the code.  
   
4. Any new comment is Active. It can be changed to Pending, Resolved, Won’t Fix, or Closed by the developer if the review is not completed. This allows the developer to discuss the changes within the comments (for a history), mark resolved when fixed, or otherwise process the review. The policy on the develop branch requires that all issues be resolved before completing the pull request.
5. Experiment with the review page. You may add comments, or resolve issues, etc. with other workshop participants.

## Task 3: Complete the pull request

1. When you are satisfied with the pull request, click **Approve** and then **Complete**. You will see a request to complete the pull request.
2. Review the title and the description of the pull request for accuracy, and edit if necessary. Ensure that the boxes for **Delete** and **Squash** are checked.  
   

**Note:** Development management and teams need to decide whether to set the “Complete linked work items after merging” box. This will mark all work items associated with the pull request to complete rather than requiring manual intervention.

1. Click *Complete Merge*. The merge request will be marked as **Completed**. The feature branch will no longer exist in DevOps Server.  
   

# Exercise 9: After the pull request is completed

## Task 1: Refresh DevOps Server and delete local feature branch

1. Open Visual Studio.
2. Open Team Explorer.
3. Click the Home icon.
4. Click Branches.
5. Double click the **develop** branch to check it out.
6. Right click the develop branch and click **Pull** from the context menu.  
   This will update your local develop repo to match the remote, and will remove the remote feature branch from the list in Visual Studio (because we set the global config **fetch.prune** to true).
7. Right click the local feature branch.
8. Click **Delete** from the context menu.  
   