

WorkshopPLUS - Essentials on Azure DevOps Services and GitHub

Lab Guides

Conditions and Terms of Use

Microsoft Confidential

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

© 2017 Microsoft Corporation. All rights reserved.

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/en-us/legal/intellectualproperty/Permissions/default.aspx>

DirectX, Hyper-V, Internet Explorer, Microsoft, Outlook, OneDrive, SQL Server, Windows, Microsoft Azure, Windows PowerShell, Windows Server, Windows Vista, and Zune 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.

Module 3: Azure Repos - Git

Lab 1: Getting Started with Git by Using Azure DevOps Services

Introduction

A Git repository, or repo, is a folder that you've told Git to help you track file changes in. You can have any number of repos on your computer, each stored in their own folder. Each Git repo on your system is independent, so changes saved in one Git repo don't affect the contents of another.

A Git repo contains every version of every file saved in the repo. Git saves these files very efficiently, so having a large number of versions doesn't mean that it uses a lot of disk space. Storing each version of your files helps Git merge code better and makes working with multiple versions of your code quick and easy.

You can create Git repos in team projects to manage your project's source code. Each Git repo has its own set of permissions and branches to isolate itself from other work in your project. A fork is a complete copy of a repository, including all files, commits, and (optionally) branches. Forks are a great way to support an Inner Source workflow: you can create a fork to suggest changes to a project when you don't have permissions to write to the original project directly. Once you're ready to share those changes, it's easy to contribute them back using pull requests.

[Exercise 1: Configuring Azure Repos Lab Environment](#)

[Exercise 2: Cloning Existing Azure Repository](#)

[Exercise 3: Saving Work Locally with Commits](#)

[Exercise 4: Reviewing History in Azure Repos](#)

[Exercise 5: Managing Branches Locally from Visual Studio Code](#)

[Exercise 6: Managing Branches from Azure DevOps Services](#)

[Exercise 7: Working with Pull Requests in Azure Repos](#)

[Exercise 8: Managing Repositories in Azure Repos](#)

Objectives

In this lab, you will perform Git operations such as:

- Clone a repository
- Stage, commit, and sync changes
- Review history
- Manage branches using Visual Studio Code and Azure DevOps
- Create and manage pull requests
- Create, manage, and delete repositories

Prerequisites

- This lab requires you to complete [Exercise 1](#) of the previous **PartsUnlimited Lab Setup** lab.
- [Visual Studio Code](#) with the C# extension installed.
- [Git for Windows 2.21.0](#) or later.

VS Code with C# and Git for Windows is already installed on your VM.

Estimated Time to Complete This Lab

60 minutes

Module 3: Azure Repos with Git, Lab 1: Getting Started with Git, Exercise 1: Configuring Azure Repos Lab Environment

Exercise 1: Configuring Azure Repos Lab Environment

Objectives

Git is a distributed version control system. Git repositories can reside locally, such as on a developer's machine and can be hosted by Azure DevOps Services. You will learn how to establish a local Git repository, which can easily be synchronized with a centralized Git repository on the Azure DevOps Services.

Prerequisites

None

Tasks

1. [Task 1: Configuring Visual Studio Code](#)

Module 3: Azure Repos with Git, Lab 1: Getting Started with Git, Exercise 1: Configuring Azure Repos Lab Environment

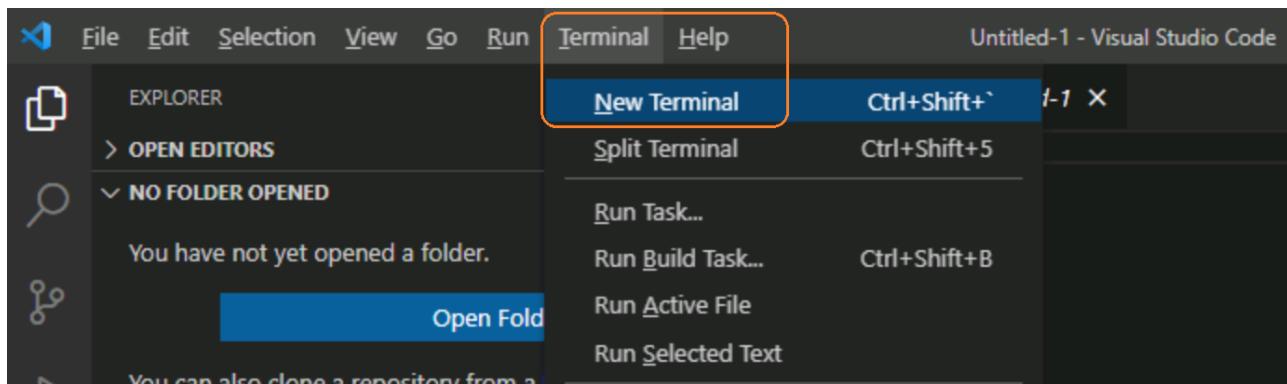
Task 1: Configuring Visual Studio Code

1. Open **Visual Studio Code** by clicking on the VS Code icon in the taskbar.



In this task, you will configure a Git credential helper to securely store the Git credentials used to communicate with Azure DevOps. If you have already configured a credential helper and Git identity, you can skip to the next task.

2. From the main menu, select **Terminal | New Terminal** to open a terminal window.



3. Execute the command below to configure a credential helper.

```
git config --global credential.helper wincred
```

4. The commands below will configure your user name and email for Git commits. You can replace the parameters with your preferred user name and email and execute them.

```
git config --global user.name "[Your Name]"
```

```
git config --global user.email [YourEmail]
```

TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE

2: pwsh

PowerShell 7.1.2
Copyright (c) Microsoft Corporation.

<https://aka.ms/powershell>
Type 'help' to get help.

A new PowerShell stable release is available: v7.1.4
Upgrade now, or check out the release page at:
<https://aka.ms/PowerShell-Release?tag=v7.1.4>

Loading personal and system profiles took 515ms

```
C:\Users\StudentPC> git config --global credential.helper wincred
C:\Users\StudentPC> git config --global user.name "Student1-18875972"
C:\Users\StudentPC> git config --global user.email Student1-18875972@lodsasdoutlook.onmicrosoft.com
C:\Users\StudentPC>
```

Module 3: Azure Repos with Git, Lab 1: Getting Started with Git, Exercise 2: Cloning Existing Azure Repository

Exercise 2: Cloning Existing Azure Repository

Objectives

Git is a distributed version control system. Git repositories can reside locally, such as on a developer's machine and can be hosted by Azure DevOps Services. You will learn how to establish a local Git repository, which can easily be synchronized with a centralized Git repository on the Azure DevOps Services.

Prerequisites

- [Exercise 1](#)

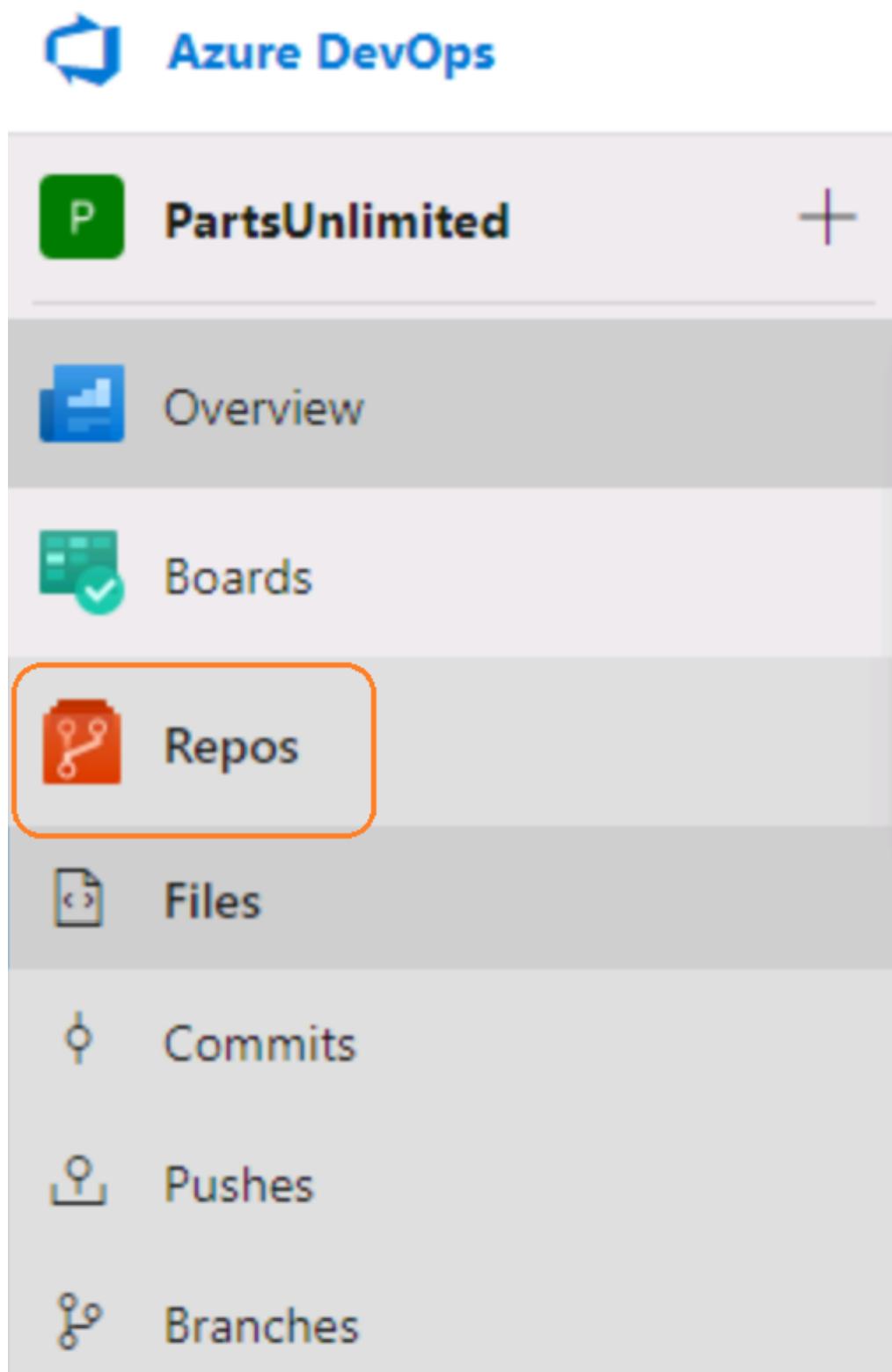
Tasks

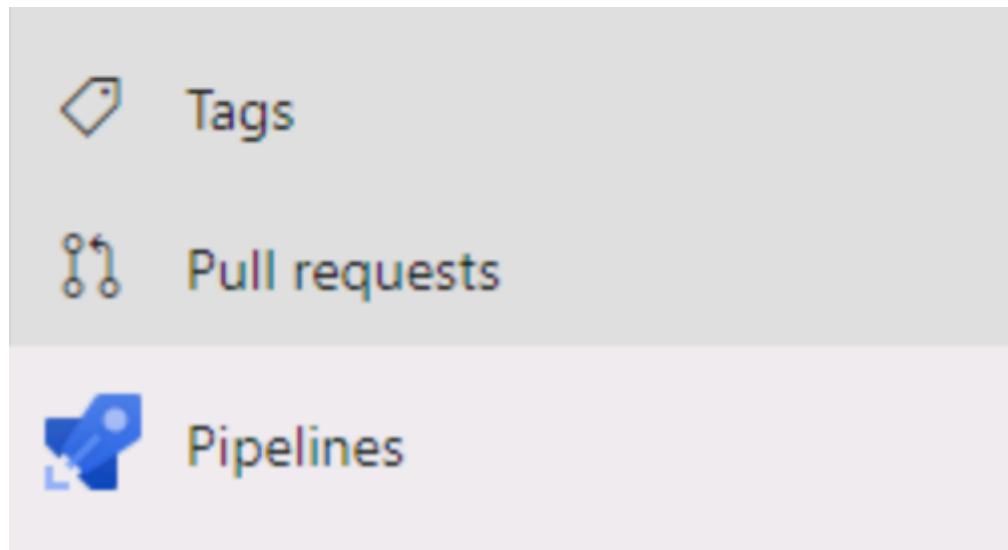
- [Task 1: Cloning an Existing Repository](#)
- [Task 2: Configuring the PartsUnlimited Project](#)

Module 3: Azure Repos with Git, Lab 1: Getting Started with Git, Exercise 2: Cloning Existing Azure Repository

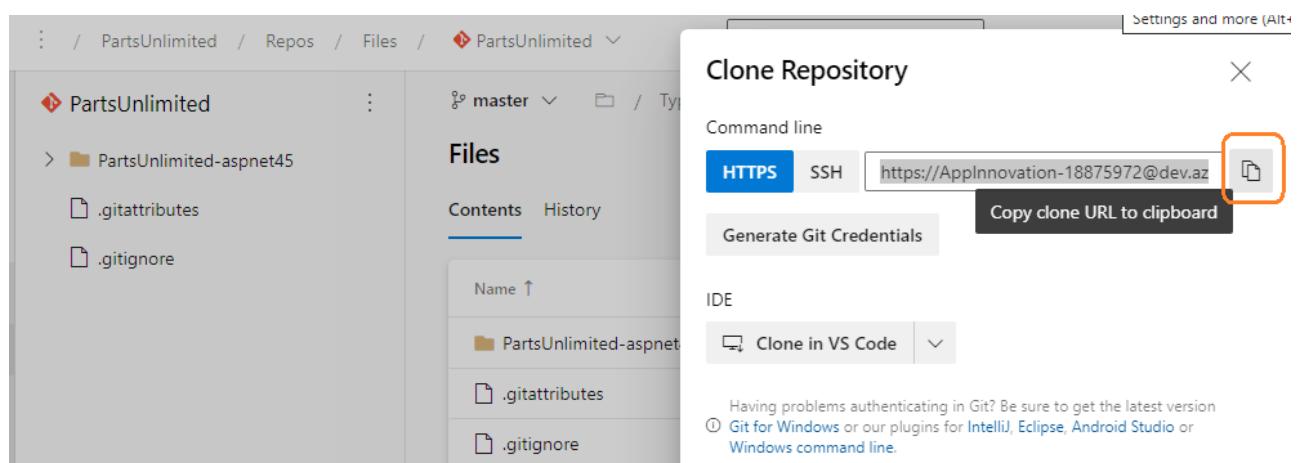
Task 1: Cloning an Existing Repository

1. Navigate to Azure DevOps Services in the browser to the **PartsUnlimited** project. The URL would be [https://dev.azure.com/AppInnovation-\[YourName\]/PartsUnlimited](https://dev.azure.com/AppInnovation-[YourName]/PartsUnlimited).
2. Navigate to the **Repos** hub and click **Clone**.





3. Click the **Copy clone URL to clipboard** button next to the repo clone URL. You can plug this URL into any Git-compatible tool to get a copy of the codebase.

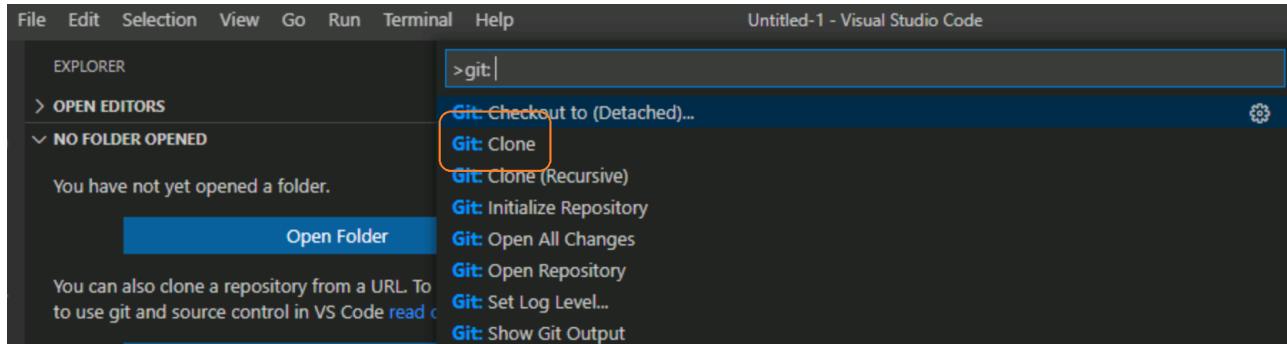


4. Switch back to the instance of **Visual Studio Code**.

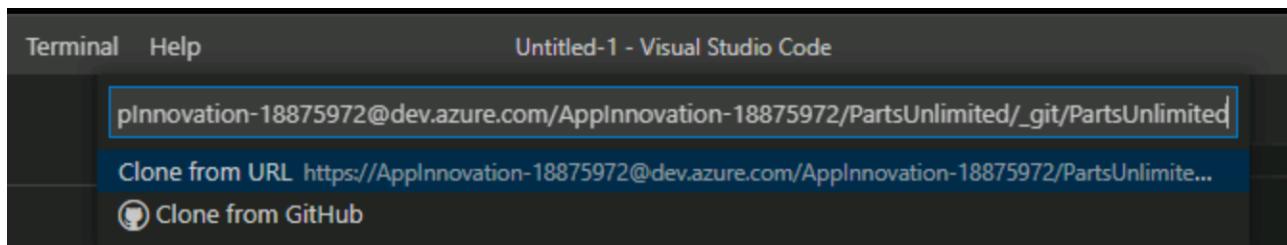
5. Press **Ctrl+Shift+P** to show the **Command Palette**. Alternatively, you can open the Command Palette by navigating to **View -> Command Palette**.

The Command Palette provides an easy and convenient way to access a wide variety of tasks, including those provided by 3rd party extensions.

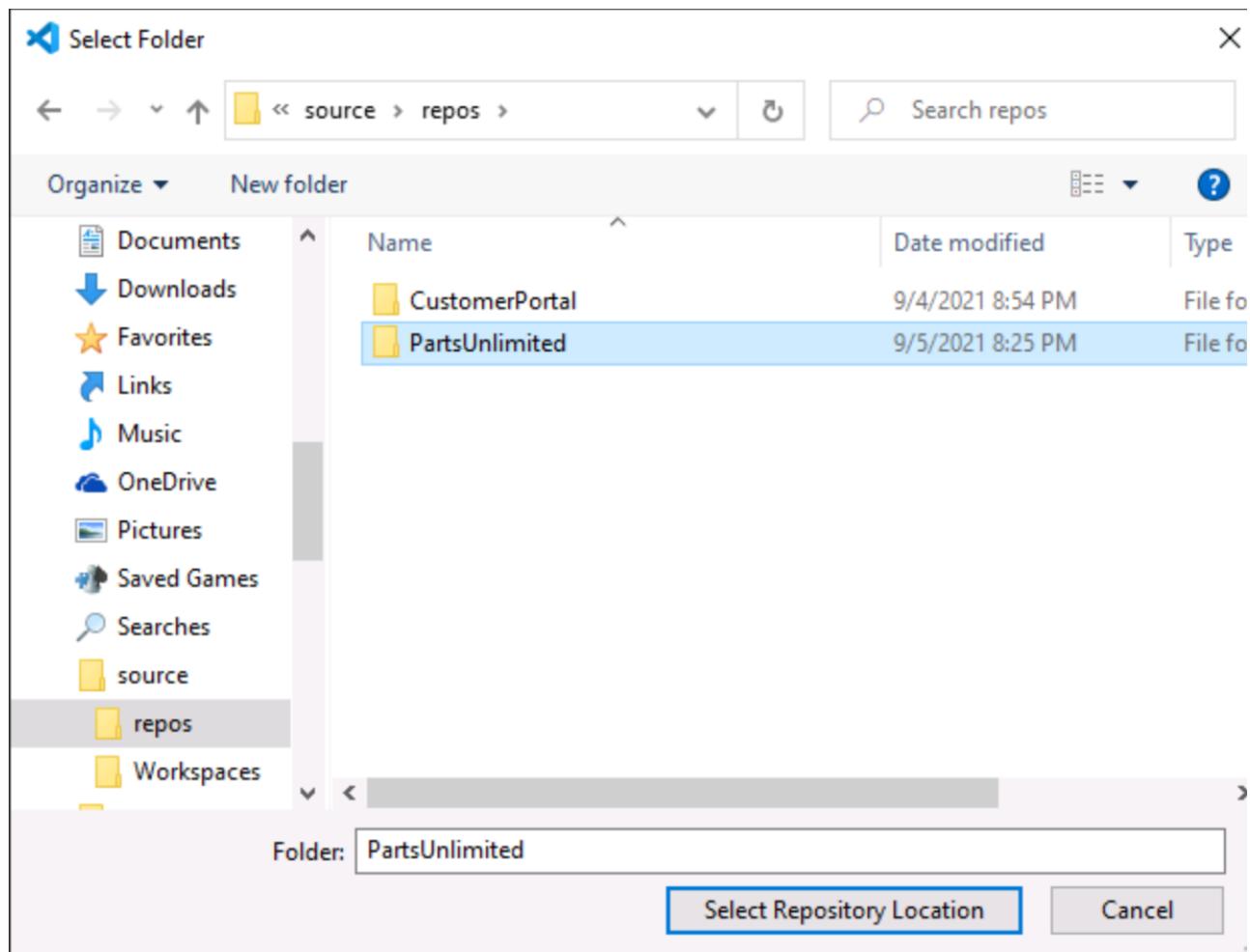
6. Execute the **Git: Clone** command. It may help to type "Git" to bring it to the shortlist.



7. Paste in the URL to your repo ([https://ApplInnovation-\[YourName\]@dev.azure.com/ApplInnovation-\[YourName\]/PartsUnlimited/_git/PartsUnlimited](https://ApplInnovation-[YourName]@dev.azure.com/ApplInnovation-[YourName]/PartsUnlimited/_git/PartsUnlimited)) and press **Enter**.

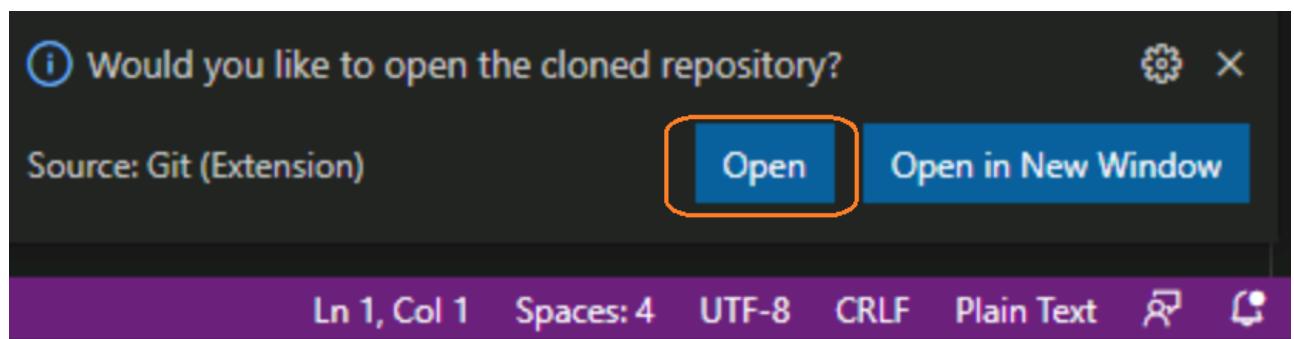


8. Navigate to the local path **C:\Users\StudentPC\source\repos**. Create a new folder **PartsUnlimited** and point to this folder.



9. If prompted, log in to your Azure DevOps account enter your credentials. Once the cloning is completed, click **Open** to open the cloned repository.

You can ignore any warnings raised about opening the projects. The solution may not be in a buildable state, but that's okay since we're going to focus on working with Git and building the project itself is not necessary.



Module 3: Azure Repos with Git, Lab 1: Getting Started with Git, Exercise 2: Cloning Existing Azure Repository

Task 2: Configuring the PartsUnlimited Project

1. Navigate to the **PartsUnlimited** project in the Web Portal. Before digging into Git, you will want to disable the **Continuous Integration** trigger for the default build definition.
2. Navigate to **Pipelines | Pipelines**.

The screenshot shows the Azure DevOps Pipelines interface for the 'PartsUnlimited' project. The left sidebar lists project navigation options: Overview, Boards, Repos, Pipelines (which is selected and highlighted with an orange border), Environments, Releases, Library, Task groups, and Deployment groups. The main content area is titled 'Pipelines' and shows the 'Recently run pipelines' section. It lists one pipeline: 'PartsUnlimitedE2E', which has 'No runs' indicated. The browser address bar shows the URL: https://dev.azure.com/AppInnovation-18875972/PartsUnlimited/_build.

3. PartsUnlimited project includes a default build definition, PartsUnlimitedE2E. Click on **More options** and click **Edit**.

The screenshot shows the Azure DevOps Pipelines interface. On the left, there's a sidebar with various project management and development tools like Boards, Repos, Pipelines, Environments, Releases, Library, Task groups, and Deployment groups. The main area displays a list of recently run pipelines. One pipeline, 'PartsUnlimitedE2E', is shown with the status 'No runs yet'. To the right of this pipeline is a context menu with options: Edit (highlighted with an orange box), Run pipeline, Manage security, Rename/move, and Delete.

4. This build is scheduled to run whenever a change is committed. We don't want to wait for this build every time there's a change, so select the **Triggers** tab and uncheck **Enable continuous integration**.

This checkbox triggers the build pipeline whenever there is a change in the master branch. This is the CI component of the CI/CD process which we will discuss later. For now, we need to disable CI so that our changes in the master branch in this lab won't trigger builds every time.

The screenshot shows the build definition 'PartsUnlimitedE2E'. At the top, there are tabs for Tasks, Variables, Triggers (which is highlighted with an orange box), Options, History, and other navigation links. The 'Continuous integration' section shows a single trigger named 'PartsUnlimited' with the status 'Disabled'. To the right, there's a summary panel for 'PartsUnlimited' which includes a checkbox labeled 'Enable continuous integration' (also highlighted with an orange box).

5. To be able to save the change in the build definition, we need to fix all the issues. Select the **Tasks** tab and select **windows-latest** in the dropdown for **Agent Specification**.

Pipeline
Build pipeline

Phase 1
Run on agent

- Get sources
- NuGet restore
- Build solution
- Test Assemblies
- Copy Files
- Publish Artifact

Name *
PartsUnlimitedE2E

Agent pool | Pool information | Manage
Azure Pipelines

Agent Specification *
windows-latest

Parameters
This pipeline doesn't have any pipeline parameters. Create them to share the most important settings between tasks and change them in one place.
Learn more

6. From the Save & queue dropdown, select **Save**.

Pipeline
Build pipeline

Phase 1
Run on agent

- Get sources
- NuGet restore
- Build solution
- Test Assemblies
- Copy Files
- Publish Artifact

Name *
PartsUnlimitedE2E

Agent pool | Pool information | Manage
Azure Pipelines

Agent Specification *
windows-latest

Parameters
This pipeline doesn't have any pipeline parameters. Create them to share the most important settings between tasks and change them in one place.
Learn more

Save build pipeline X

Comment

Save

Cancel

Module 3: Azure Repos with Git, Lab 1: Getting Started with Git, Exercise 3: Saving Work Locally with Commits

Exercise 3: Saving Work Locally with Commits

Objectives

In this exercise, you will learn how to change code in a Git repository.

When you make changes to your files, Git will record the changes in the local repository. You can select the changes that you want to commit by staging the changes. Commits are always made against your local Git repository, so you don't have to worry about the commit being perfect or ready to share with others. You can make more commits as you continue to work and push the changes to others when they are ready to be shared.

What's in a commit?

Git commits consist of the following:

- The file(s) changed in the commit. Git keeps the contents of all file changes in your repo in the commits. This keeps it fast and allows intelligent merging.
- A reference to the parent commit(s). Git manages your code history using these references.
- A message describing a commit. You give this message to Git when you create the commit. It's a good idea to keep this message descriptive, but to the point.

Prerequisites

- Complete [Exercise 2](#)

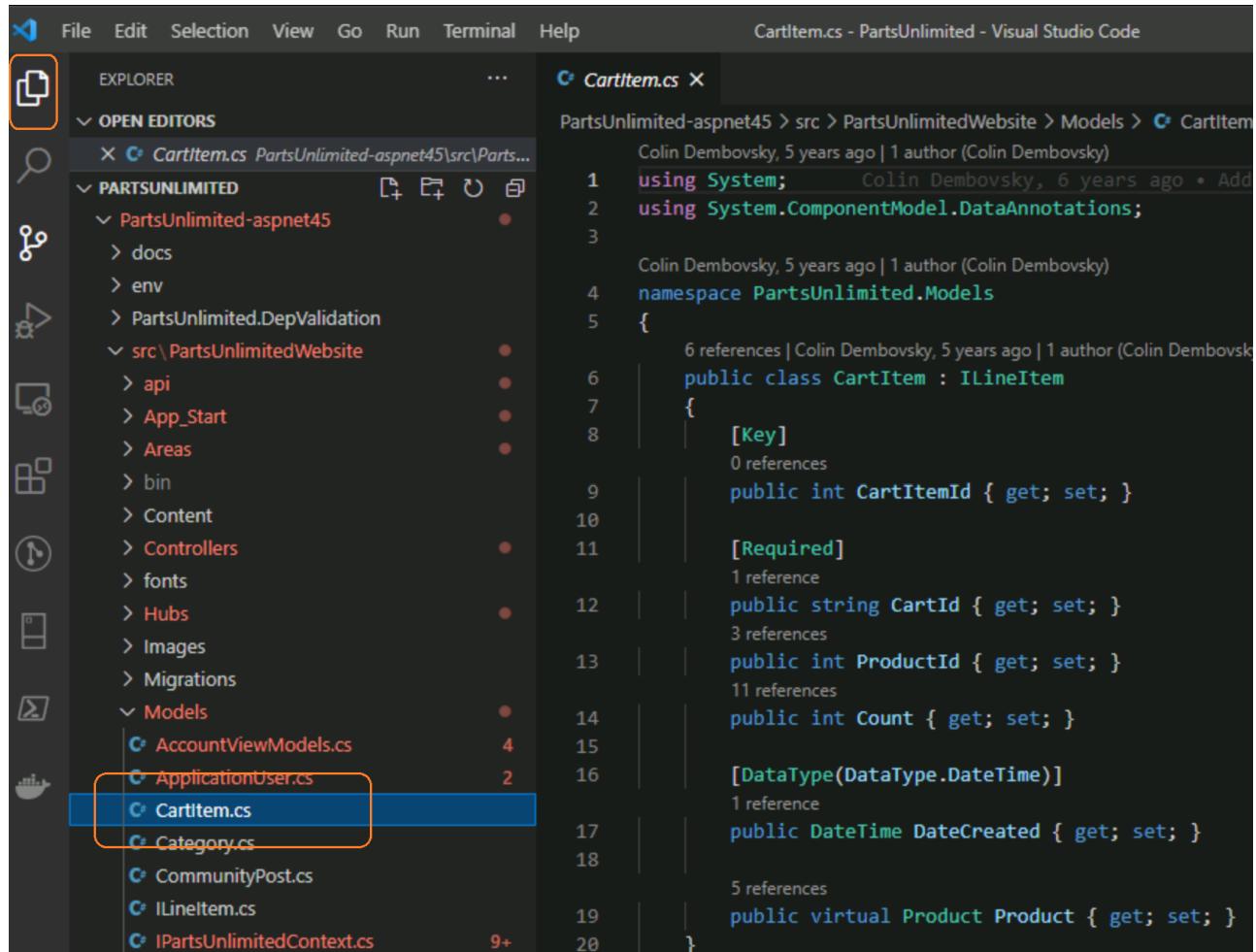
Tasks

- [Task 1: Committing Changes](#)
- [Task 2: Reviewing Commits](#)
- [Task 3: Staging Changes](#)

Module 3: Azure Repos with Git, Lab 1: Getting Started with Git, Exercise 3: Saving Work Locally with Commits

Task 1: Committing Changes

1. Go to the **Visual Studio Code** that has PartsUnlimited project open. From the Explorer tab, open `/PartsUnlimited-aspnet45/src/PartsUnlimitedWebsite/Models/CartItem.cs`.

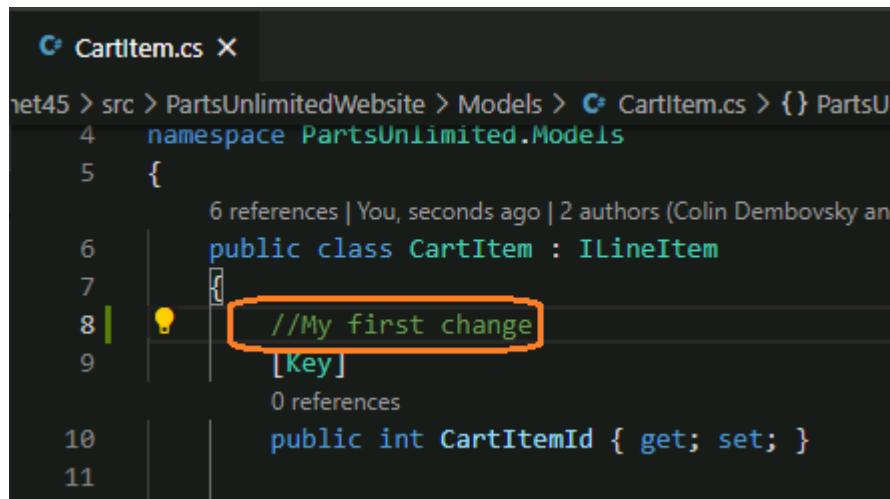


The screenshot shows the Visual Studio Code interface. The Explorer sidebar on the left shows a tree view of the project structure under 'PARTSUNLIMITED'. The 'src\PartsUnlimitedWebsite' folder contains 'api', 'App_Start', 'Areas', 'bin', 'Content', 'Controllers', 'fonts', 'Hubs', 'Images', 'Migrations', and 'Models'. Inside 'Models', there are files: 'AccountViewModels.cs' (4), 'ApplicationUser.cs' (2), 'CartItem.cs' (selected and highlighted with a blue border), 'Category.cs', 'CommunityPost.cs', 'ILineItem.cs', and 'IPartsUnlimitedContext.cs' (9+). The main editor window on the right displays the code for 'CartItem.cs'. The code defines a class 'CartItem' that implements 'ILineItem'. It includes properties for CartItemId (with [Key] and [Required] annotations), CartId, ProductId, Count, DateCreated (with [DataType(DataType.DateTime)] annotation), and Product (a virtual property). There are also comments indicating authorship by Colin Dembovsky.

```
1 using System;           Colin Dembovsky, 6 years ago • Add
2 using System.ComponentModel.DataAnnotations;
3
4 namespace PartsUnlimited.Models
5 {
6     public class CartItem : ILineItem
7     {
8         [Key]
9         public int CartItemId { get; set; }
10
11        [Required]
12        public string CartId { get; set; }
13
14        public int ProductId { get; set; }
15
16        public int Count { get; set; }
17
18        [DataType(DataType.DateTime)]
19        public DateTime DateCreated { get; set; }
20    }
}
```

2. Add a comment to the file: `//My first change`. Press **Ctrl+S** to save the file. You can also save the file by navigating to **File -> Save**

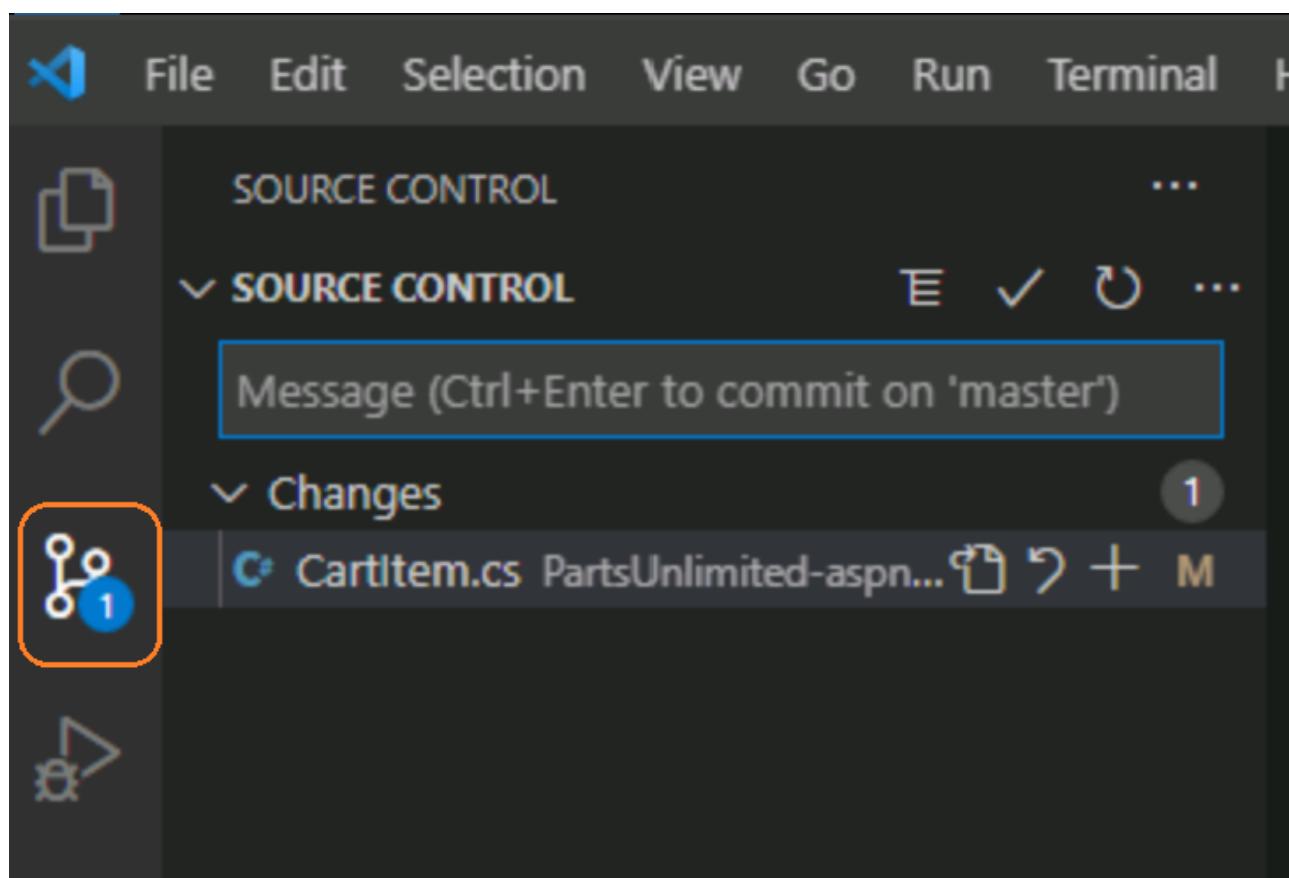
It doesn't really matter what the comment is and where you add it in the file since the goal is just to make a change in the file.



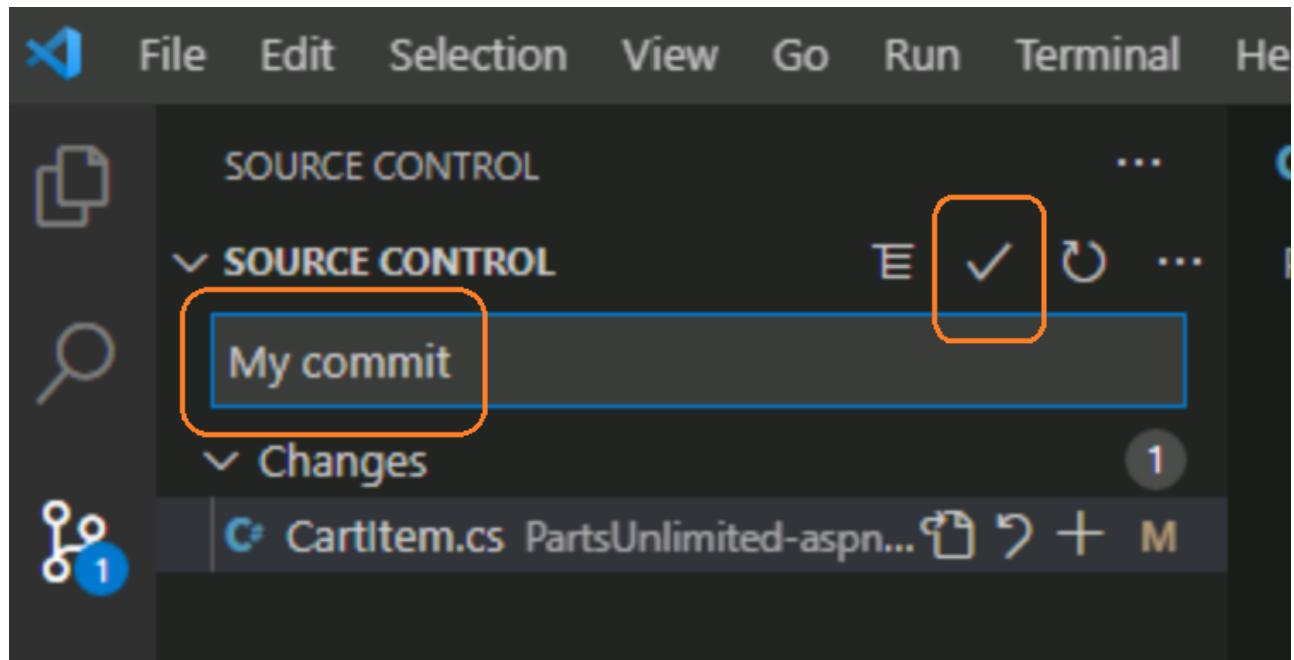
```
CartItem.cs X

4  namespace PartsUnlimited.Models
5  {
6      public class CartItem : ILineItem
7      {
8          //My first change
9          [Key]
10         public int CartItemId { get; set; }
11     }
}
```

3. Select the **Source Control tab** to see the one change to the solution.

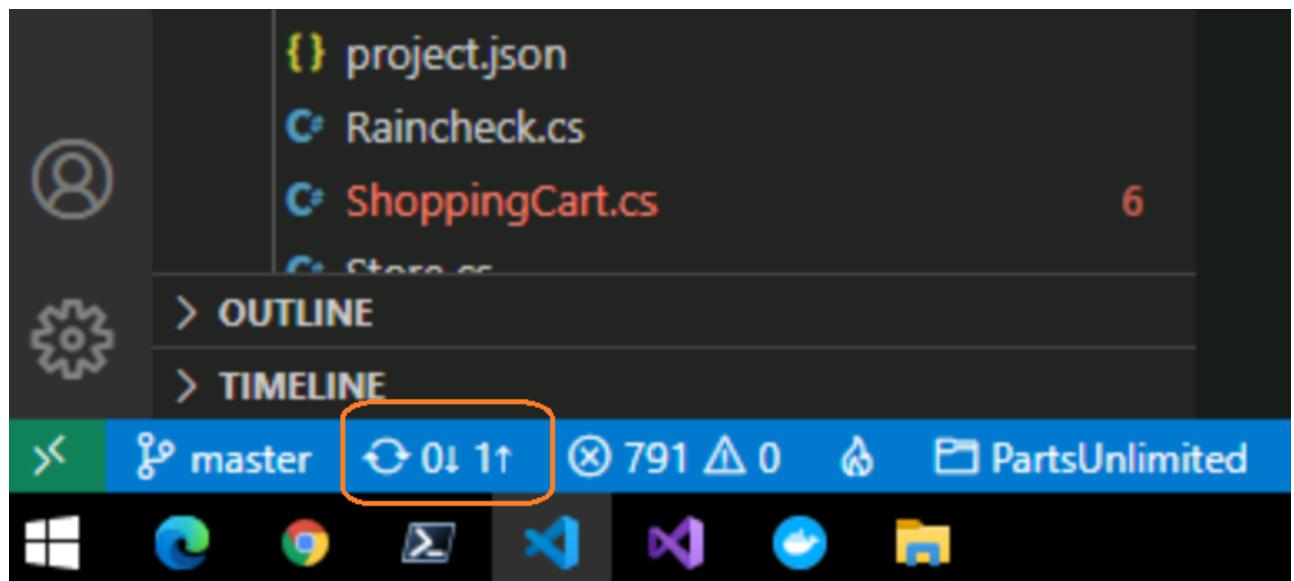


4. Enter a commit message of **My commit** and press **Ctrl+Enter** to commit it locally. Alternatively, you can click on **Commit**.



If asked whether you would like to **automatically stage your changes and commit them directly**, click **Always**. We will discuss staging later in the lab.

5. Click **Sync** to **Push commits to origin/master** on the bottom-left of VS Code to synchronize local repository changes with the server. Confirm the sync if prompted.



Since you have made one commit locally that we need to push to the remote repository, the sync icon reflects this one change to be uploaded.

Module 3: Azure Repos with Git, Lab 1: Getting Started with Git, Exercise 3: Saving Work Locally with Commits

Task 2: Reviewing Commits

1. Switch to the web browser that has Azure DevOps Services open. Navigate to **Commits** under **Repos** hub. You can review the latest commits in Azure DevOps.

The screenshot shows the Azure DevOps interface for the 'PartsUnlimited' project. At the top, there's a green button with a white 'P' and the text 'PartsUnlimited'. To its right is a red '+' sign. Below this, there's a horizontal line. The main area contains several items:

- Overview**: Represented by a blue icon with a chart.
- Boards**: Represented by a green icon with a checkmark.
- Repos**: Represented by an orange icon with a gear and branch symbols.
- Files**: Represented by a blue icon with a document.
- Commits**: Represented by a grey icon with a circular arrow.
- Pushes**: Represented by a grey icon with a person icon.
- Branches**: Represented by a grey icon with a branch symbol.
- Tags**: Represented by a grey icon with a tag symbol.
- Pull requests**: Represented by a grey icon with two circular arrows.

The 'Commits' item is highlighted with a light grey background, indicating it is the current view.

2. The recent commit **My commit** should be right at the top.

The screenshot shows a GitHub repository interface for 'PartsUnlimited'. The 'Commits' tab is selected. On the left, there's a 'Graph' view showing a vertical timeline of commits. On the right, the 'Commit' details are shown for each commit. The most recent commit, 'My commit' (commit hash 4c0c1133), is highlighted with an orange border. This commit was made by 'Student1-18875972' today at 12:53 PM. Below it is another commit by 'Akshay H' (commit hash 8086ee35) on Sep 30, 2019, at 7:30 AM, which updated 'FullEnvironmentSetupMerged.json'. Further down is a commit by 'Srivatsa Marichi' (commit hash 58d9b870) on Nov 24, 2017, at 1:52 AM, which updated 'FullEnvironmentSetupMerged.param.json'. Another commit by 'Srivatsa Marichi' (commit hash bef6d46f) on Nov 24, 2017, at 1:47 AM, updated 'FullEnvironmentSetupMerged.json'. The oldest commit shown is 'Removing ARM project' (commit hash 816d9487) by 'vsts' on Jan 30, 2017, at 7:47 PM.

Commit Hash	Author	Date	Message
4c0c1133	Student1-18875972	Today at 12:53 PM	My commit
8086ee35	Akshay H	Sep 30, 2019 at 7:30 AM	Updated FullEnvironmentSetupMerged.json
58d9b870	Srivatsa Marichi	Nov 24, 2017 at 1:52 AM	Updated FullEnvironmentSetupMerged.param.json
bef6d46f	Srivatsa Marichi	Nov 24, 2017 at 1:47 AM	Updated FullEnvironmentSetupMerged.json
816d9487	vsts	Jan 30, 2017 at 7:47 PM	Removing ARM project

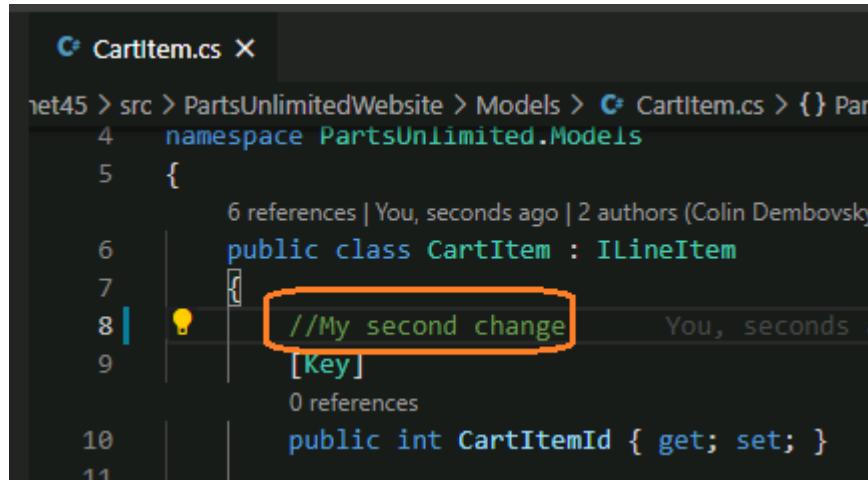
Also note that **My commit** commit shows the person who performed the commit. This should match with the username you setup in VS Code at the beginning of this lab.

Module 3: Azure Repos with Git, Lab 1: Getting Started with Git, Exercise 3: Saving Work Locally with Commits

Task 3: Staging Changes

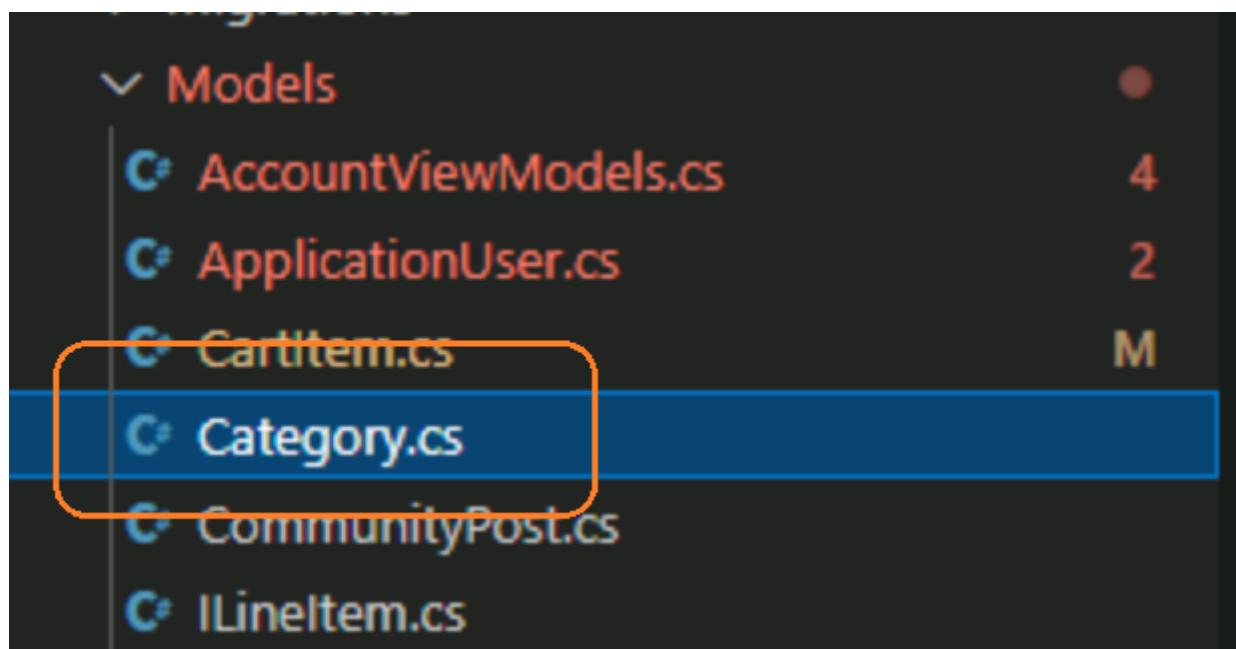
Staging changes allows you to selectively add certain files to a commit while passing over the changes made in other files.

1. Return to **Visual Studio Code** with **PartsUnlimited** project open in it.
2. Edit the **CartItem.cs** file by editing the comment you made earlier and changing it to **//My second change**. **Save** the changes.

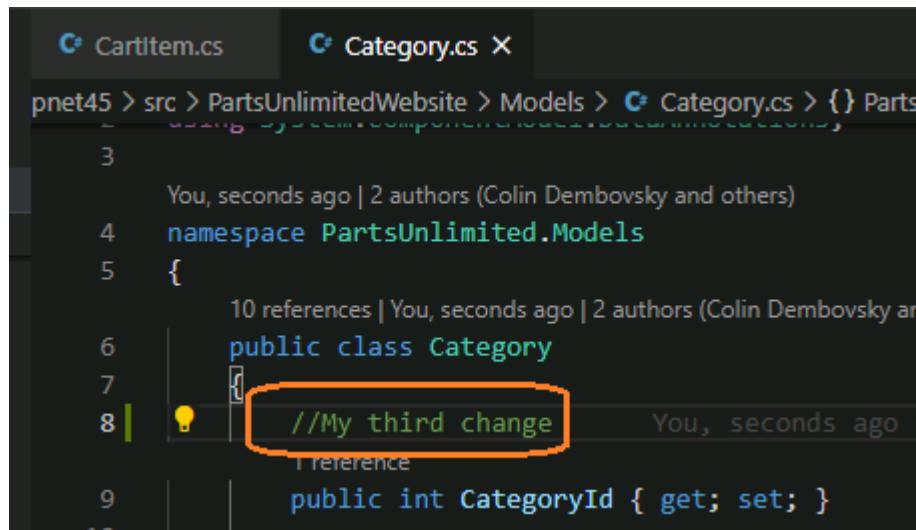


```
CartItem.cs X
jet45 > src > PartsUnlimitedWebsite > Models > CartItem.cs > {} Part
4   namespace PartsUnlimited.Models
5   {
6       6 references | You, seconds ago | 2 authors (Colin Dembovsky
7       public class CartItem : ILineItem
8       {
9           8 | [Key]
10          0 references
11          public int CartItemId { get; set; }
```

3. Click on the **Explorer** in the left tab of Visual Studio Code and open **Category.cs** file.

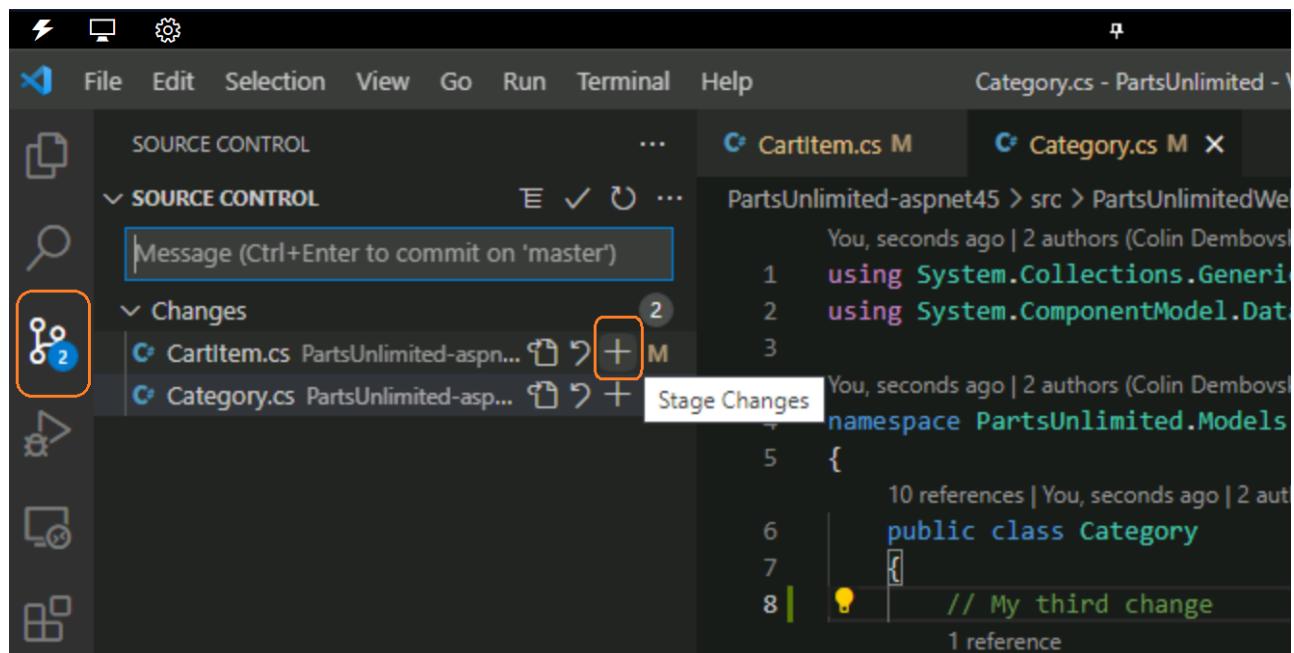


4. Add a new comment to Category.cs file: **//My third change** so there will be two files with changes. **Save** the file.

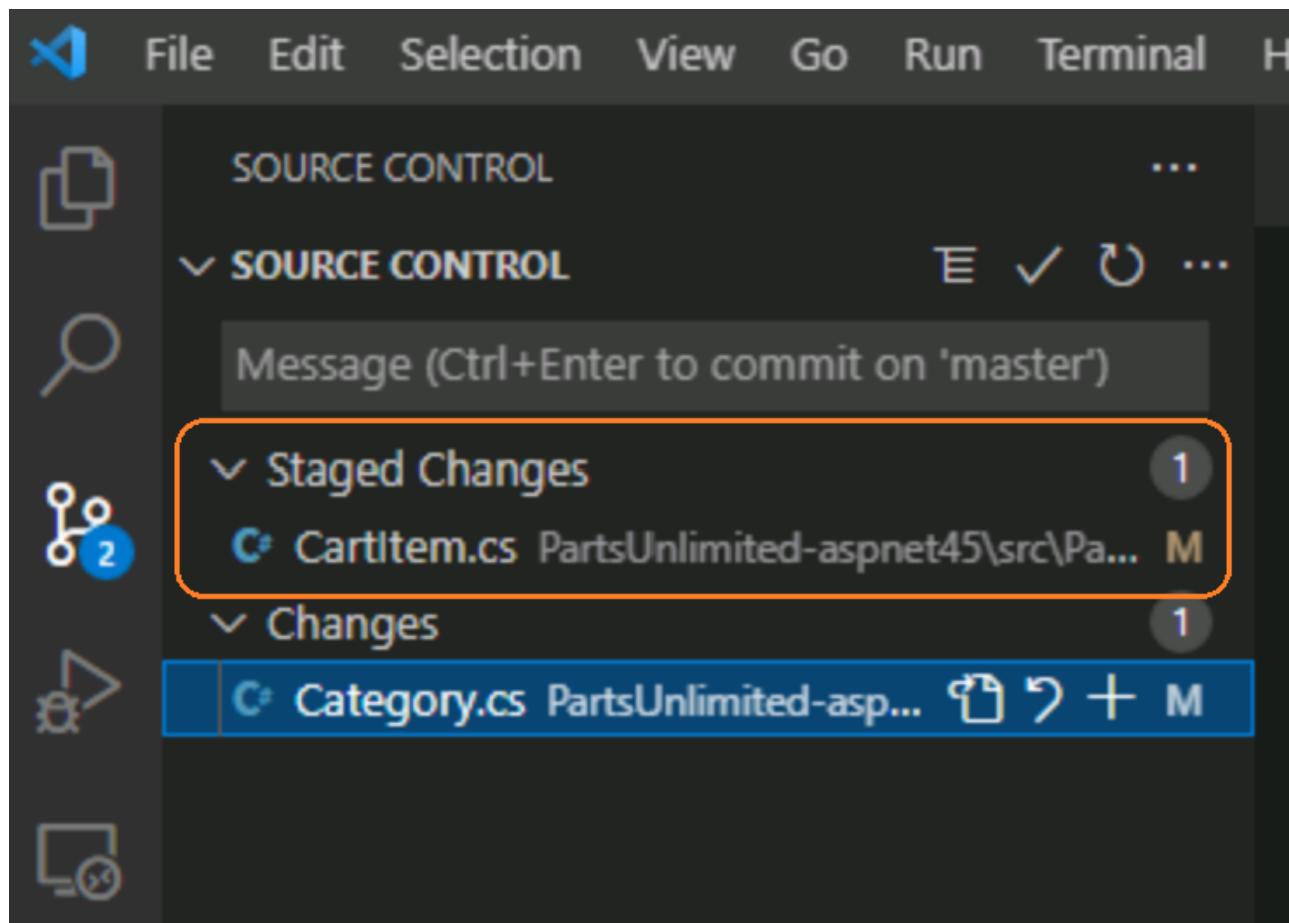


```
CartItem.cs Category.cs X
pnet45 > src > PartsUnlimitedWebsite > Models > Category.cs > { } Parts
3
You, seconds ago | 2 authors (Colin Dembovsky and others)
4 namespace PartsUnlimited.Models
5 {
6     10 references | You, seconds ago | 2 authors (Colin Dembovsky an
7         public class Category
8             [ ] // My third change You, seconds ago
9                 1 reference
10                public int CategoryId { get; set; }
```

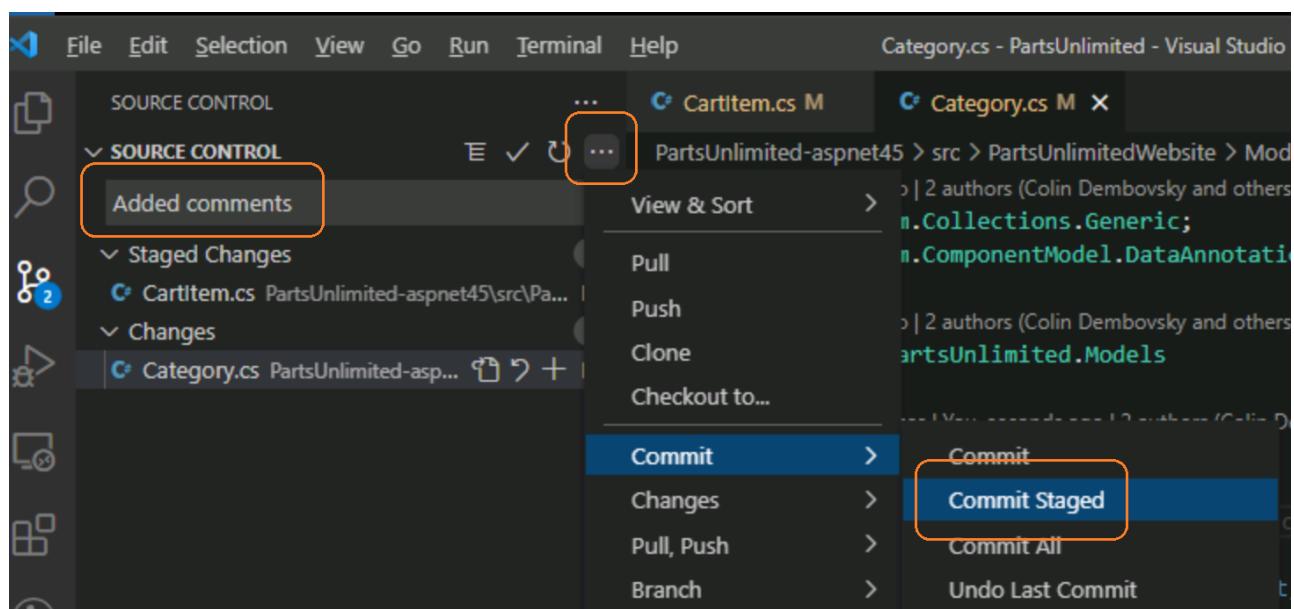
5. From the **Source Control** tab, click the **Stage Changes (+)** button for **CartItem.cs**.



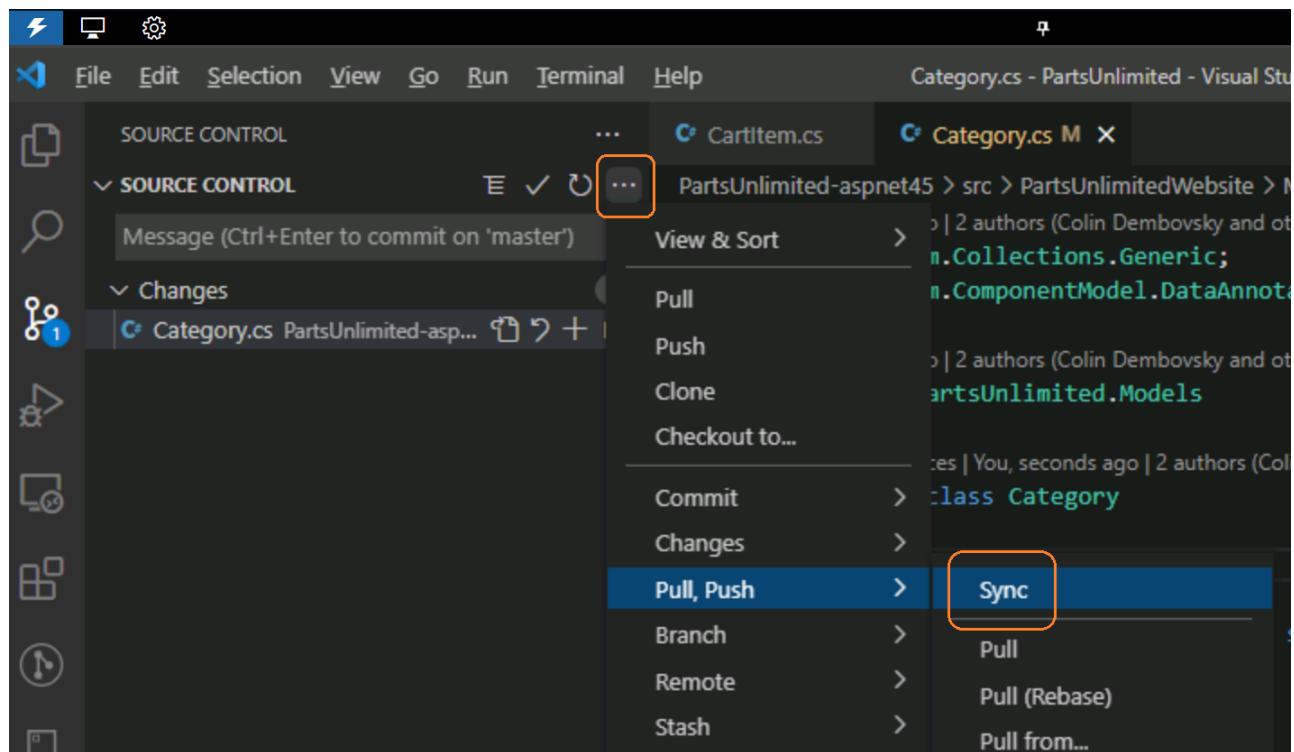
6. This will prepare CartItem.cs for committing without Category.cs.



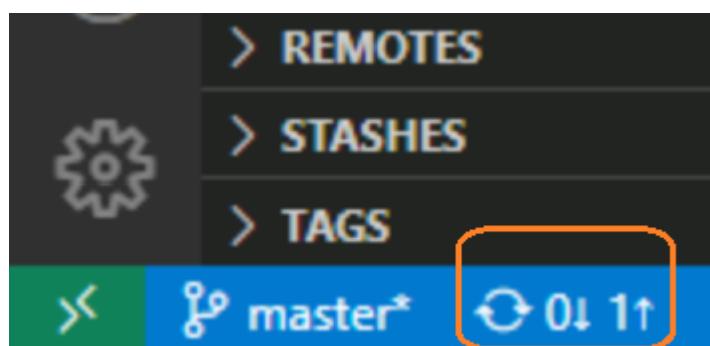
7. Enter a commit message **Added comments**. From the **More Actions** dropdown, select **Commit Staged**.



8. From **More Actions**, select **Pull, Push -> Sync**. This will synchronize changes to the remote repository in Azure DevOps under PartsUnlimited project. Since only the staged changes were committed, the other changes are still pending locally.



Alternatively, you can also quickly sync changes by clicking the **Synchronize Changes** at the bottom-left corner of VS Code.



Module 3: Azure Repos with Git, Lab 1: Getting Started with Git, Exercise 4: Reviewing History in Azure Repos

Exercise 4: Reviewing History in Azure Repos

Objectives

Git uses the parent reference information stored in each commit to manage a full history of your development. You can easily review this commit history to find out when file changes were made and determine differences between versions of your code.

Git uses branches and merges feature works through Pull Requests, so the commit history of your development doesn't necessarily form a straight, chronological line. When you use history to compare versions, think in terms of file changes between two commits instead of file changes between two points in time. A recent change to a file in the master branch may have come from a commit created two weeks ago in a feature branch but was only merged yesterday.

Prerequisites

- Complete [Exercise 3](#)

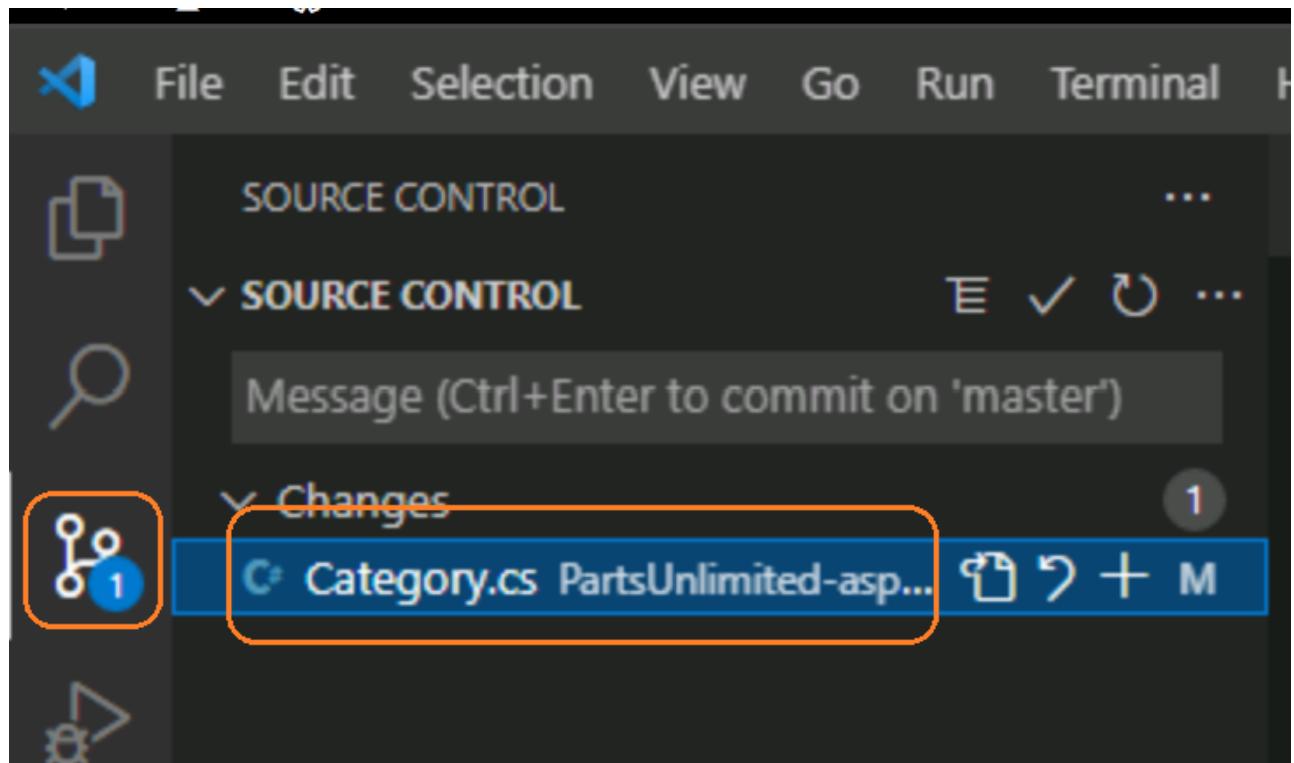
Tasks

- [Task 1: Comparing Files](#)

Module 3: Azure Repos with Git, Lab 1: Getting Started with Git, Exercise 4: Reviewing History in Azure Repos

Task 1: Comparing Files

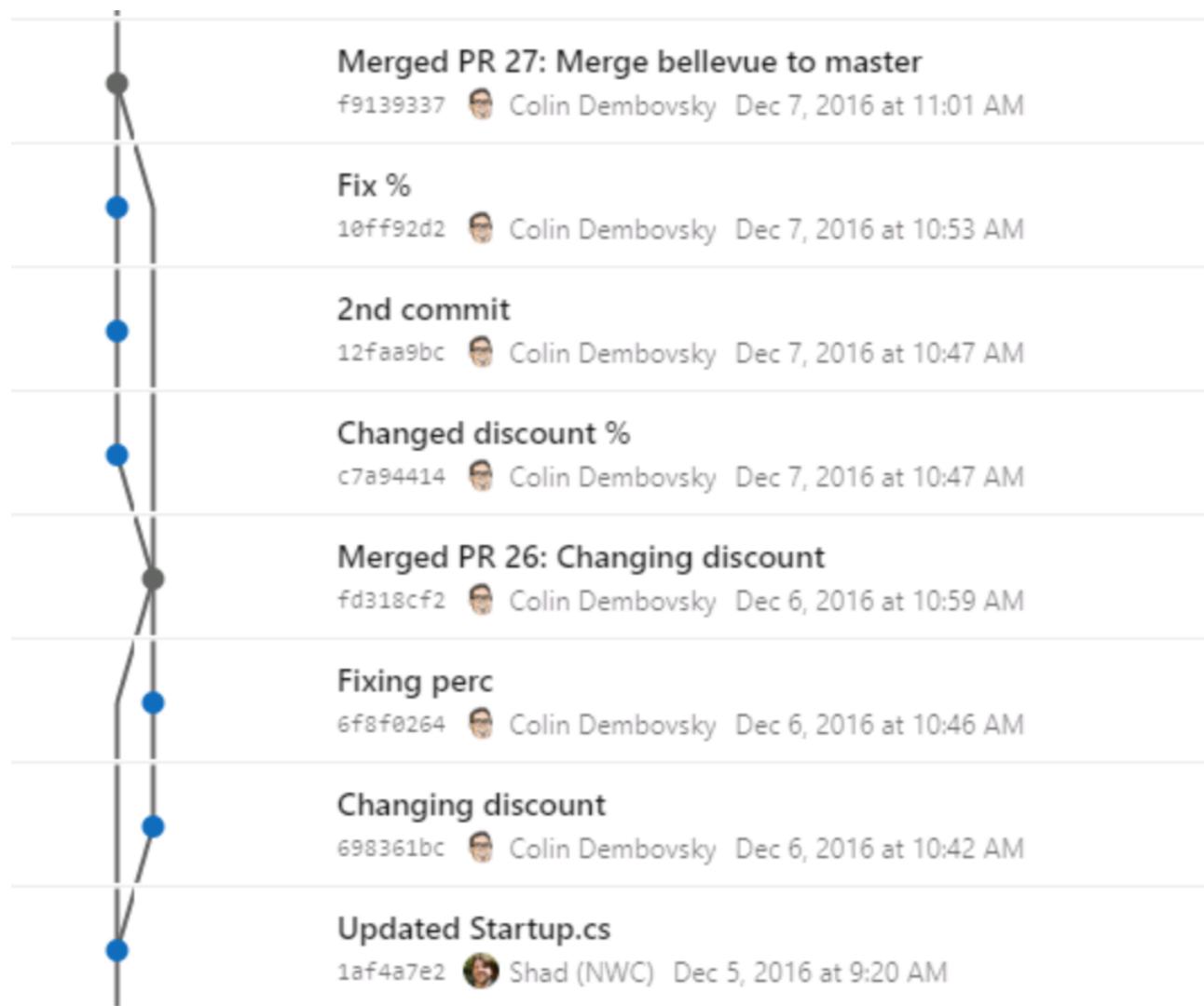
1. In **Visual Studio Code**, click on the **Source Control** tab and then click on **Category.cs** file.



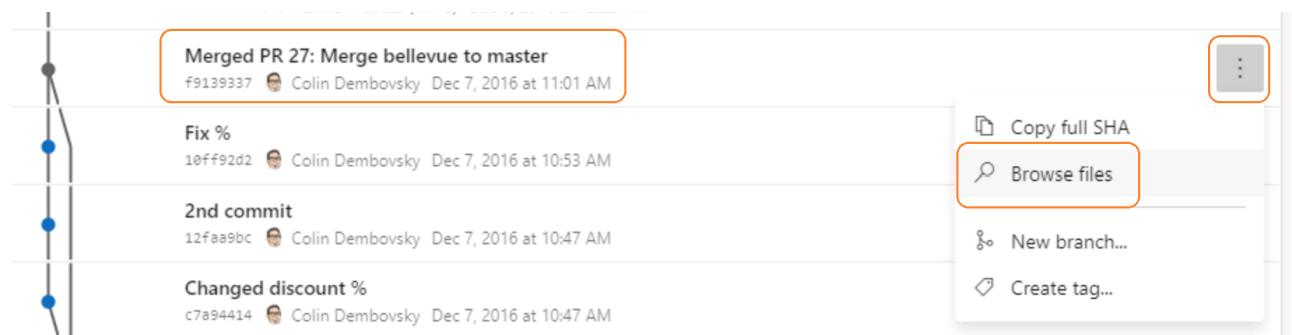
2. A comparison view is opened to enable you to easily locate the changes you've made. In this case, it's just the one comment.

A screenshot of the Visual Studio Code interface showing a code comparison between "Category.cs" and "Category.cs (Working Tree)". The left pane shows the original code, and the right pane shows the modified code with a green highlight. Line 8 is annotated with a plus sign and "My third change" above the code. The status bar at the bottom right shows "1" changes.

3. Go to **Azure DevOps Services** in the web browser and navigate to **Repos** hub. Click on **Commits** to view list of all the commit history along with the commit graph. **Scroll down** to locate the commit history of some merges and Pull Requests. These provide a convenient way to visualize when and how changes were made to the source.



4. Locate **Merged PR 27: Merge bellevue to master** and click on **More options** and select **Browse files**.



5. This view offers the ability to navigate around the state of the source at that commit so you can review and download those files. You can drill into a folder, locate a file, and download.

You don't need to download any files right now.

⌚ f9139337 ⚽ / src / PartsUnlimitedWebsite / Models

Models

+ New ⚽ :

Contents History ↗

Name ↑	Last change	Commits
C# AccountViewModels.cs	Nov 18, 2015	45f7ffff9 Add PartsU...
C# ApplicationUser.cs	Nov 18, 2015	45f7ffff9 Add PartsU...
C# CartItem.cs	Nov 18, 2015	45f7ffff9 Add PartsU...
C# Category.cs	Nov 18, 2015	Edit
C# CommunityPost.cs	Nov 18, 2015	Rename
C# IPartsUnlimitedContext.cs	Nov 18, 2015	Delete
C# ManageViewModels.cs	Nov 18, 2015	Download

The screenshot shows a file listing interface for a GitHub repository. The path is 'src / PartsUnlimitedWebsite / Models'. A search bar at the top left contains the commit hash '⌚ f9139337 ⚽'. The main area displays a table of files with columns for Name, Last change, and Commits. The file 'CartItem.cs' is highlighted with an orange box. A context menu is open over this file, showing options: Edit, Rename, Delete, and Download. The 'Edit' option has a red arrow pointing to it. The 'Delete' option is preceded by a trash icon. The 'Download' option is preceded by a download icon.

Module 3: Azure Repos with Git, Lab 1: Getting Started with Git, Exercise 5: Managing Branches Locally from Visual Studio Code

Exercise 5: Managing Branches Locally from Visual Studio Code

Objectives

Committing changes to a branch will not affect other branches, and you can share branches with others without having to merge the changes into the main project. You can also create new branches to isolate changes for a feature or a bug fix from your master branch and other work. Since the branches are lightweight, switching between branches is quick and easy. Git does not create multiple copies of your source when working with branches, but rather uses the history information stored in commits to recreate the files on a branch when you start working on it. Your Git workflow should create and use branches for managing features and bug fixes. The rest of the Git workflow, such as sharing code and reviewing code with pull requests, all work through branches. Isolating work in branches makes it very simple to change what you are working on by simply changing your current branch.

Git keeps track of which branch you are working on and makes sure that when you check out a branch your files match the most recent commit on the branch. Branches let you work with multiple versions of the source code in the same local Git repository at the same time. You can manage the branches in your Visual Studio Code.

Prerequisites

- Complete [Exercise 2](#).

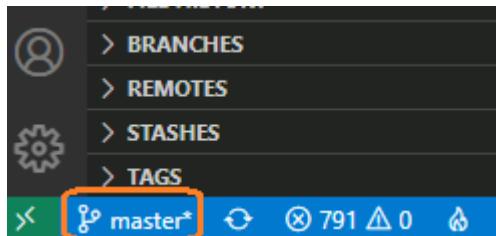
Tasks

- [Task 1: Create a New Branch in Your Local Repository](#)
- [Task 2: Checking Out and Publishing Branches](#)

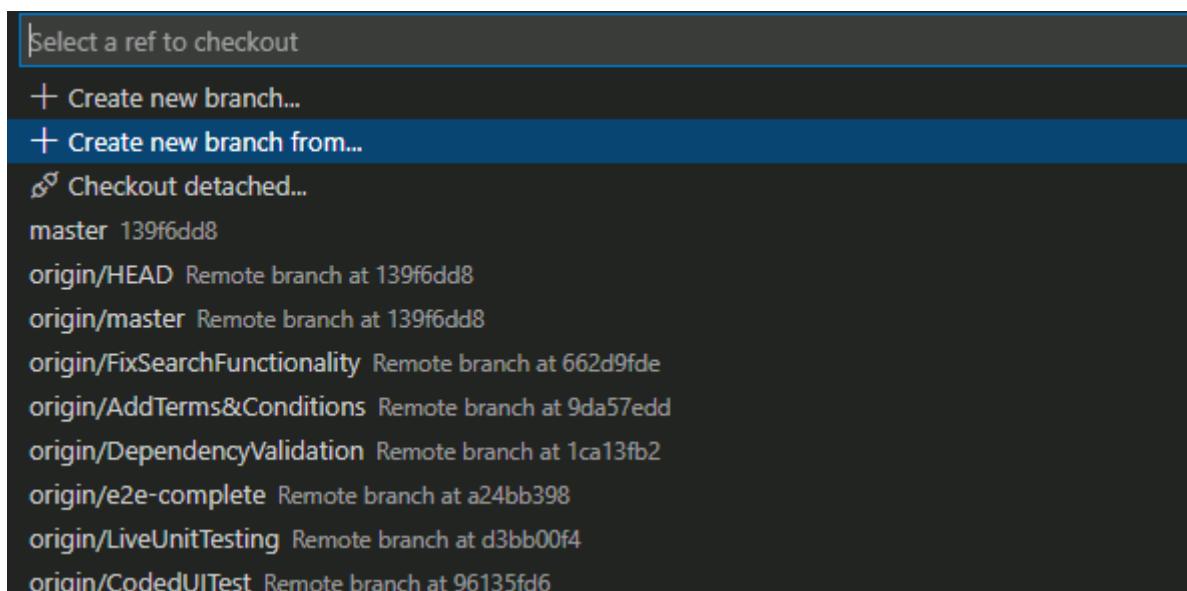
Module 3: Azure Repos with Git, Lab 1: Getting Started with Git, Exercise 5: Managing Branches Locally from Visual Studio Code

Task 1: Create a New Branch in Your Local Repository

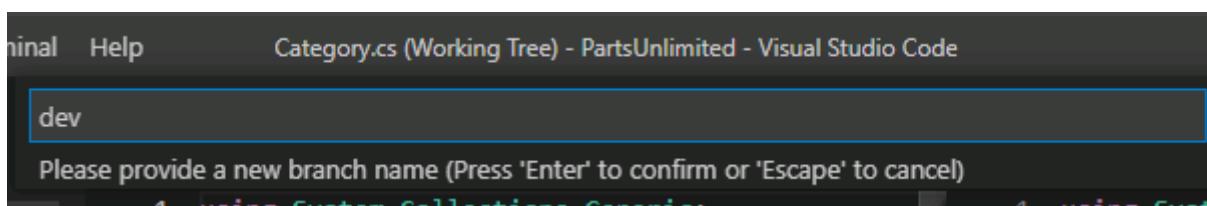
1. Return to **Visual Studio Code** and click the **master** branch from the bottom-left corner.



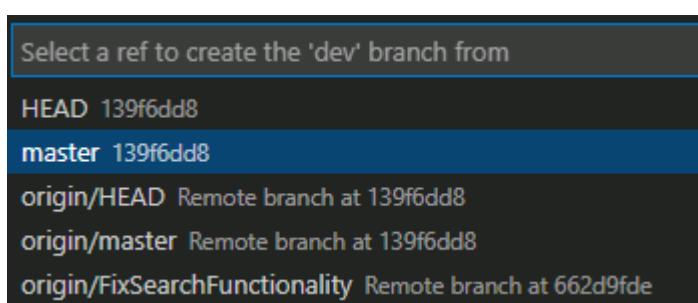
2. Select + **Create new branch from....**



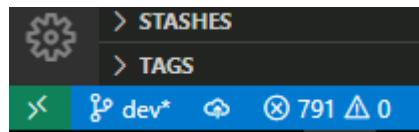
3. Enter the **Branch name** as **dev** for the new branch and press **Enter**.



4. Select the **master** as the reference branch to create the dev branch from.



5. Confirm from the bottom-left corner of the Visual Studio Code that are now working in the **dev** branch.

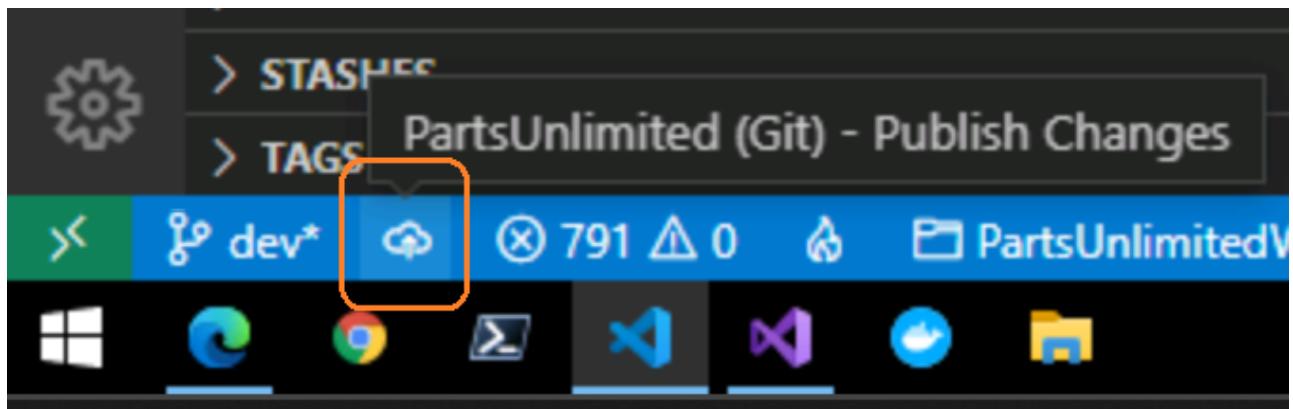


Module 3: Azure Repos with Git, Lab 1: Getting Started with Git, Exercise 5: Managing Branches Locally from Visual Studio Code

Task 2: Checking Out and Publishing Branches

Git keeps track of which branch you are working on and makes sure when you checkout a branch, your files match the most recent commit on the branch. Branches let you work with multiple versions of the source code in the same local Git repository at the same time. You can use Visual Studio Code to publish, check out and delete branches.

1. Click the **Publish changes** button next to the **dev** branch.



2. In the web browser go to **Azure DevOps Services**. Navigate to **Repos** hub and click **Branches**.

The screenshot shows the Azure DevOps interface for the 'PartsUnlimited' project. At the top, there's a green button with a white 'P' and the text 'PartsUnlimited'. To its right is a red '+' sign. Below this, there are several navigation links:

- Overview** (blue icon)
- Boards** (green icon)
- Repos** (orange icon)
- Files** (blue icon)
- Commits** (light blue icon)
- Pushes** (light blue icon)
- Branches** (orange icon) - This link is highlighted with an orange rounded rectangle.
- Tags** (blue icon)
- Pull requests** (light blue icon)

3. You should see the newly pushed **dev** branch. Click on **More actions** and select the **Delete branch** option to delete it. Confirm the delete.

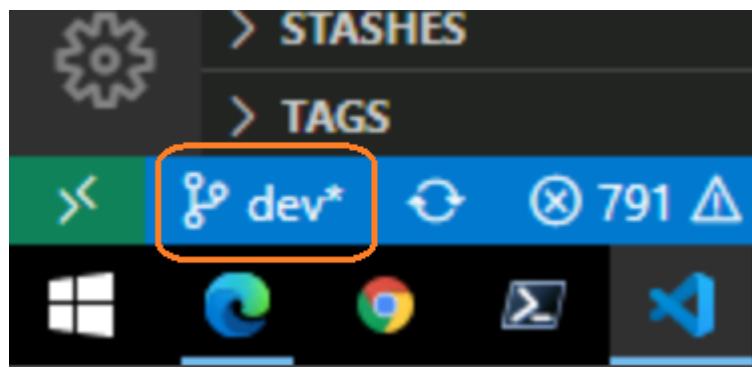
The screenshot shows a list of branches in a GitHub repository. The branches listed are:

- dev (highlighted with an orange box)
- e2e-complete
- FixSearchFunctionality
- IntelliTest
- LiveUnitTesting
- master (Default Compare)
- PerfAndLoadTesting

A context menu is open on the right side of the screen, with the "Delete branch" option highlighted by an orange box. Other options in the menu include:

- New branch
- New pull request
- Delete branch (highlighted)
- View files
- View history
- Compare branches
- Set as compare branch

4. Return to **Visual Studio Code** and click the **dev** branch.

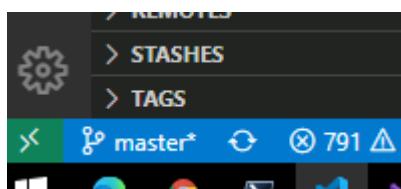
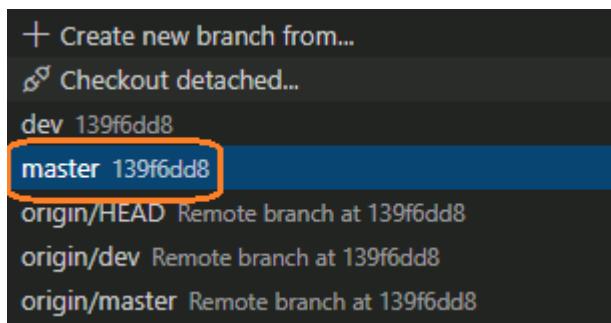


5. Note that there are two dev branches listed. The local **dev** branch is there because it's not deleted when the remote branch is deleted. The remote **origin/dev** is there because it hasn't been pruned.

The screenshot shows the Visual Studio Code interface with a dropdown menu open in the bottom-left corner. The menu is titled "Select a ref to checkout" and lists the following options:

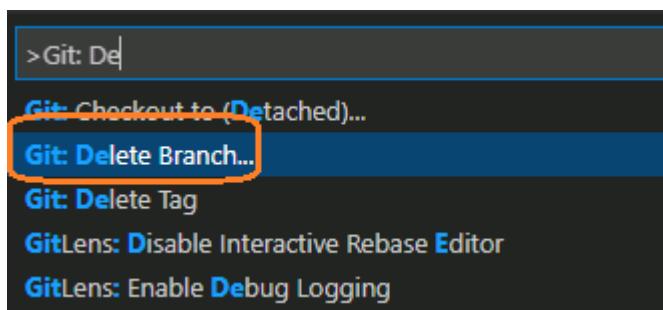
- + Create new branch...
- + Create new branch from...
- Checkout detached...
- dev 139f6dd8 (highlighted with an orange box)
- master 139f6dd8
- origin/HEAD Remote branch at 139f6dd8
- { origin/dev Remote branch at 139f6dd8 (highlighted with an orange box)
- origin/master Remote branch at 139f6dd8
- origin/FixSearchFunctionality Remote branch at 662d9fd
- origin/AddTerms&Conditions Remote branch at 9da57edd
- origin/DependencyValidation Remote branch at 1ca13fb2
- origin/e2e-complete Remote branch at a24bb398

6. Select the **master** branch to check it out. Now you should see **master** branch checked out at the bottom-left corner of Visual Studio Code.

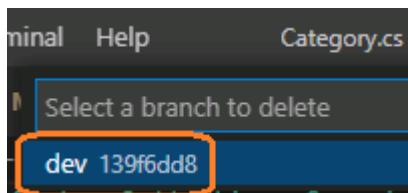


7. Press **Ctrl+Shift+P** to open the **Command Palette**.

8. Start typing **Git: Delete** and select **Git: Delete Branch...** when it becomes visible.

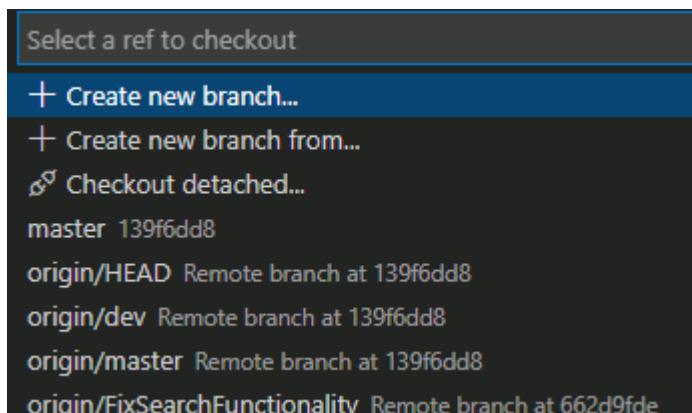


9. There is only one local branch (**dev**) to delete, so select this **dev** branch.



10. Click the **master** branch from bottom-left corner of Visual Studio Code.

Note that the local **dev** branch is now gone, but the remote **origin/dev** is still showing.

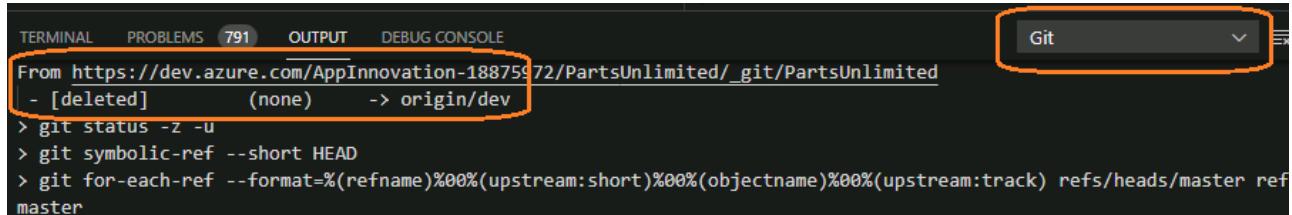


11. Press **Ctrl+Shift+P** to open the **Command Palette**.

12. Start typing **Git: Fetch** and select **Git: Fetch (Prune)** when it becomes visible.

This command will update the original branches in the local snapshot and delete those that are no longer in the remote repository.

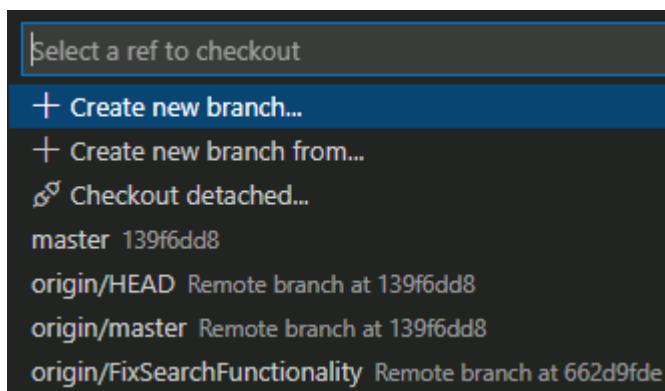
13. You can check the **Output** tab of the Visual Studio Code to view the Git output of the Git: Fetch (Prune) command. Make sure you have selected **Git** option from the dropdown to see the Git output.



The screenshot shows the VS Code interface with the 'OUTPUT' tab selected. A dropdown menu is open, with 'Git' highlighted and outlined in orange. The terminal output shows the results of a 'git fetch' command:

```
From https://dev.azure.com/AppInnovation-18875972/PartsUnlimited/_git/PartsUnlimited
- [deleted]      (none)    -> origin/dev
> git status -z -u
> git symbolic-ref --short HEAD
> git for-each-ref --format=%(refname)%00%(upstream:short)%00%(objectname)%00%(upstream:track) refs/heads/master ref
master
```

14. Click the **master** branch. The **origin/dev** branch should no longer be in the list.



Module 3: Azure Repos with Git, Lab 1: Getting Started with Git, Exercise 6: Managing Branches from Azure DevOps Services

Exercise 6: Managing Branches from Azure DevOps Services

Objectives

You can manage the work in your Azure DevOps Git repo from the Branches view in web portal. You can also customize the view to track the branches you care most about so you can stay on top of changes made by your team.

Locking is ideal for preventing new changes that might conflict with an important merge or to place a branch into a read-only state. Alternatively, you can use branch policies and pull requests instead of locking if you just want to ensure that changes in a branch are reviewed before they are merged.

Locking does not prevent cloning of a repo or fetching updates made in the branch into your local repo. If you lock a branch, share with your team the reason why and make sure they know what to do to work with the branch after it is unlocked.

Prerequisites

- Complete [Exercise 2](#) .

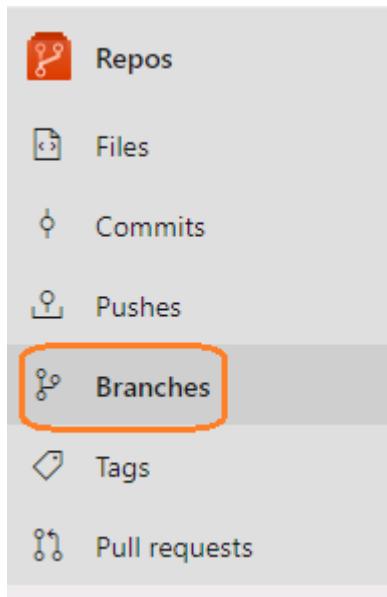
Tasks

- [Task 1: Creating a New Branch](#)
- [Task 2: Restoring a Branch](#)
- [Task 3: Locking a Branch](#)

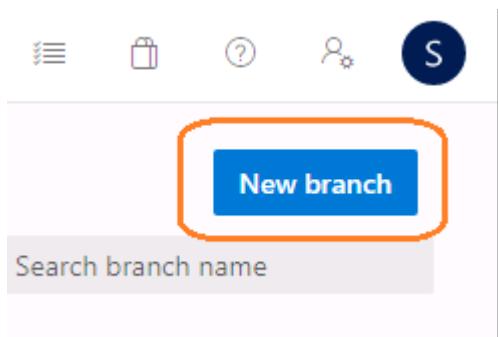
Module 3: Azure Repos with Git, Lab 1: Getting Started with Git, Exercise 6: Managing Branches from Azure DevOps Services

Task 1 Creating a New Branch

1. Switch to the **Azure DevOps Services** in the web browser.
2. Navigate to **Repos** hub and click on **Branches**



3. Click on the **New branch** from top-right corner of the page.



4. Name the new branch as **release**. Use the **Work items to link** dropdown to select one of the work items to link to this new branch. Click **Create** to create a branch.

Create a branch

Name *

Based on

 ▼Work items to link 1[Clear all](#) ▼

Task 50: Add Terms and Conditions for Sale & Sup...

Updated Saturday, ● To Do

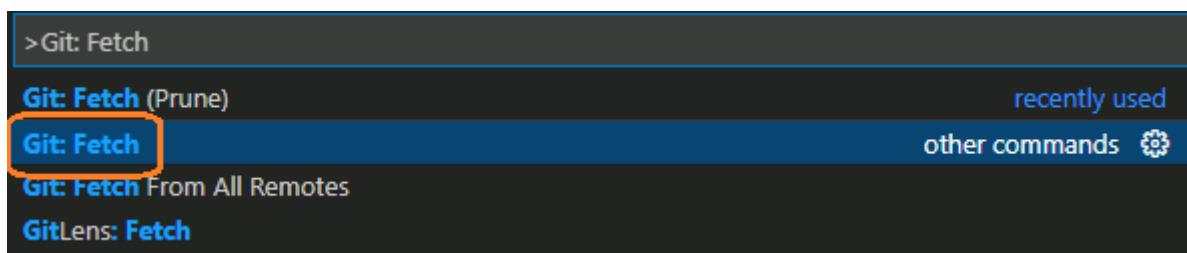
[Cancel](#)[Create](#)

5. After the branch has been created, it will be available in the list.

6. Return to **Visual Studio Code**.

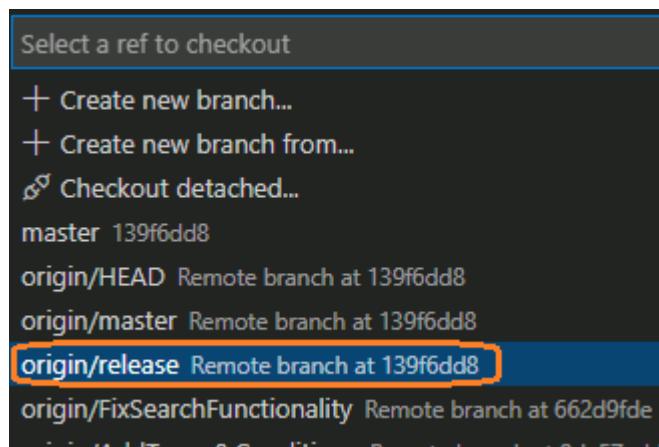
7. Press **Ctrl+Shift+P** to open the **Command Palette**.

8. Start typing **Git: Fetch** and select **Git: Fetch** when it becomes visible. This command will update the origin branches in the local snapshot.



9. Click the **master** branch from the bottom-left corner of Visual Studio Code.

10. In the list of branches you should now see **origin/release**. Click **origin/release** and this will create a new local branch called **release** and check it out.



Module 3: Azure Repos with Git, Lab 1: Getting Started with Git, Exercise 6: Managing Branches from Azure DevOps Services

Task 2: Restoring a Branch

1. Remember that you deleted the **dev** branch earlier from Azure DevOps Services? You can restore this branch, as long as you remember the exact name of the branch.
2. Navigate to **Azure DevOps Services** and **Repos** hub for the PartsUnlimited project. Click on **Branches**. In the **Search branch name**, enter **dev**.

The screenshot shows the 'Branches' page in the Azure DevOps 'Repos' hub. At the top, there is a search bar with the text 'dev' entered, which is highlighted with an orange rectangle. To the right of the search bar is a blue button labeled 'New branch'. Below the search bar, there is a decorative illustration of a person standing on a small island with a dog, looking through a telescope at the sky. A message 'No active branches found' is displayed. In the bottom section, there is a heading 'Deleted branches' followed by a table. The table has three columns: 'Branch', 'Deleted by', and 'Deleted on'. There is one row in the table, showing a branch named 'dev' deleted by 'Student1-19052640' 14m ago. The 'Branch' column for the deleted branch is also highlighted with an orange rectangle.

Branch	Deleted by	Deleted on
dev	Student1-19052640	14m ago

You can search deleted branch by searching the exact branch name only. A deleted Git branch can be restored at any time, regardless of when it was deleted.

3. Click on **More options** for the deleted **dev** branch and click **Restore branch** as shown below.

The screenshot shows the 'Deleted branches' table again. The 'dev' branch is listed, and the 'More options' menu icon (three vertical dots) is highlighted with an orange rectangle. The 'Restore branch' button in the same row is also highlighted with an orange rectangle.

Branch	Deleted by	Deleted on	More options
dev	Student1-19052640	18m ago	[More options]

Module 3: Azure Repos with Git, Lab 1: Getting Started with Git, Exercise 6: Managing Branches from Azure DevOps Services

Task 3: Locking a Branch

1. In the web browser, navigate to **Azure DevOps Services**. Navigate to **Repos | Branches**.
2. Select **More options** from the **master** branch and click on **Lock**

The screenshot shows a list of branches in a repository. The 'master' branch is highlighted with a red box. A context menu is open over the 'master' branch, with the 'Lock' option highlighted by a red box. Other options in the menu include 'Set as default branch', 'Branch policies', and 'Branch security'. The 'master' branch card shows it is the Default branch.

Branch	Last Commit	Author	Date	Commits
e2e-complete	a24bb3		Fe...	5 3
FixSearchFunctionality	662d9f		M...	5 1
IntelliTest	816d94		Ja...	5 0
LiveUnitTesting	d3bb00		Fe...	5 2
master	139f6d	S	Tu...	5 1
PerfAndLoadTesting	96135f		Fe...	5 1

3. The master branch is now locked.

The screenshot shows the same list of branches. The 'master' branch is now marked with a lock icon (a padlock with a slash) in its status column, indicating it is locked. The other branches (e2e-complete, FixSearchFunctionality, IntelliTest, LiveUnitTesting, and PerfAndLoadTesting) are shown without locks.

Branch	Last Commit	Author	Date	Commits
e2e-complete	d3bb00		Fe...	5 3
FixSearchFunctionality	139f6d	S	Tu...	5 1
IntelliTest	96135f		Fe...	5 1

4. Now **Unlock** the branch using the same process.

Module 3: Azure Repos with Git, Lab 1: Getting Started with Git, Exercise 7: Working with Pull Requests in Azure Repos

Exercise 7: Working with Pull Requests in Azure Repos

Objectives

Pull requests let your team provide feedback on changes in feature branches before merging the code into the master branch. Reviewers can step through the proposed changes, leave comments, and vote to approve or reject the code.

Prerequisites

- Complete [Exercise 5](#).

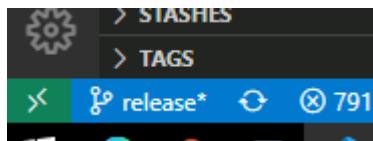
Tasks

- [Task 1: Creating a New Pull Request from Visual Studio Code](#)
- [Task 2: Managing Pull Requests](#)
- [Task 3: Tagging a Release](#)

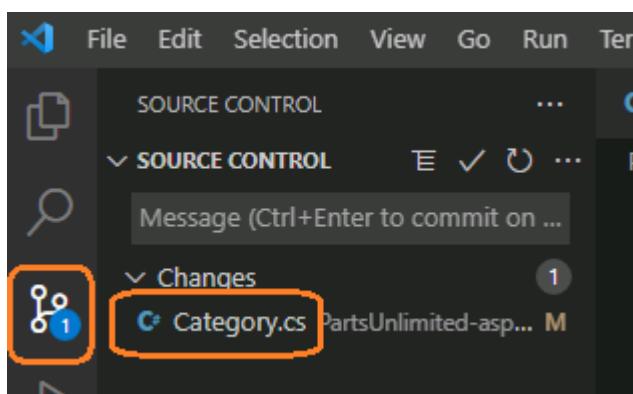
Module 3: Azure Repos with Git, Lab 1: Getting Started with Git, Exercise 7: Working with Pull Requests in Azure Repos

Task 1: Creating a New Pull Request from Visual Studio Code

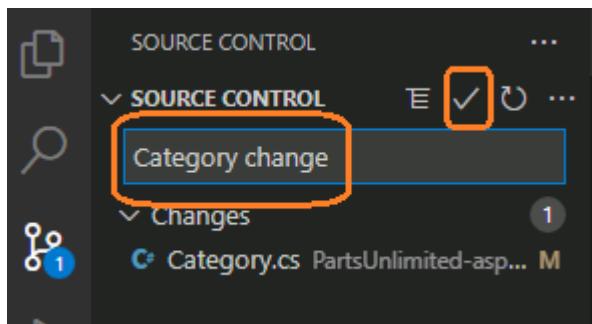
1. Return to **Visual Studio Code** and confirm that you have **release** branch selected.



2. Select the **Source Control** tab. It should recognize that you have uncommitted changes to **Category.cs**.



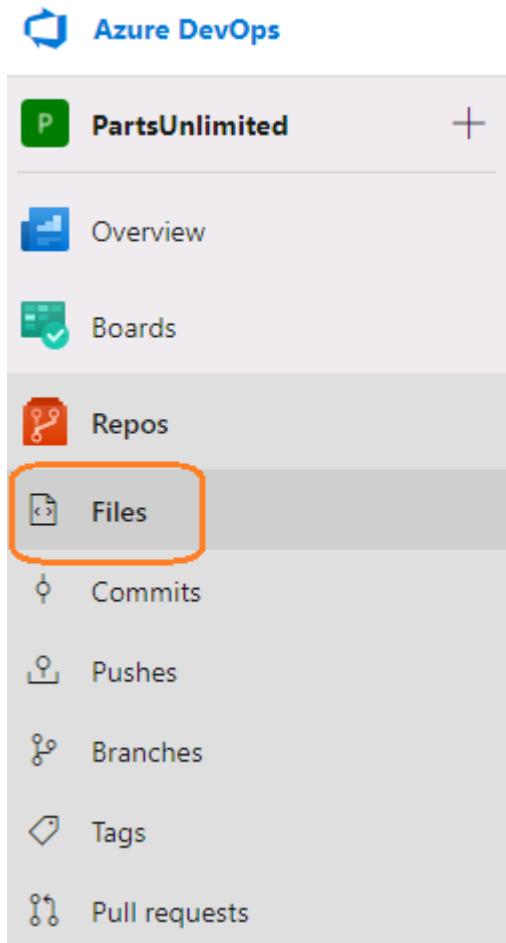
3. Enter a **commit message Category change** and press **Ctrl+Enter** to commit the change to the local release branch. You can also commit by clicking the **commit** button.



4. Click the **Synchronize Changes** button at the bottom-left corner of the VS Code to push the commit to the remote repository.



5. Navigate back to the web browser with Azure DevOps Services open. Click on the **Files** tab from **Repos**.



6. Click on the **Create a pull request** button in the information message to create a pull request.

A screenshot of the 'Files' page in Azure DevOps. At the top, there are buttons for 'Set up build', 'Clone', and more. Below that, tabs for 'Contents' and 'History' are visible. A message box contains the text 'You updated 🏠 release 3m ago' with an info icon. To the right of the message is a blue button labeled 'Create a pull request', which is also highlighted with an orange rectangular box. Below the message, there's a table header with columns for 'Name ↑', 'Last change', and 'Commits'.

7. You can customize the Pull Request details and some of it may be required based on policy. If there is no linked work item showing up under **Work items to link**, click on the dropdown and select any work item. Click **Create**.

New pull request

From release into master

Overview Files 1 Commits 1

Title
Category change

Description
Category change

Markdown supported Drag & drop, paste, or select files to insert.

Link work items.

Category change

Reviewers Add required reviewers

Search users and groups to add as reviewers

Work items to link 1 Clear all

Search work items by ID or title

Task 50: Add Terms and Conditions for Sale & Support in Layout.cshtml Page
Updated 8m ago, To Do

Tags

Create

Module 3: Azure Repos with Git, Lab 1: Getting Started with Git, Exercise 7: Working with Pull Requests in Azure Repos

Task 2: Managing Pull Requests

1. The **Overview** tab of the Pull Request contains all of the key information specified in the creation form, as well as options to approve and complete the request.

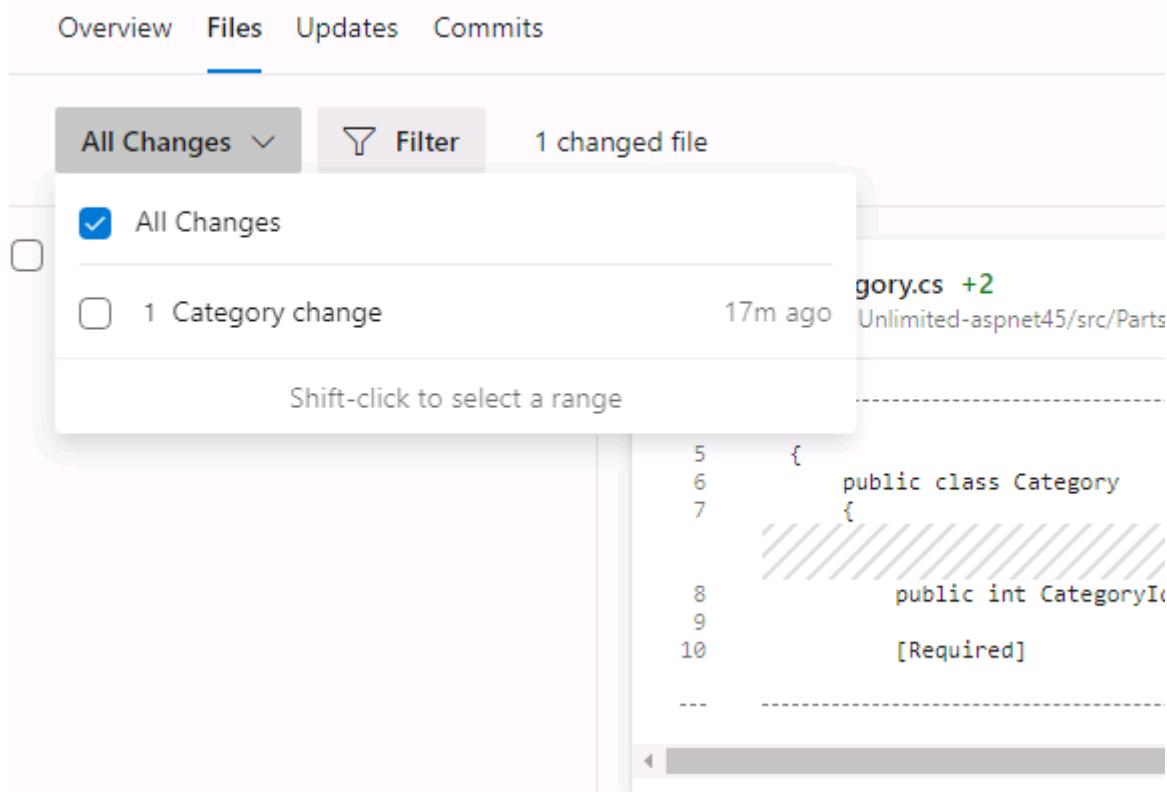
The screenshot shows the 'Category change' pull request in the 'Overview' tab. It includes sections for 'No merge conflicts', 'Reviewers' (both required and optional), 'Tags', and 'Work items'. Activity logs show a comment from 'Student1-18875972' and a task update from 'Task 50: Add Terms and Conditions ...'.

2. Select the **Files** tab of the Pull Request to review the files involved in the commit.

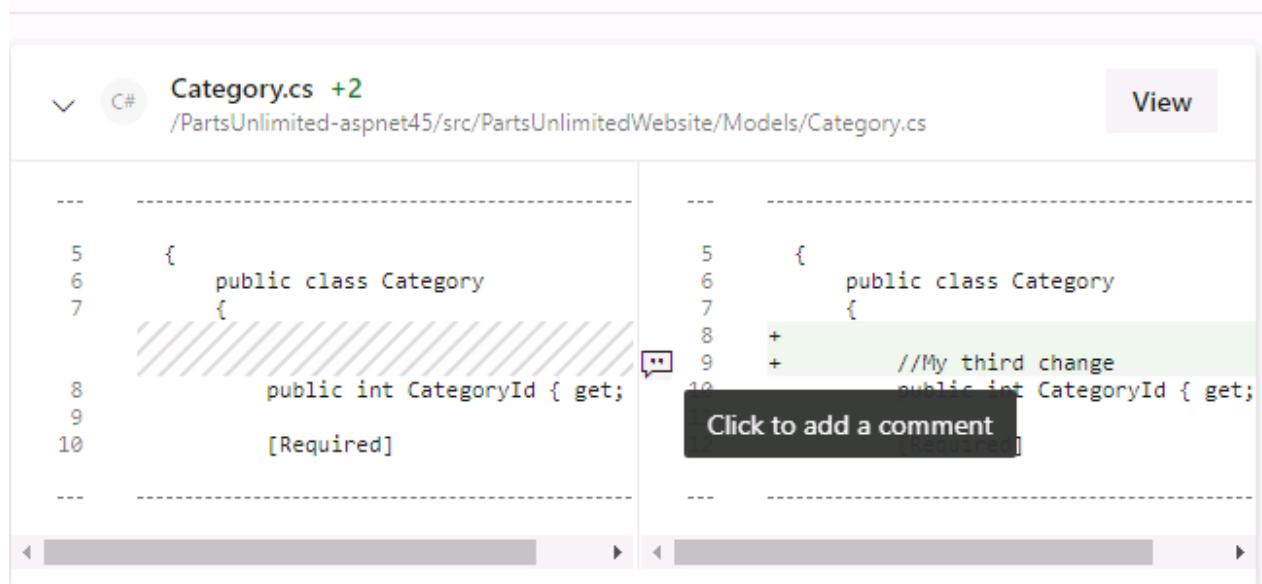
You might see slightly different view of files. Make sure you have selected **Side-by-side** from the dropdown on the top-right corner of the Files sections of the Pull Request.

The screenshot shows the 'Category change' pull request in the 'Files' tab. It displays a 'Side-by-side' comparison of the 'Category.cs' file between two commits. The left pane shows the original code, and the right pane shows the modified code with a green highlight for the addition of a new line.

3. Note that you can select a specific changes from the dropdown if you like.



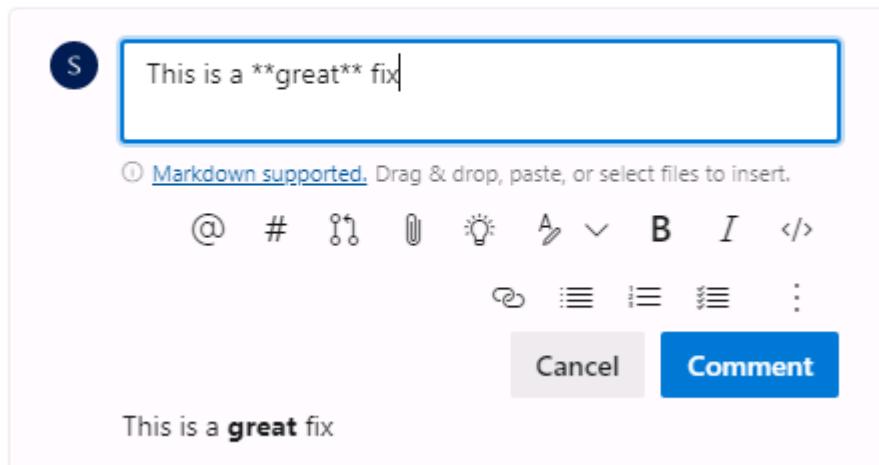
4. You can add a comment in the file that is displayed in the Pull Request. Click on **Click to add comment**.



5. Enter a comment using markdown **This is a *great* fix** and click **Comment** to save it. Close the **Discussion** pane.

Note the live preview of your comment before you commit to it.

Discussion



6. The new comment is tracked as part of the pull request. It's expected that every comment will be resolved before a pull request will be completed, so this convenient marker lets you know if there's anything else that needs to be reviewed.

Category change

Active | 6 | S Student1-18875972 release into master 0/1 comments resolved

7. The comment is also placed in line with the code. **Click** on the comment to start the discussions about lines and sections of code within their proper context. Click **Resolve** to complete the discussion and **close** the Discussion pane.

The screenshot shows a code editor with a pull request open. The file is Category.cs, located at /PartsUnlimited-aspnet45/src/PartsUnlimitedWebsite/Models/Category.cs. The code is as follows:

```
5  public class Category
6  {
7      [Required]
8  }
```

A comment from Student1-18875972 is highlighted with a callout bubble: "This is a **great** fix".

Discussion

A comment card from Student1-18875972 posted 7m ago. The comment text is "This is a **great** fix". There is a "Resolve" button at the bottom right of the card.

8. Note that this now updates the tracking as well.

The screenshot shows a "Category change" pull request. The status is "Active". It has 16 comments from Student1-18875972. A callout bubble highlights the text "All comments resolved". Below the pull request details are tabs for "Overview", "Files", "Updates", and "Commits".

9. Select the **Updates** tab. This contains details on the updates in the branch.

10. Select the **Commits** tab, where you can review the commits made to the branch.

11. Since everything seems to be in order, **Approve** the pull request. As the approvers have signed off, **Complete** the pull request.

The screenshot shows a pull request interface. At the top right, there are buttons for 'Approve' and 'Complete'. The 'Complete' button is highlighted with an orange box. Below the buttons, there are tabs for 'Overview', 'Files', 'Updates', and 'Commits', with 'Commits' being the active tab. A commit message is visible: 'Category change' by 'Student1-18875972' at 'Today at 12:19 PM'.

12. You can accept the default options in the **Complete pull request** dialog. The first option is to complete the work items linked to the branch being merged. The second checkbox indicates that the release branch will be deleted after merging. Click **Complete merge**.

Please see [Complete, abandon, or revert pull requests - Azure Repos](#) for different Merge Type options.

The screenshot shows the 'Complete pull request' dialog. It starts with a 'Merge type' dropdown set to 'Merge (no fast forward)'. Below it is a diagram illustrating a merge operation. The 'Post-completion options' section contains three checkboxes: 'Complete associated work items after merging' (checked), 'Delete release after merging' (checked), and 'Customize merge commit message' (unchecked). At the bottom are 'Cancel' and 'Complete merge' buttons, with 'Complete merge' highlighted with an orange box.

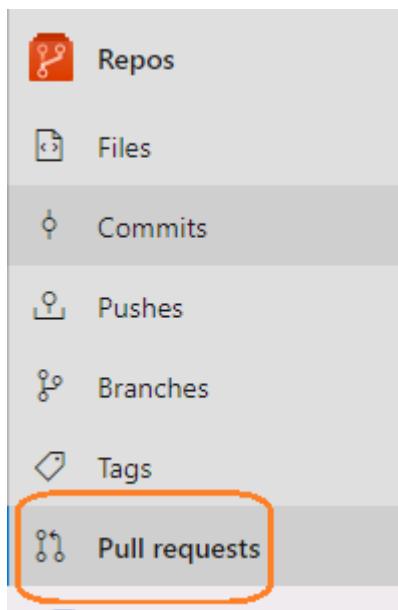
13. When the merge completes, the pull request should be marked as *Completed*.

Category change

Completed !6 S Student1-18875972 release into master All comments resolved

Overview Files Updates **Commits**

14. Click on the **Pull requests** home.



15. Select the **Completed** tab and click the pull request as though you were visiting it fresh.

Pull requests

Mine Active Completed Abandoned

Pull Request ID

Created by

S

Category change

Student1-18875972 request !6 into master

16. Click on the pull request to review it. You can *Cherry-pick* or *Revert* if needed.

Cherry-picking is the process of selecting specific commits from one branch to apply to another, conceptually similar to a copy/paste operation.

Category change

Completed !6 S Student1-18875972 release into master All comments resolved

Overview Files Updates Commits

Student1-18875972 completed this pull request 8m ago

Cherry-pick Revert

Merged PR 6: Category change
02ddc0cc S Student1-18875972 Today at 1:58 PM

Show details

No merge conflicts Last checked 8m ago

Description

Category change

Reviewers

Required

No required reviewers

Optional

✓ S Student1-18875972 Approved

Tags

No tags

Work items

+ Task 50: Add Terms and Conditions ...

17. Under Work Items, click one of the linked work items.

Work items +

Task 50: Add Terms and Conditions ... X

Updated 8m ago, ● Done

18. Note that the work item has now been marked as Done.

TASK 50

50 Add Terms and Conditions for Sale & Support in Layout.cshtml Page

Unassigned 1 comment Add tag

State	● Done	Area	PartsUnlimited
Reason	Work finished	Iteration	PartsUnlimited\Sprint 4

19. Under the **Development** tab, you can see the commit and pull request have been associated with the work item.

Development

 Add link

 S Category change

Created 15 minutes ago, ✓ Completed

 S Added Terms and Conditions for Sale in ...

Created Wednesday, Active

 S adfb0d3b Merged PR 3: Category change

Created 6 minutes ago

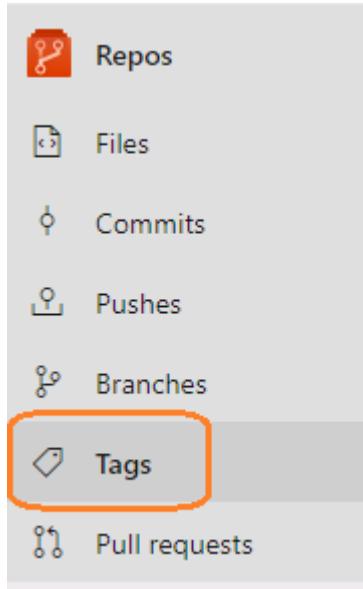
Related Work

 Add link

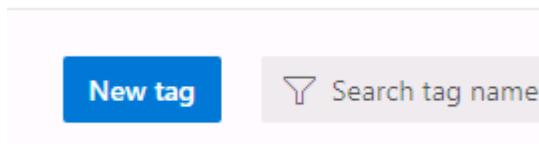
Module 3: Azure Repos with Git, Lab 1: Getting Started with Git, Exercise 7: Working with Pull Requests in Azure Repos

Task 3: Tagging a Release

1. Let's assume that this version of the site is exactly what's needed for v1.1. You can add a tag mentioning this. Navigate to **Repos** hub and click on **Tags** tab.



2. Click on **New Tag** from the top-right corner of the page.



3. Enter a **Name** as **v1.1** and a **Description** as **Great release!**. Click **Create**.

Create a tag

Name *

 v1.1

Based on

master ▼

Description *

Great release!

Cancel

Create

4. You have now tagged the project at this release. You could tag commits for a variety of reasons, and Azure DevOps offers the flexibility to edit and delete them, as well as manage their permissions.

Module 3: Azure Repos with Git, Lab 1: Getting Started with Git, Exercise 8: Managing Repositories in Azure Repos

Exercise 8: Managing Repositories in Azure Repos

Objectives

In this exercise you will learn to create and manage repositories in Azure DevOps Services.

Prerequisites

- Complete [Exercise 1](#).

Tasks

- [Task 1: Creating a New Repo from Azure DevOps Services](#)
- [Task 2: Deleting and Renaming Git Repos](#)
- [Task 3: Managing Repository and Branch Policies](#)
- [Task 4: Forking Repos](#)
- [Task 5: Remove Branch Policies](#)

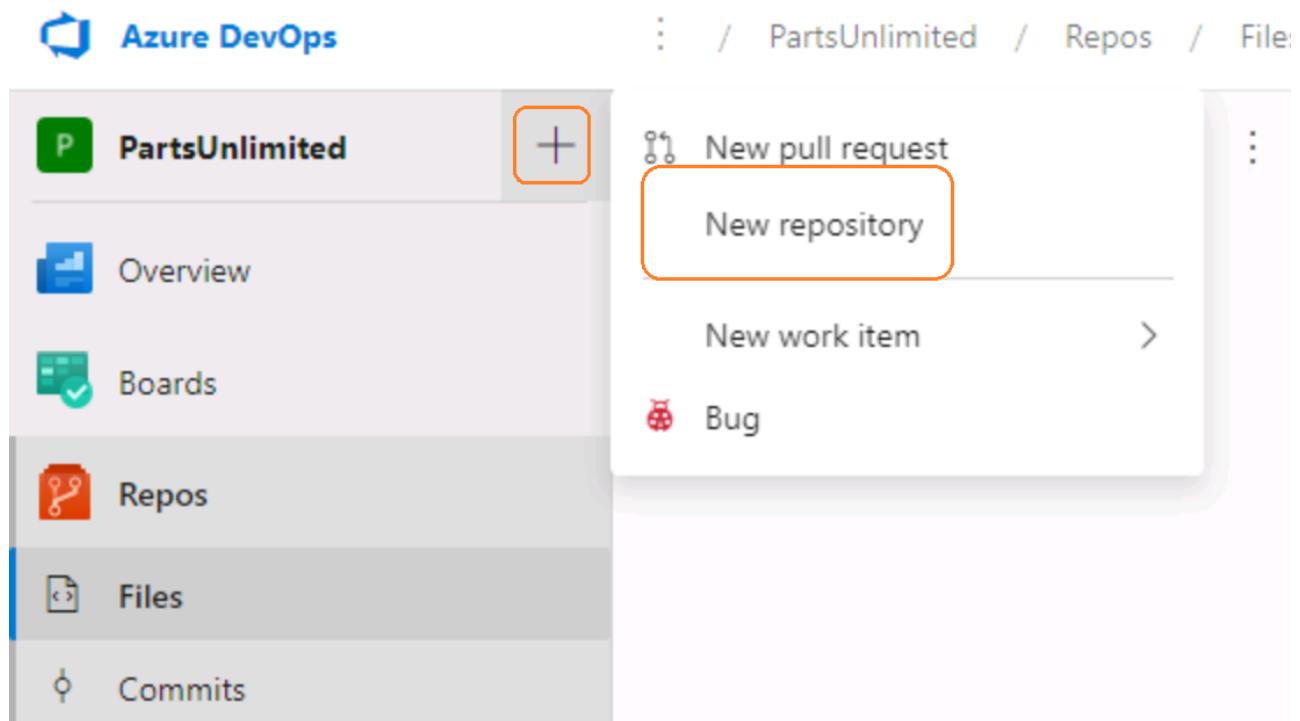
Module 3: Azure Repos with Git, Lab 1: Getting Started with Git, Exercise 8: Managing Repositories in Azure Repos

Task 1: Creating a New Repo from Azure DevOps Services

1. In the web browser, navigate to **Azure DevOps Services**. From the **PartsUnlimited** project **Add** dropdown, select **New repository**.

If you don't see the **New Repository** option in the dropdown, this maybe because you aren't in the context of **Repos** hub. The **Add** dropdown displays contextual options, meaning the options change based on whether you have are in **Repos** hub or **Boards** hub or **Pipelines** hub.

Alternatively, you can also select the dropdown from the **/ PartsUnlimited / Repos / Files / Partslimited** from the top-center and select **+ New repository**



2. Set the **Repository name** to **New Repo** and click **Create**.

Create a repository



Repository type



Repository name *

New Repo

Add a README

Add a .gitignore: None



Your repository will be initialized with a main branch.

Cancel

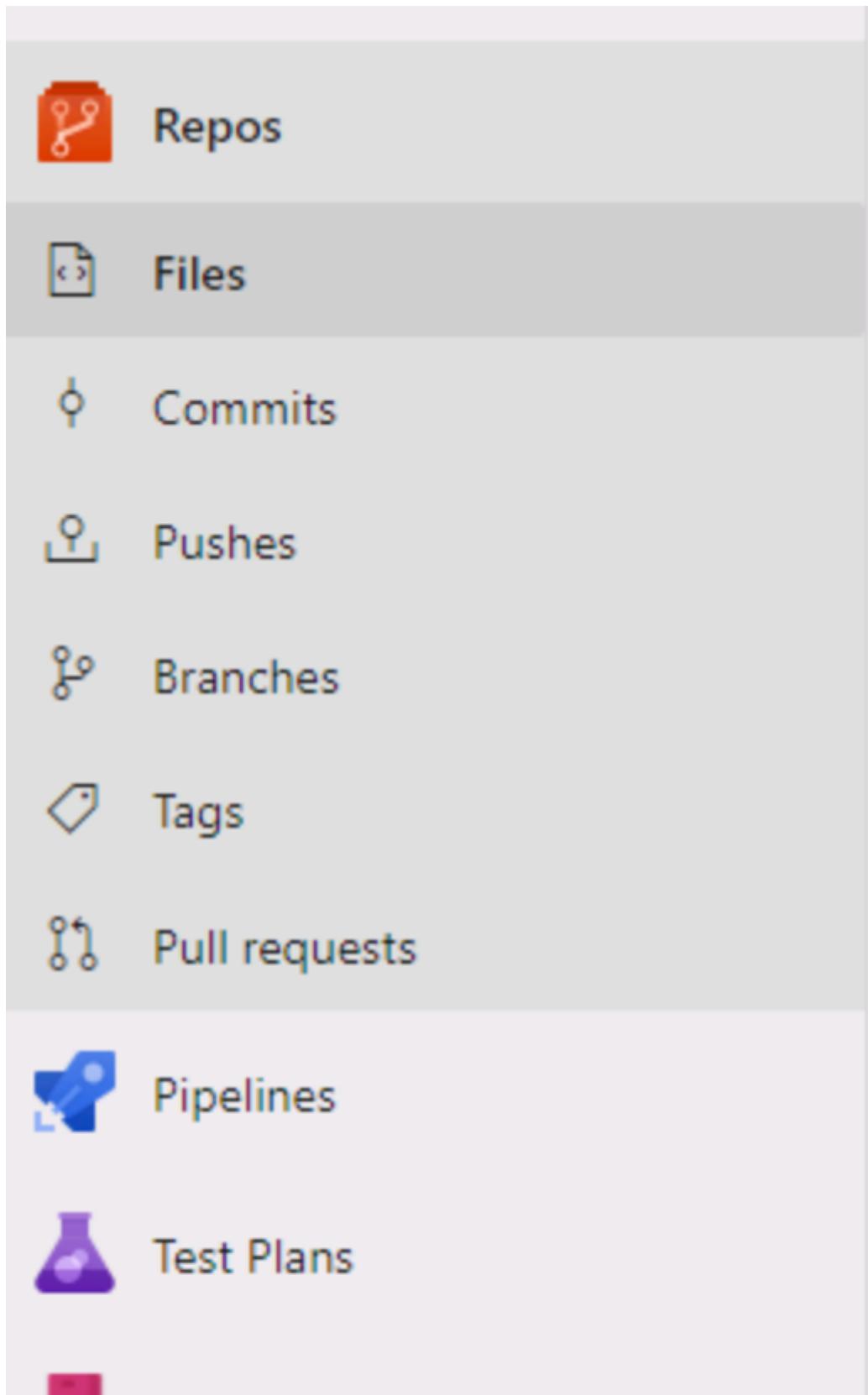
Create

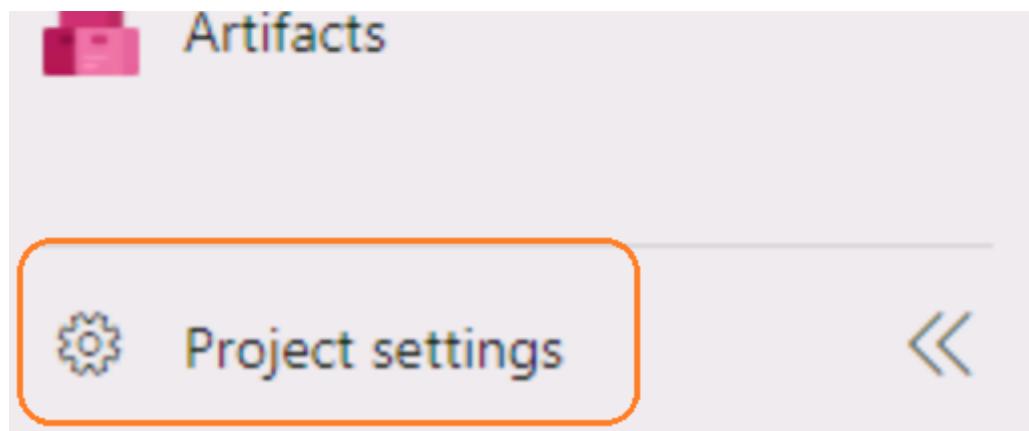
3. That's it. Your repo is ready. You can now clone it with Visual Studio Code or tool of your choice.

Module 3: Azure Repos with Git, Lab 1: Getting Started with Git, Exercise 8: Managing Repositories in Azure Repos

Task 2: Deleting and Renaming Git Repos

1. Sometimes you'll have a need to rename or delete a repo, which is just as easy. Open **Project settings** from the bottom-left corner of Azure DevOps Services.





2. Select **Repositories** under **Repos**.

A screenshot of the "Project Settings" menu in Azure DevOps. On the left is a vertical sidebar with icons for different sections: Project configuration (green P), Pipelines (plus sign), Agent pools (blue chart), Parallel jobs (orange gear), Settings (blue gear), Test management (checkmark), Release retention (document), Service connections (lightbulb), and XAML build services (purple flask). The "Repos" section is expanded, showing options like "Repositories" (which is highlighted with an orange box) and "Artifacts". The "Repositories" option has a sub-menu icon next to it.

3. Click on **More options** menu for the **New Repo**. From this menu, you can delete or rename the repository. Click on **Delete** to delete New Repo repository.

The screenshot shows the 'All Repositories' page. At the top right is a blue 'Create' button. Below it is a navigation bar with 'Repositories' (selected), 'Settings', 'Policies', and 'Security'. To the right is a search bar labeled 'Filter by keywords' with a magnifying glass icon. On the left, there's a list of repositories: 'New Repo' (selected) and 'PartsUnlimited'. On the right, a context menu is open for 'PartsUnlimited', showing options: 'Browse', 'Rename', and 'Delete'. The 'Delete' option is highlighted with an orange circle.

4. Enter the name **New Repo** to confirm the repo and click **Delete**.

Delete New Repo repository

This repository will be permanently deleted. This is a destructive operation. Please type the repository's name (New Repo) to confirm.



Module 3: Azure Repos with Git, Lab 1: Getting Started with Git, Exercise 8: Managing Repositories in Azure Repos

Task 3: Managing Repository and Branch Policies

1. Make sure you are under **Project settings | Repos | Repositories**. Select the **PartsUnlimited** repo. Like everything else in Azure DevOps Services, you can manage security to a great level of detail. Select the **Settings** tab.

This allows you to set some useful policies, such as whether or not you want to allow users to fork the repo, whether the work items are automatically linked, and so on.

The screenshot shows the 'PartsUnlimited' repository settings in Azure DevOps. On the left, there's a sidebar with 'All Repositories' and a filter for 'PartsUnlimited'. The main area has tabs for 'Settings', 'Policies', 'Security', and 'Approvals and checks'. The 'Settings' tab is selected. Under 'Advanced Security', the toggle is off. It includes a note about billing based on active committers and links to 'Estimate committers', 'View billing', and 'View alerts'. Below this is a 'Repository Settings' section with four policy options:

- Forks**: On, allowing users to create forks from this repository.
- Commit mention linking**: Off, automatically creating links for work items mentioned in commit comments.
- Commit mention work item resolution**: Off, allowing mentions in commit comments to close work items (e.g., "Fixes #123").
- Work item transition preferences**: On, remembering user preferences for completing work items with pull requests.

2. Click **Policies**. You can define a wide variety of policies for the repositories and branches to enforce.

Azure DevOps Services branch policies are very effective in enforcing a level of quality control in the repo.

For example, you can control pull requests by requiring a minimum number of reviewers, checking for linked work items, requiring that all comments have been resolved, and more. You can even require validation through a successful build and configure external approval services. If there are certain sections of code that require their own approvers to be included, you can include them here as well.

3. Scroll down to select **master** branch from **Branch Policies**.

The screenshot shows the 'Branch Policies' page for the 'PartsUnlimited' repository. The 'Policies' tab is active. A list of policies is displayed, with 'master Default' highlighted by a red box. Other policies listed include AddTerms&Conditions, CodedUITest, demo-start, DependencyValidation, e2e-complete, FixSearchFunctionality, IntelliTest, LiveUnitTesting, and PerfAndLoadTesting.

- Turn on **Require a minimum number of reviewers**. By default, this requires at least two reviewers to approve a pull request, and also requires (by default) that the original author is not one of them.

master

The screenshot shows the 'Branch Policies' configuration for the 'master' branch. The 'Require a minimum number of reviewers' toggle is turned 'On'. The minimum number of reviewers is set to 2. Below this, there are several optional checkboxes:

- Allow requestors to approve their own changes
- Prohibit the most recent pusher from approving their own changes
- Allow completion even if some reviewers vote to wait or reject
- When new changes are pushed:

- Scroll down and look for **Automatically included reviewers** section. Click + to **Add new reviewer policy**.

The screenshot shows the 'Automatically included reviewers' section in the Azure DevOps Services interface. On the left, there's a sidebar with a list of repository branches: master, Default, and Compare. The main area displays three sections: 'Build Validation' (0 policies), 'Status Checks' (0 policies), and 'Automatically included reviewers' (0 policies). The 'Automatically included reviewers' section includes a descriptive text about designating code reviewers for pull requests changing certain areas of code, a 'Filter' button, and a prominent '+ Add new reviewer policy' button, which is highlighted with a red box.

6. Under **Add new reviewer policy** enter following details:

- **Reviewers:** Add yourself as the reviewer. If asked to authenticate, enter your credentials.
- **Policy requirement: Required.** This ensures that above reviewer will be required to sign off on any changes proposed in the path mentioned below.
- **For pull request affecting these folders:** **/PartsUnlimited-aspnet45/src/PartsUnlimitedWebsite/Controllers/*.**
- Click **Save.**

Add new reviewer policy

Reviewers

 Student1-18875972 X +

Policy requirement

Optional

Required

For pull request affecting these folders

`artsUnlimited-aspnet45/src/PartsUnlimitedWebsite/Controllers/*` ⓘ

Leave blank to include the specified reviewers on all pull requests

Completion options

Allow requestors to approve their own changes

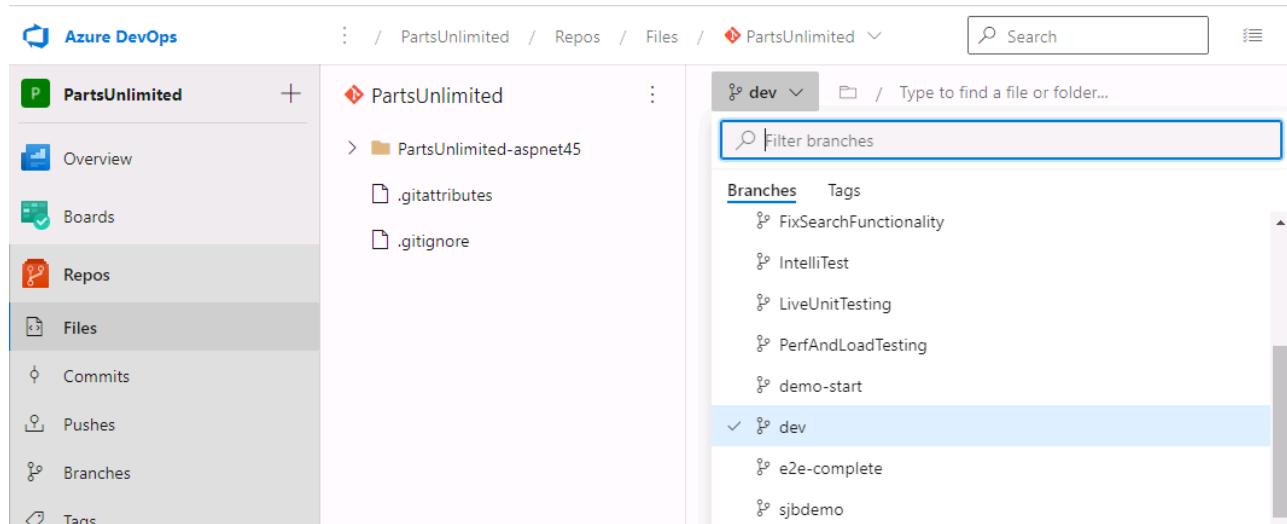
Activity feed message

Save Cancel

You will now test the **Automatically included reviewer** policy we implemented above.

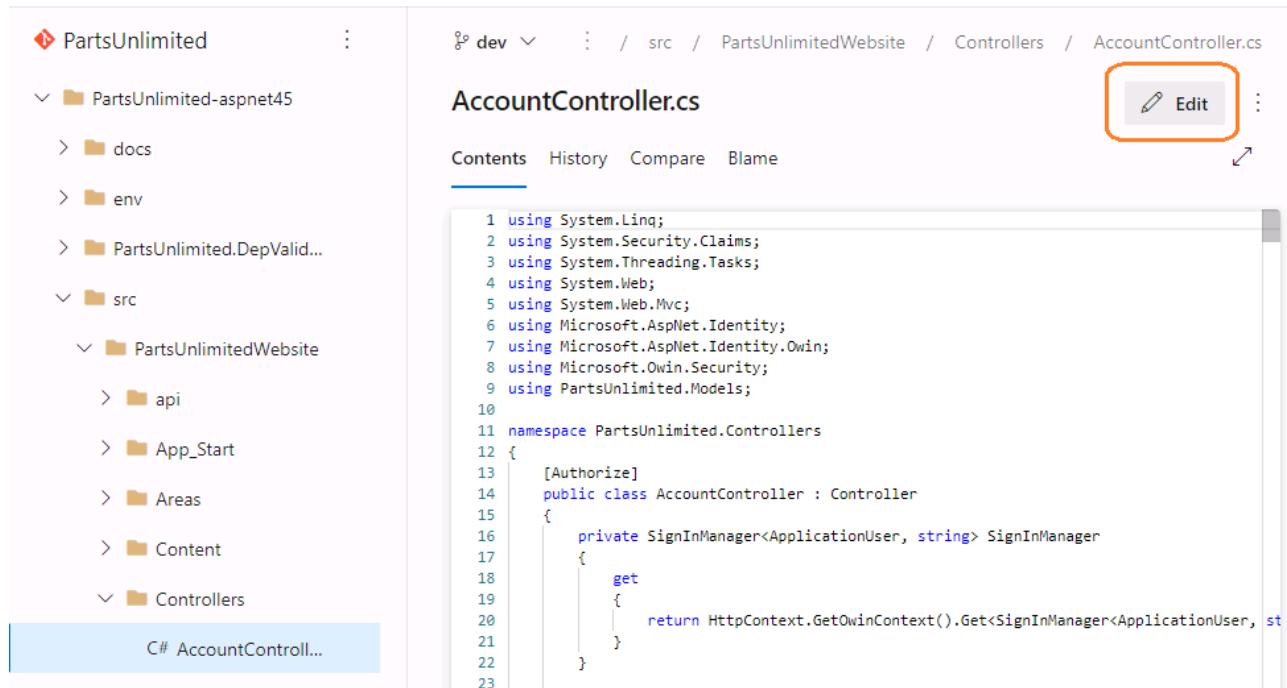
7. Navigate to **Repos | Files** and select **dev** branch.

If Repository "New Repo" not found message is displayed, click on **Switch to the default PartsUnlimited repository**.



8. Navigate to **PartsUnlimited**-

aspnet45/src/PartsUnlimitedWebsite/Controllers/AccountController.cs and click **Edit**.



9. Add a comment **//Testing automatic reviewer policy** to this **AccountController.cs** file and click **Commit**. Confirm the **Commit** with the default commit message.

The screenshot shows a code editor interface with the file path: dev / src / PartsUnlimitedWebsite / Controllers / AccountController.cs. The file content is as follows:

```
1 using System.Linq;
2 using System.Security.Claims;
3 using System.Threading.Tasks;
4 using System.Web;
5 using System.Web.Mvc;
6 using Microsoft.AspNet.Identity;
7 using Microsoft.AspNet.Identity.Owin;
8 using Microsoft.Owin.Security;
9 using PartsUnlimited.Models;
10
11 namespace PartsUnlimited.Controllers
12 {
13     //Testing automatic reviewer policy
14     [Authorize]
15     public class AccountController : Controller
16     {
```

The line //Testing automatic reviewer policy is highlighted with an orange box. At the top right, there are two buttons: "Commit" (highlighted with an orange box) and "Revert". Below the code editor, there are links for "Contents" and "Highlight changes".

10. Click on the **Create a pull request** to create a new pull request to merge this change to the master branch.

The screenshot shows a code editor interface with the file path: AccountController.cs. The status bar at the bottom left shows "Committed ca054ba6: Updated AccountController.cs". On the right side, there is a button labeled "Create a pull request" which is highlighted with an orange box.

11. Click **Create** to create a pull request with the default values.
12. As the new pull request gets created, notice that it automatically added you as the reviewer. It shows "**Required reviewers have not approved**" message based on the policy we set up.

Updated AccountController.cs

Active | 4 | Student1-19623872 dev into master

Approve | Set auto-complete | ...

Overview Files Updates Commits

Required reviewers have not approved

No merge conflicts Last checked 6m ago

Reviewers

Required

S Student1-19623872 No review yet

Add

Description

Updated AccountController.cs

Show everything (2) ▾

Tags

No tags

Work items

No work items

Add a comment...

S Student1-19623872 was added as a required reviewer for AccountController.cs. 6m ago

13. Click **Approve** to see the policy requirement being met. You will now see the other branch policy being enforced. You should see a message *At least 2 reviewers besides author must approve*.

Updated AccountController.cs

Active | 5 | Student1-19188957 dev into master

Approve | Set auto-comp

Overview Files Updates Commits

At least 2 reviewers besides author must approve

No merge conflicts Last checked 10m ago

Reviewers

Required

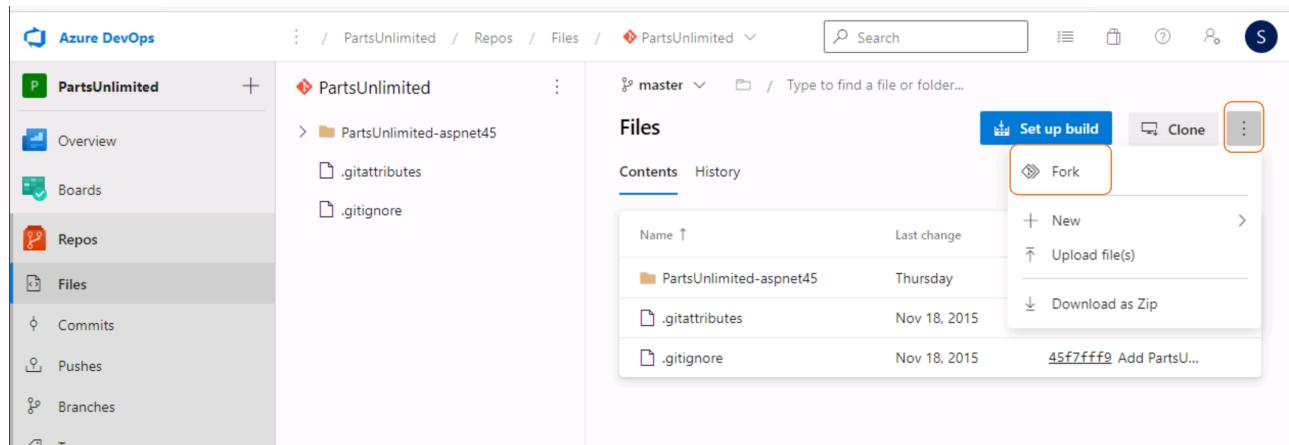
S Student1-19188957 Approved

Leave this pull request as is. You won't be able to complete the PR as you need two more reviewers to approve it.

Module 3: Azure Repos with Git, Lab 1: Getting Started with Git, Exercise 8: Managing Repositories in Azure Repos

Task 4: Forking Repos

1. Click on **Repos | Files**.
2. From the **More options** for the repository, from the top-right corner, click **Fork**.



Fork can be used by people outside the team to contribute changes to the project. For example, if someone wants to offer a bug fix, they would push it to a branch on this fork and request a pull. Someone from the team will monitor the pull requests and manage changes back through merges until they get to the primary project.

3. Set the **Repository name** to **PartsUnlimited.External.fork** and click **Fork**. Note that you could specify to include all branches, although only the default branch is usually forked.

Fork PartsUnlimited to...



Repository name

PartsUnlimited.External.fork

Project

PartsUnlimited



Branches to include:

- Only the default branch (master)
- All branches

Cancel

Fork

4. You can now work with the repo you forked.

Module 3: Azure Repos with Git, Lab 1: Getting Started with Git, Exercise 8: Managing Repositories in Azure Repos

Task 5: Remove Branch Policies

1. In Azure DevOps Services, open **Project settings** from the bottom-left corner.
2. Select **Repositories** under **Repos**.
3. Select the **PartsUnlimited** repo. Click **Policies**.

The screenshot shows the Azure DevOps Project Settings interface. On the left, the 'Project Settings' sidebar is visible with options like Team configuration, GitHub connections, Pipelines, Agent pools, Parallel jobs, Settings, Test management, Release retention, Service connections, and XAML build services. Below this, the 'Repos' section is expanded, showing 'Repositories' (which is selected and highlighted with an orange box) and 'Artifacts'. The main area shows the 'All Repositories' list with 'PartsUnlimited' selected (also highlighted with an orange box). To the right, the 'PartsUnlimited' repository details page is displayed, with the 'Policies' tab selected (also highlighted with an orange box). The 'Repository Policies' section contains four policy items, each with a toggle switch set to 'Off':

- Commit author email validation**: Block pushes with a commit author does not match the following pattern.
- File path validation**: Block pushes from introducing file p match the following patterns.
- Case enforcement**: Avoid case-sensitivity conflicts by bl that change name casing on files, fo branches, and tags. [Learn more](#)
- Reserved names**: Disallow names that introduce file eni

4. Scroll down and select the **master** branch from the Branch Policies section.

The screenshot shows the 'Branch Policies' page for the 'master' branch. At the top, there are tabs for 'Settings', 'Policies', and 'Security'. The 'Policies' tab is selected. Below the tabs, there's a search bar labeled 'Search branch name'. A list of policies is shown, with 'master' being the selected one, indicated by an orange border around its row.

Policy
AddTerms&Conditions
CodedUITest
demo-start
DependencyValidation
e2e-complete
FixSearchFunctionality
IntelliTest
LiveUnitTesting
master
PerfAndLoadTesting

5. **Turn off** Require a minimum number of reviewers.

The screenshot shows the 'Branch Policies' settings for the 'master' branch. It includes a note about deleting the branch if any policy is enabled. A large orange box highlights the toggle switch for 'Require a minimum number of reviewers', which is currently set to 'Off'.

Branch Policies
Note: If any required policy is enabled, this branch cannot be deleted and changes must be made via pull request.

Require a minimum number of reviewers
Require approval from a specified number of reviewers on pull requests.

6. Scroll down to **Automatically included reviewers** and click on **More options**. Click **Delete** to remove the automatically added reviewer from the earlier task.

The screenshot shows the 'Automatically included reviewers' section of a branch policy configuration. At the top, there's a header with a dropdown arrow, a 'Filter' button, and a '+' button. Below the header, a descriptive text explains the feature: 'Designate code reviewers to automatically include when pull requests change certain areas of code.' There are four tabs at the top: 'Enabled' (which is 'On'), 'Reviewers ↑', 'Path filter', and 'Inheritance'. Under the 'Reviewers ↑' tab, there's a list item for 'Student1...' with a status of 'Required' and a path filter of '/PartsUnlimi...'. To the right of this list item is a three-dot menu icon. Below the list item are two buttons: 'Edit/View' and 'Delete', with 'Delete' highlighted by an orange border.

Optionally, you can go back to the earlier pull request and complete it. Now that you have removed the branch policies, you should be able to complete the pull request without any issues.