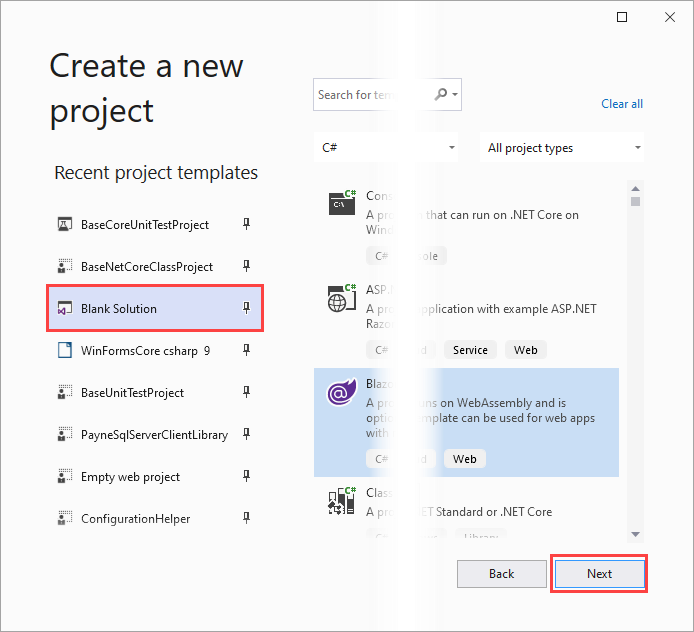
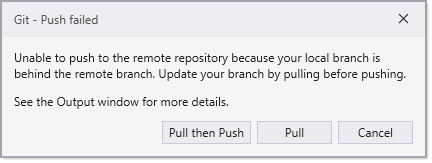
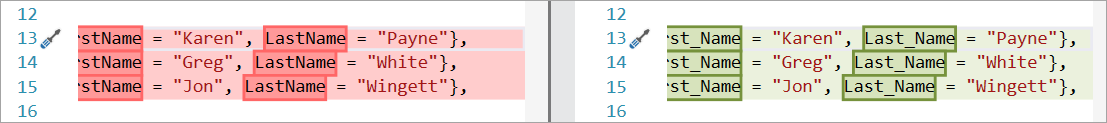
1. Create a new Visual Studio solution using the blank template option under C:\OED\Dotnetland\VS2019\LearnGit. Name of solution should not contain spaces in the name and should begin with an upper cased character e.g. LearnGit.  
     
   
2. Create the following solution folders
   1. Class projects
      1. Add a new C# .NET Core class project
   2. Unit test projects
      1. Add a new C# .NET Core test project
3. Under Class projects, in the class project created above
   1. Add a new folder named Interfaces
      1. Add a new Interface named IBase
      2. Add the code below after the namespace, between the Bob Hope brackets
   2. Add a new folder named Classes
      1. Add a new class named Person
      2. Add the code below after the namespace, between the Bob Hope brackets
         1. You will need to add a using statement to the Interface namespace
4. Login to your personal GitHub account
   1. Create a new repository, name it learn-git
   2. Copy the URL from your browser’s address bar into notepad
5. In Visual Studio, from the menu select Git then Create Git Repository
   1. Click Existing repository
   2. Make sure Local path is under C:\OED\Dotnetland\VS2019\LearnGit
   3. From notepad, paste in the URL into Remote URL
   4. Click Create and Push
   5. Under the GitHub page
      1. Next to About, click the gear, enter a short description
      2. Save changes
      3. There is a button Add readme, click it
      4. Add some text
      5. Click commit new file
6. In Visual Studio
   1. Open IBase
   2. Add /// above Id property, add this is the primary key
   3. Save the file
   4. Click on Git Changes window
   5. Enter a comment
   6. Click Commit all and push
   7. The following appears (because of the readme file created online)  
      
   8. Click Pull then Push
   9. Open Solution Explorer, note that even though the readme file was added it’s not seen.
   10. In Solution Explorer, right click, Add, Existing item
   11. Add readme.md, Visual Studio places it under a virtual folder named Solution items.
7. Open another Visual Studio
   1. Click on Git menu, Clone repository
   2. Enter the URL from notepad into Repository Location
   3. Enter a path with the original path above with Clone on the end e.g.   
      C:\OED\Dotnetland\VS2019\LearnGitClone
   4. Click Clone
   5. Under the file menu click Open, Project/Solution
   6. Enter C:\OED\Dotnetland\VS2019\LearnGitClone
   7. Click on LearnGit.sln, click Open
8. Go back to the first opened Visual Studio.
   1. Append text to the summary for Id property of IBase
   2. In Git Changes, add comment, click Commit all and Push
9. Go back to the second opened Visual Studio
   1. Click Fetch in Git Changes
   2. Click Git, view history, there is one incoming change
   3. Right click on the line, view commit details. Since this is just a comment
   4. Go back to the History window and click pull
   5. Open IBase and note the pull was successful
10. Has anyone noticed two readme.md files under solution items? We will get to that later
11. Go back to the first Visual Studio
    1. Under Classes folder add a new class Worker (see Worker version 1)
    2. Go to Git Changes, enter a commend
    3. Select command all and push
12. Go back to the second Visual Studio
    1. Perform a fetch
    2. Select Git menu, view history
    3. If incoming shows 0, click the refresh button
    4. Right click, view history, one file added
    5. Perform a pull
    6. See the new class in Solution Explorer
    7. Under Classes folder add a new class, Mocked (see Mocked 1)
    8. Under Classes folder add a new class Worker (see Worker version 2)
    9. Commit and push (as done several times above)
    10. Perform a fetch (as done above several times)
    11. View history, we have a new file and one which will become a merge? Or a discussion on which one to pick.
    12. Let’s do a commit all and get  
        
    13. All is good because there were no conflicts
13. Go back to the first Visual Studio
    1. In Person class, change FirstName to First\_Name, LastName to Last\_Name (this also changed code in the Mocked class)
    2. Commit and push
14. Go back to the second Visual Studio
    1. Perform a fetch
    2. View history  
         
       
    3. Do a pull we get  
         
       
    4. Code is updated to reflect the merge in regards to properties in (b)
    5. Take the underscored out of first and last names
    6. Commit all and push
    7. Do pull in first Visual Studio

*Git is good at automatically merging file changes in most circumstances, as long as the file contents don't change dramatically between commits*.

Okay, let’s get into a **merge conflict**.

In either or project, add a string property PostalCode to Person than in the Mocked class set the properties. Commit all and push

In the other project add an int property PostalCode to Person than in the Mocked class set the properties. Do and commit all and push. We get a dialog, select pull.

We get (just remember, seeing <<<<HEAD means to use the merge editor)

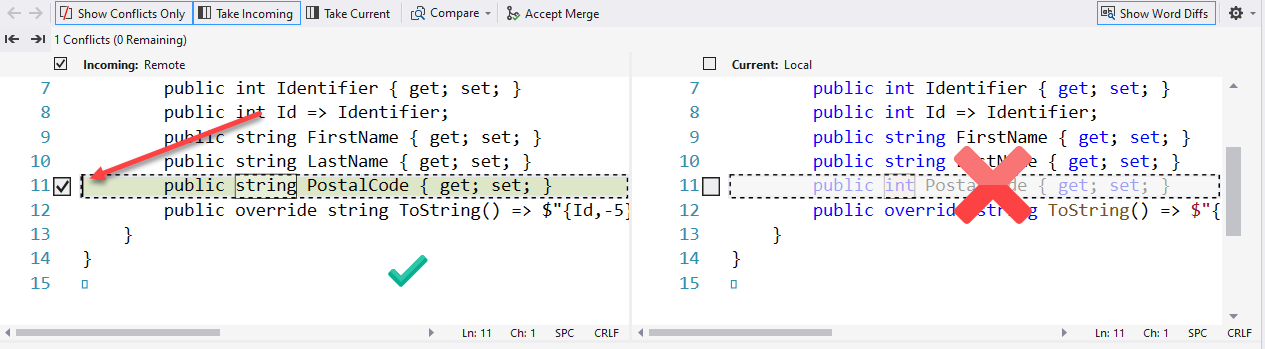


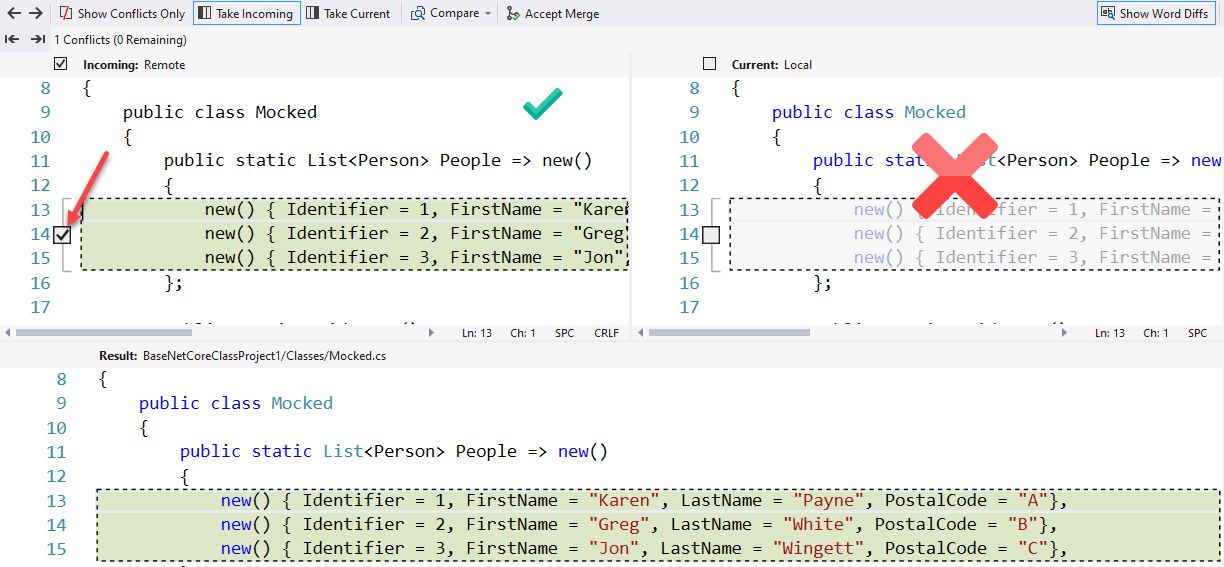
Open merge editor



Okay, who would have postal code as an int so let’s go with string.

We must perform a merge on both Mocked and Person classes else we can’t compile because of type differences in the property PostalCode





https://github.com/karenpayneoregon/learn-git

**IBase interface**

public interface IBase

{

    public int Id { get; }

}

**Person class**

Replace BaseNetCoreClassProject1 with your project namespace

using BaseNetCoreClassProject1.Interfaces;

namespace BaseNetCoreClassProject1.Classes

{

    public class Person : IBase

    {

        public int Identifier { get; set; }

        public int Id => Identifier;

        public string FirstName { get; set; }

        public string LastName { get; set; }

        public override string ToString() => $"{Id,-5}{FirstName} {LastName}";

    }

}

**Worker version 1**

Replace BaseNetCoreClassProject1 with your project namespace

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using BaseNetCoreClassProject1.Interfaces;

namespace BaseNetCoreClassProject1.Classes

{

    public class Worker

    {

        public static void CompareValue<T>(List<T> sender) where T : class

        {

            foreach (var item in sender)

            {

                if (item is IBase data)

                {

                    Debug.WriteLine(data.Id);

                }

            }

        }

    }

}

**Worker version 2**

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using BaseNetCoreClassProject1.Interfaces;

namespace BaseNetCoreClassProject1.Classes

{

    public class Worker

    {

        public static void CompareValue<T>(List<T> sender) where T : class, IBase

        {

            foreach (var item in sender)

            {

                Debug.WriteLine(item.Id);

            }

        }

    }

}

**Mocked class 1**

Replace BaseNetCoreClassProject1 with your project namespace

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace BaseNetCoreClassProject1.Classes

{

    public class Mocked

    {

        public static List<Person> People => new()

        {

            new() { Identifier = 1, FirstName = "Karen", LastName = "Payne"},

            new() { Identifier = 2, FirstName = "Greg", LastName = "White"},

            new() { Identifier = 3, FirstName = "Jon", LastName = "Wingett"},

        };

        public static void Demo()

        {

            IEnumerable<Person> query = from p in People select p;

            Worker.CompareValue(People);

        }

    }

}

## Definitions

**Bob Hope brackets**

{ }

**Git Fetch**

The fetch command downloads commits, files, and refs from a remote repository into your local repo. Fetching is what you do when you want to see what everybody else has been working on. It’s similar to svn update in that it lets you see how the central history has progressed, but it doesn’t force you to actually merge the changes into your repository. Git isolates fetched content from existing local content; it has absolutely no effect on your local development work. Fetched content has to be explicitly checked out using the git checkout command. This makes fetching a safe way to review commits before integrating them with your local repository.

*It's important to fetch and pull before you push. Fetching checks if there are any remote commits that you should incorporate into your local changes. If you see any, pull first to prevent any upstream merge conflicts*.

