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

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.

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

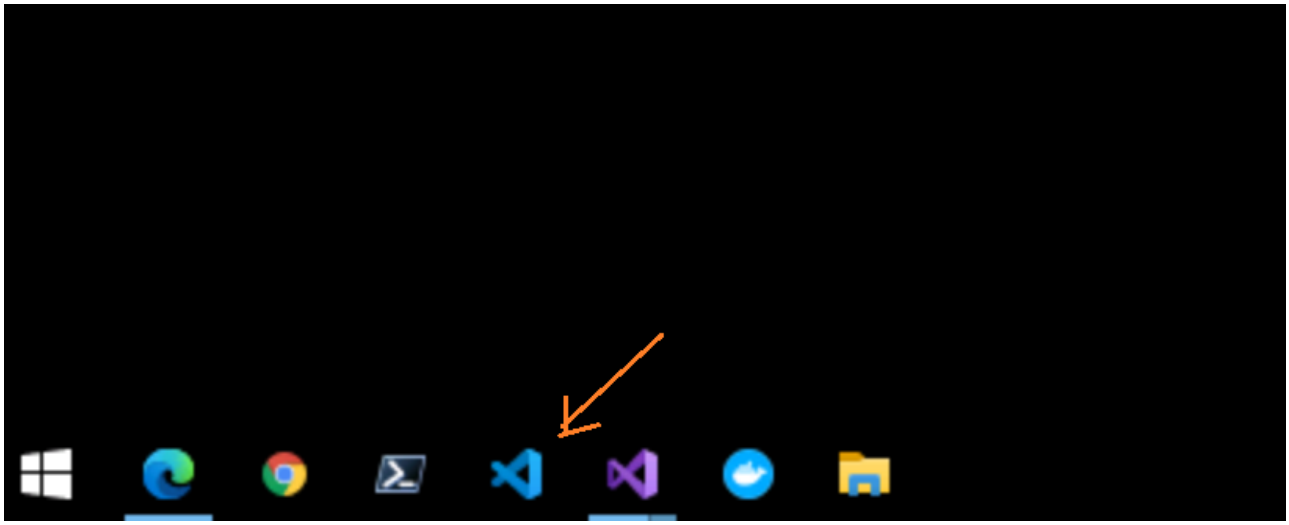
1. Task 1: Configuring Visual Studio Code

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

---

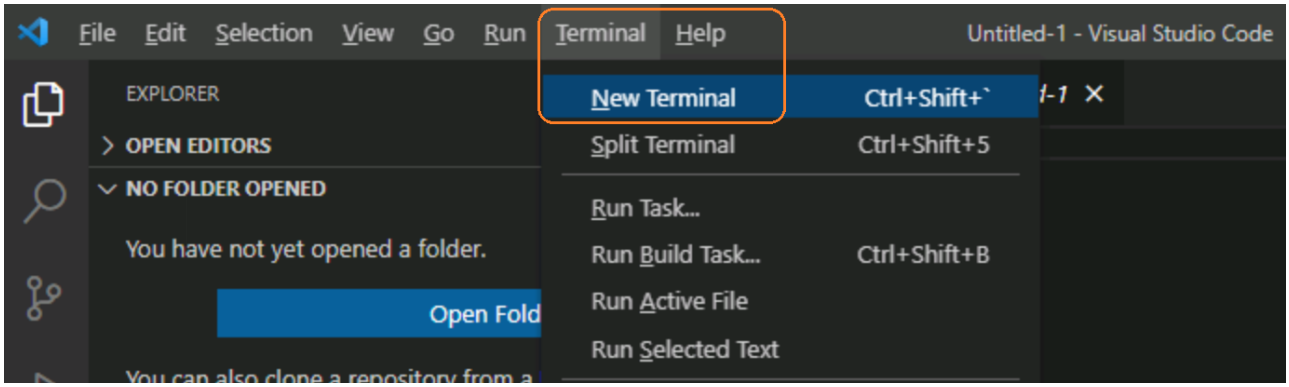
Task 1: Configuring Visual Studio Code

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.



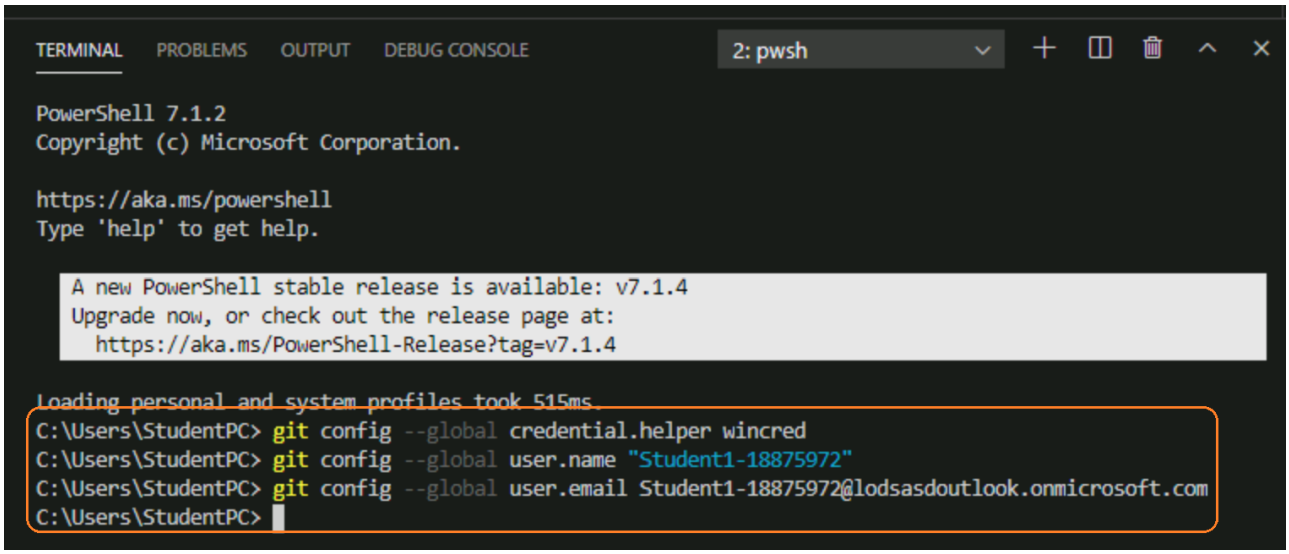
3. Execute the command below to configure a credential helper.

```
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

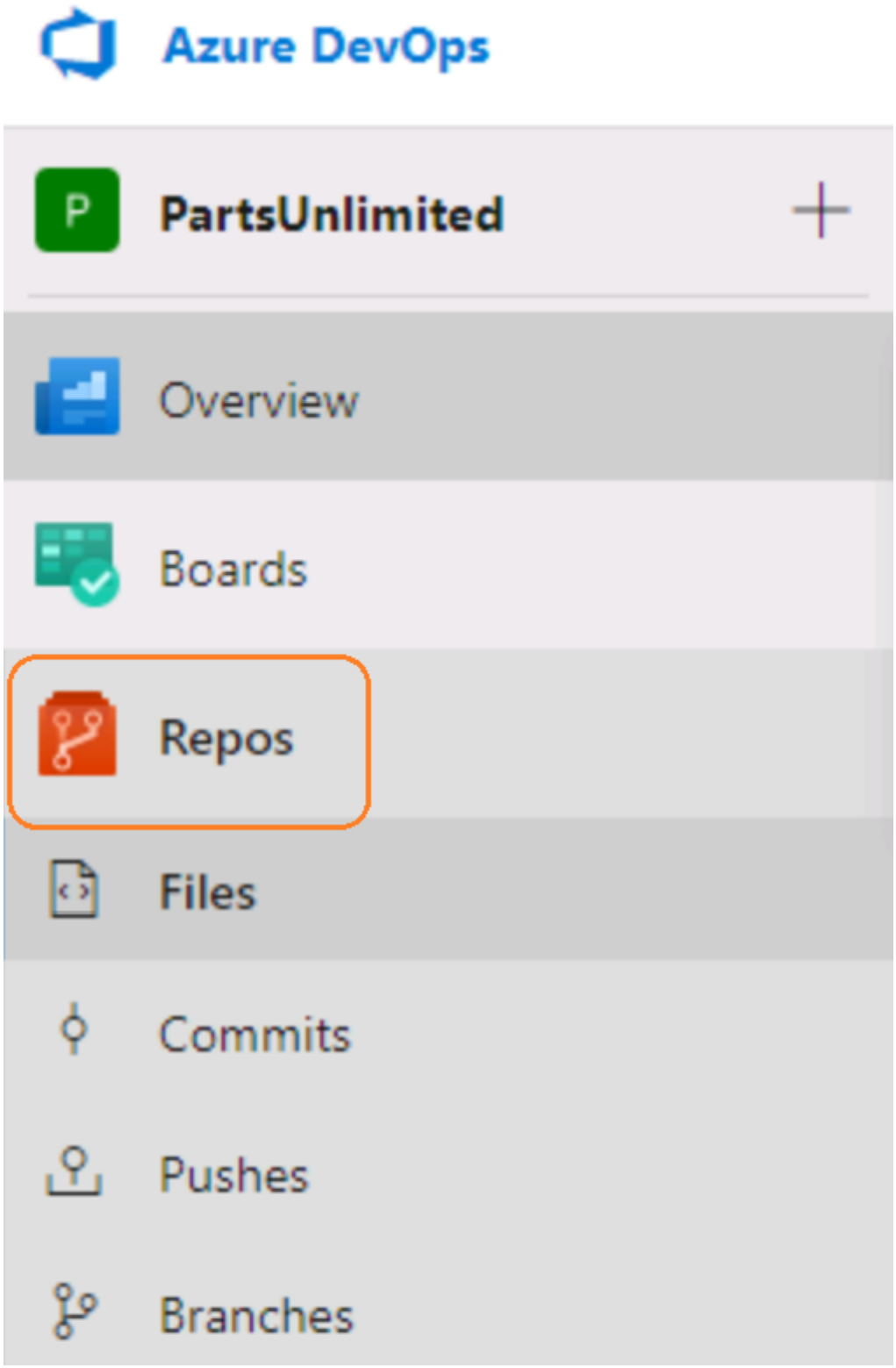
Module 3: **Azure Repos with Git**, **Lab 1: Getting Started with Git**, Exercise 2: Cloning Existing Azure Repository

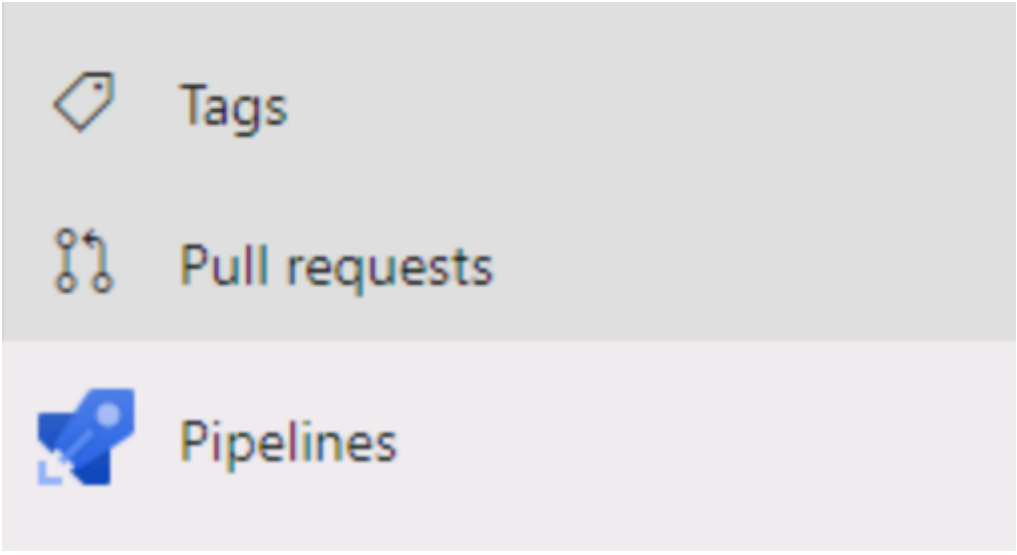
---

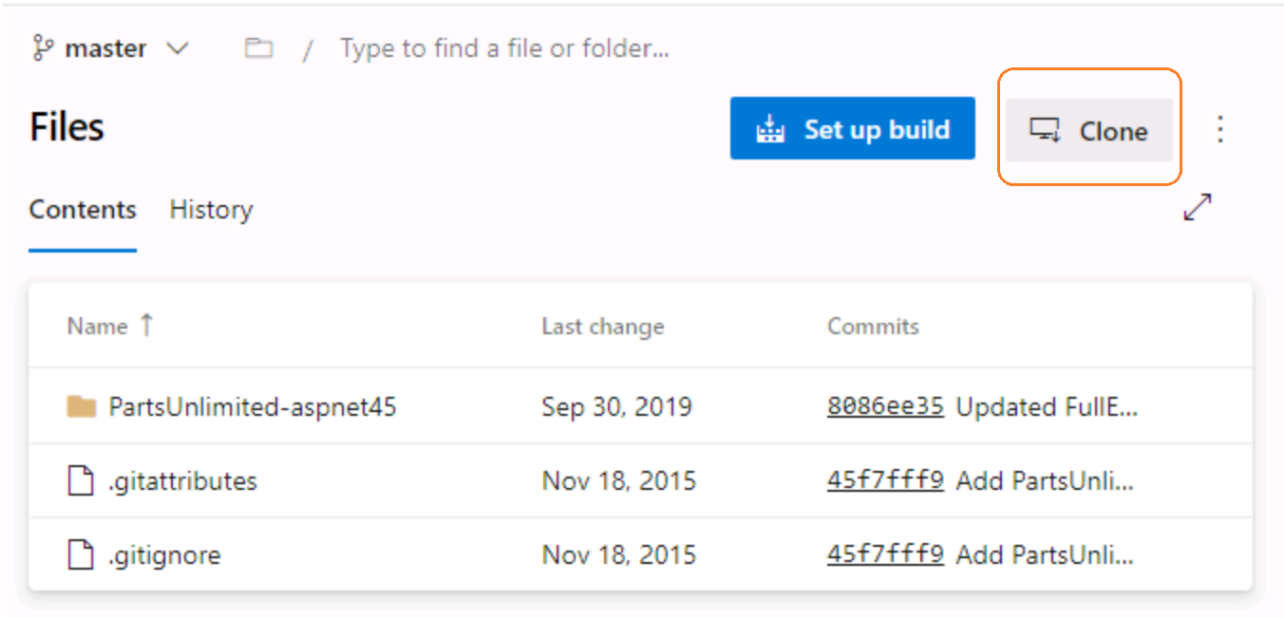
Task 1: Cloning an Existing Repository

1. Navigate to Azure DevOps Services in the browser to the **PartsUnlimited** project. The URL would be https://dev.azure.com/AppInnovation-[YourName]/PartsUnlimited.

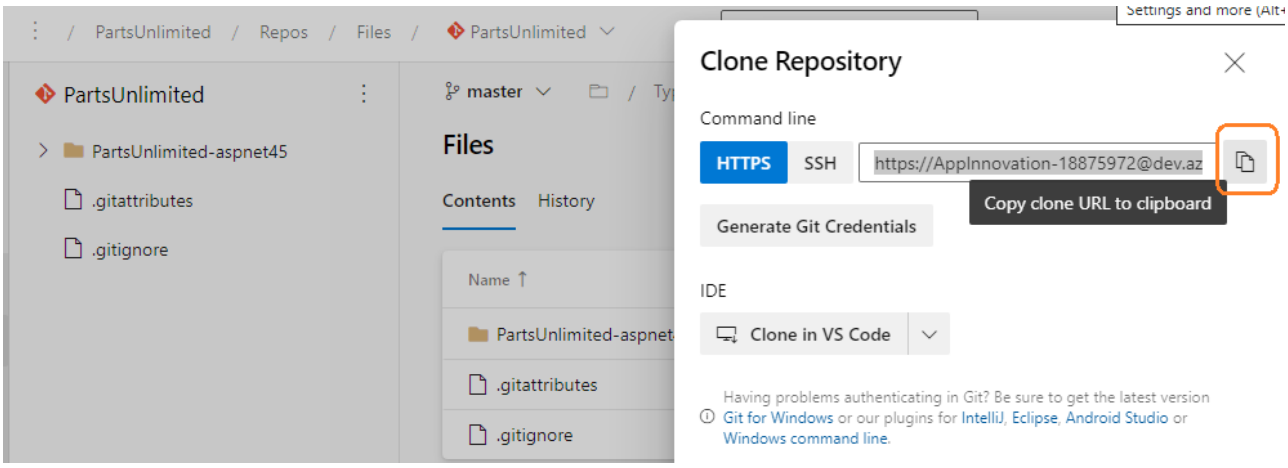
2. Navigate to the **Repos** hub and click **Clone**.







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.

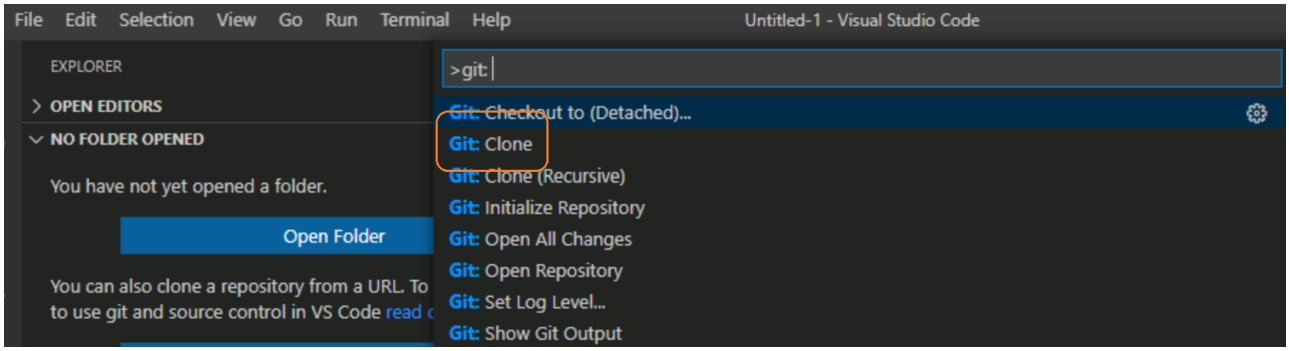


4. Switch back to the instance of **Visual Studio Code**.

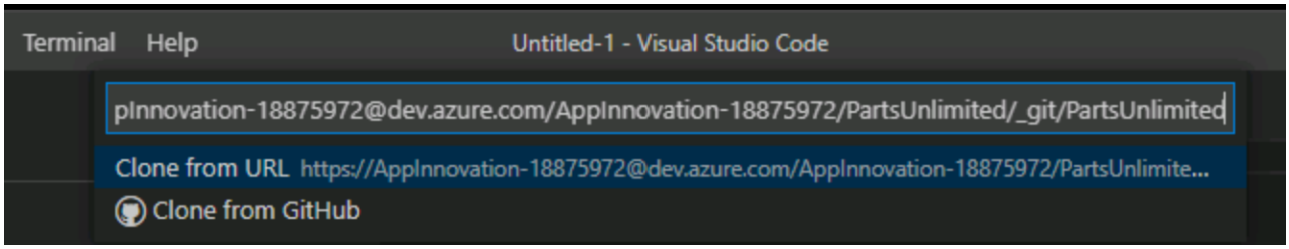
5. Press **Ctrl+Shift+P** to show the **Command Palette**. Alternatively, you can open the Command Palette by navigating to **View -> Command Palette**.

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

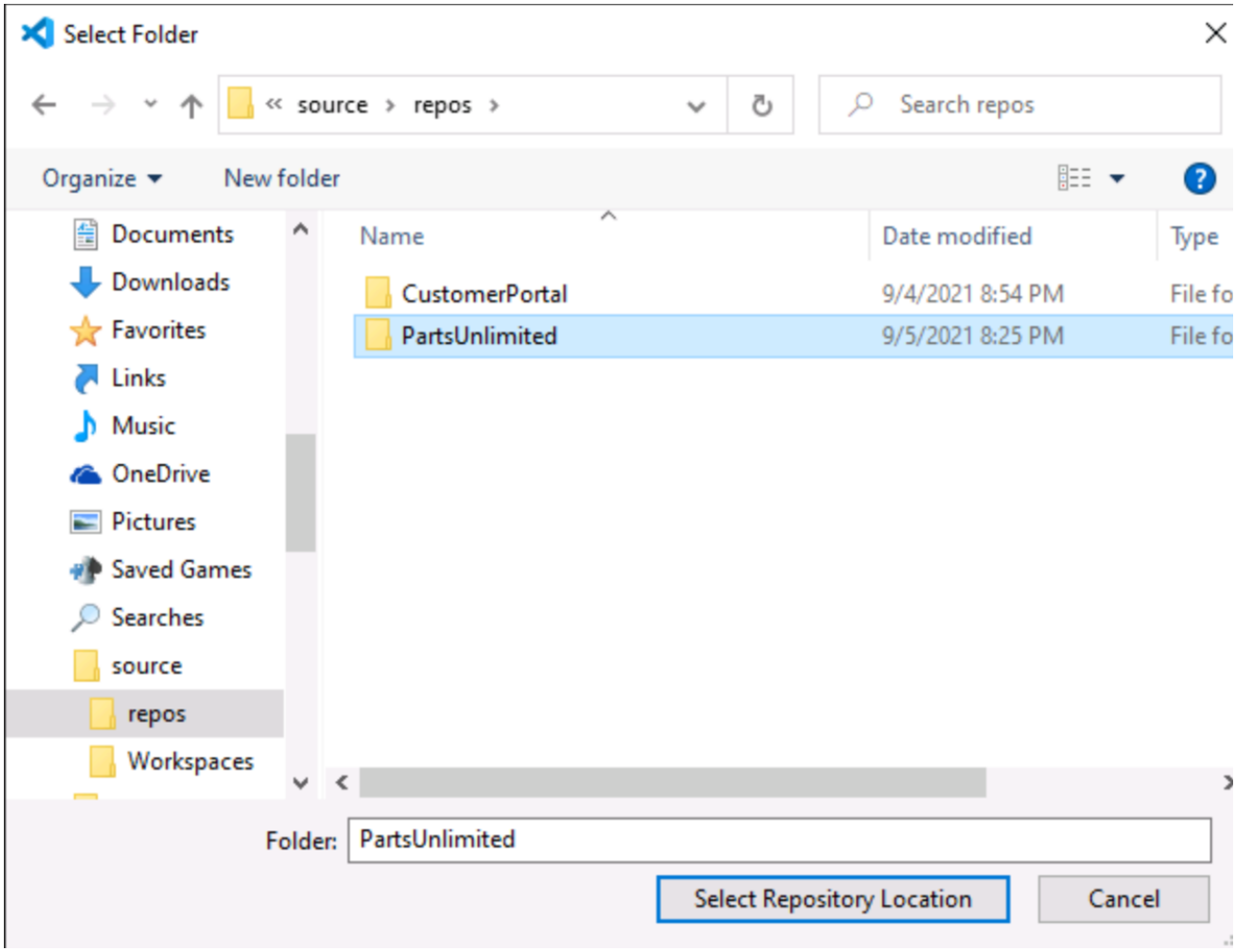
6. Execute the **Git: Clone** command. It may help to type "Git" to bring it to the shortlist.



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

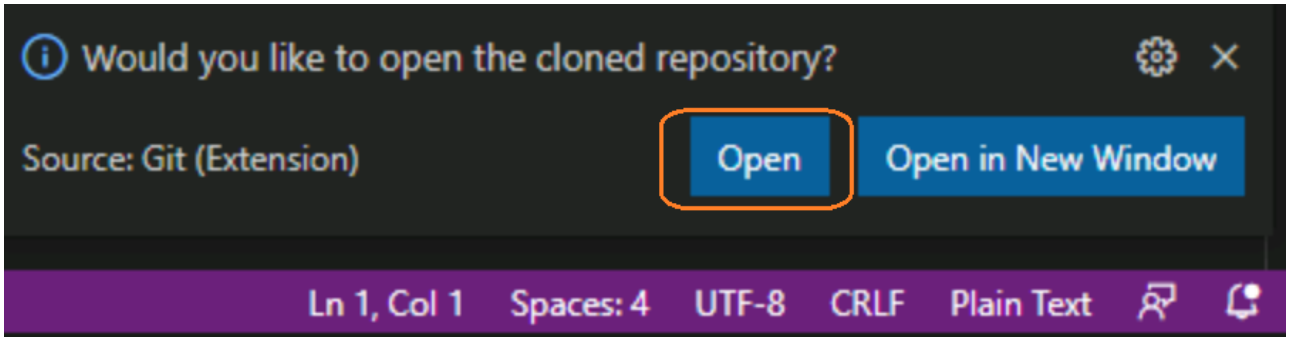


8. Navigate to the local path **C:\Users\StudentPC\source\repos**. Create a new folder **PartsUnlimited** and point to this folder.



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.



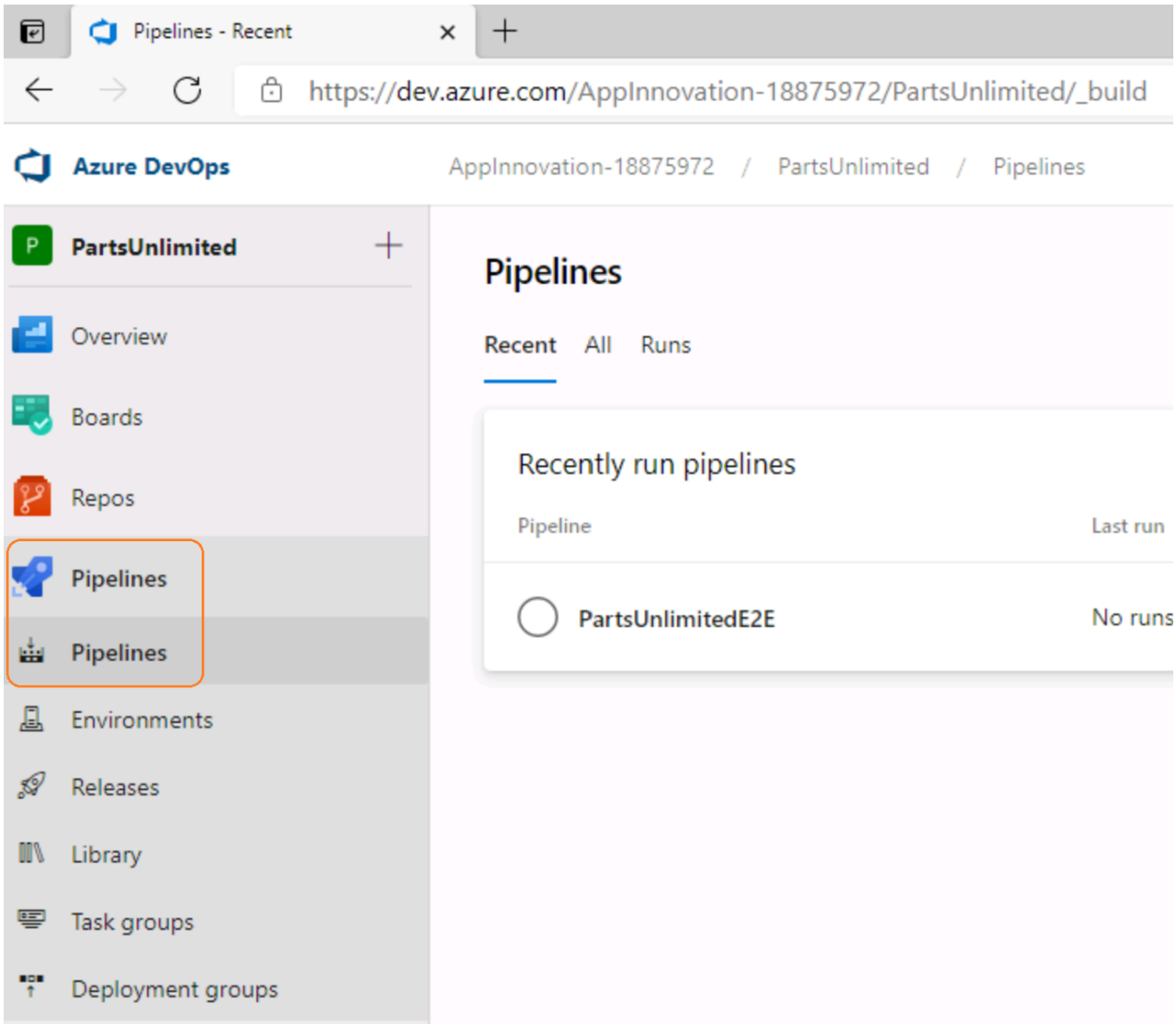
Module 3: **Azure Repos with Git**, **Lab 1: Getting Started with Git**, Exercise 2: Cloning Existing Azure Repository

---

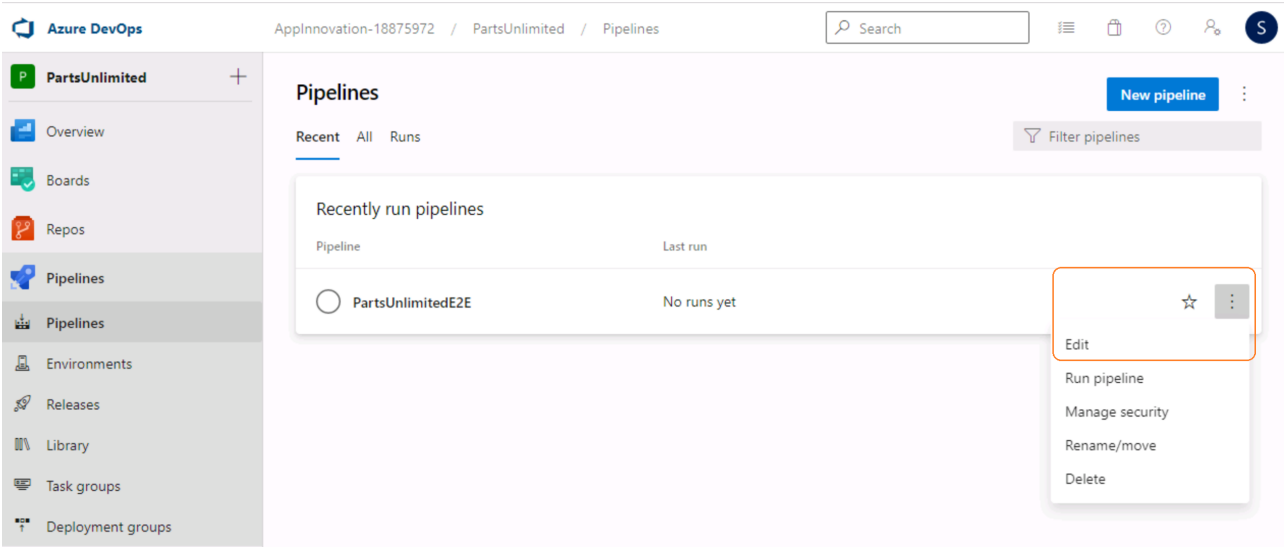
Task 2: Configuring the PartsUnlimited Project

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**.

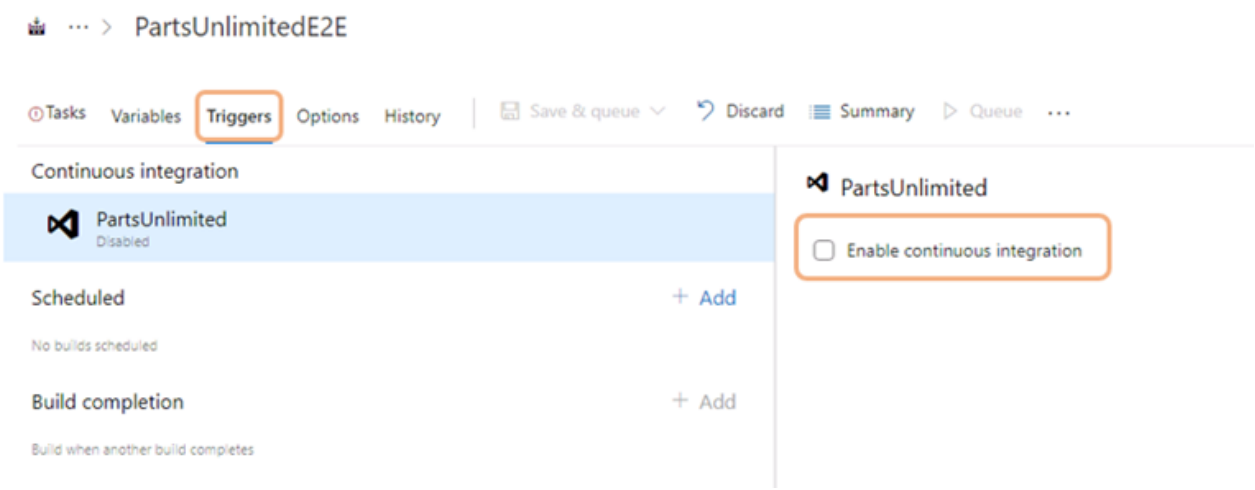


3. PartsUnlimited project includes a default build definition, PartsUnlimitedE2E. Click on **More options** and click **Edit**.

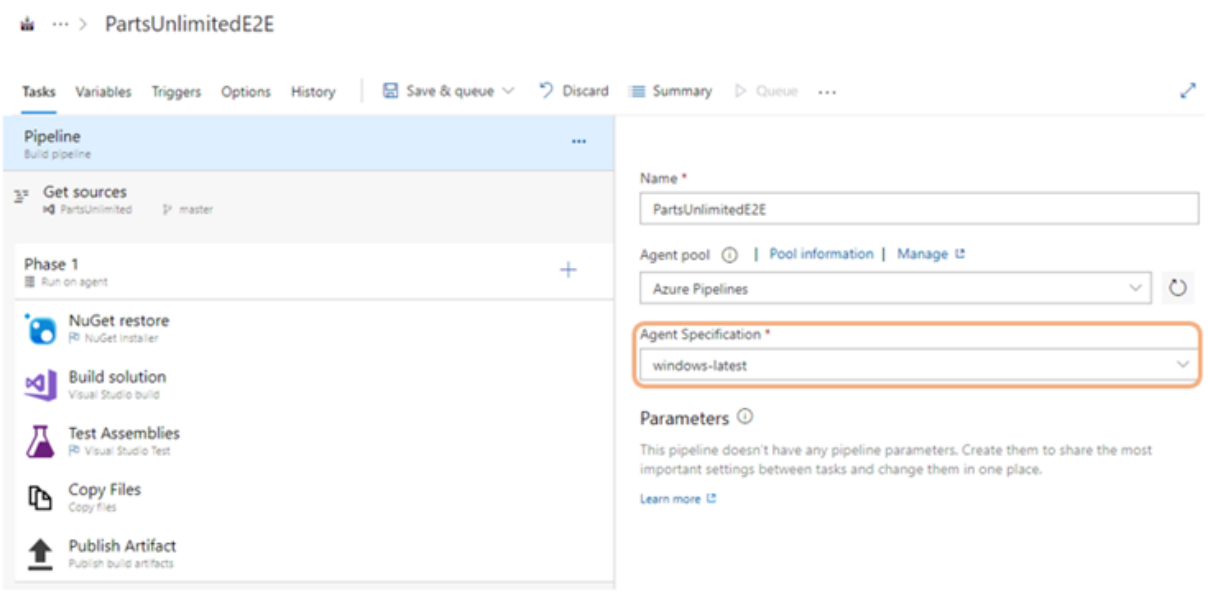


4. This build is scheduled to run whenever a change is committed. We don't want to wait for this build every time there's a change, so select the **Triggers** tab and uncheck **Enable continuous integration**.

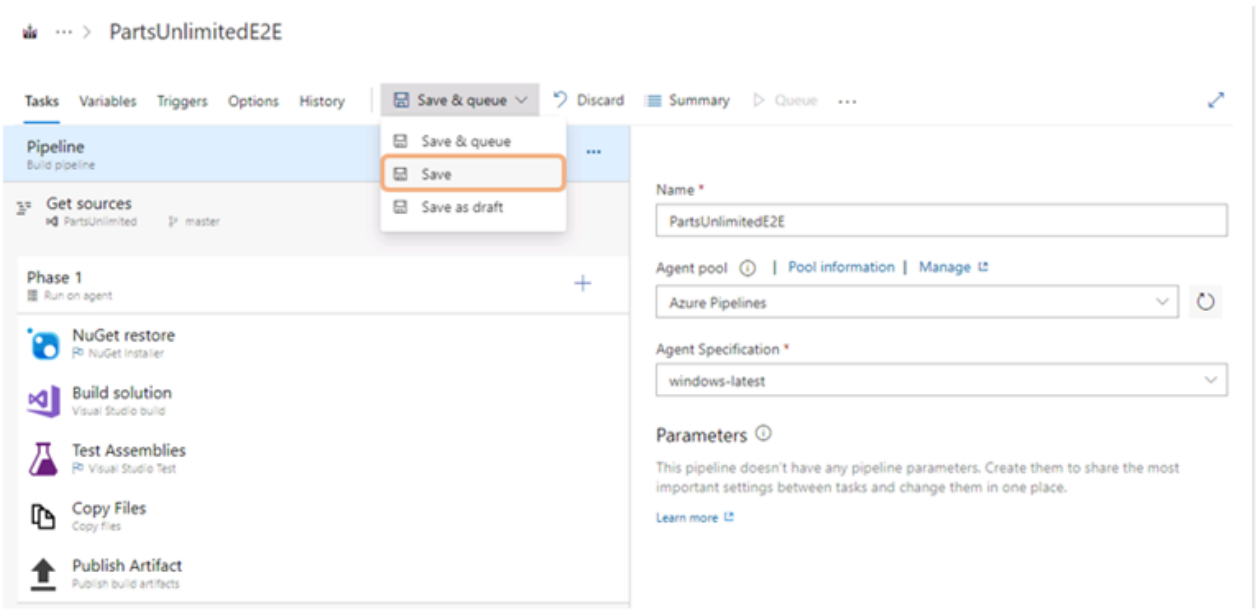
> This checkbox triggeres the build pipeline whenever there is a change in the master branch. This is the *CI* component of the CI/CD process which we will discuss later. For now, we need to disable CI so that our changes in the master branch in this lab won't trigger builds every time.

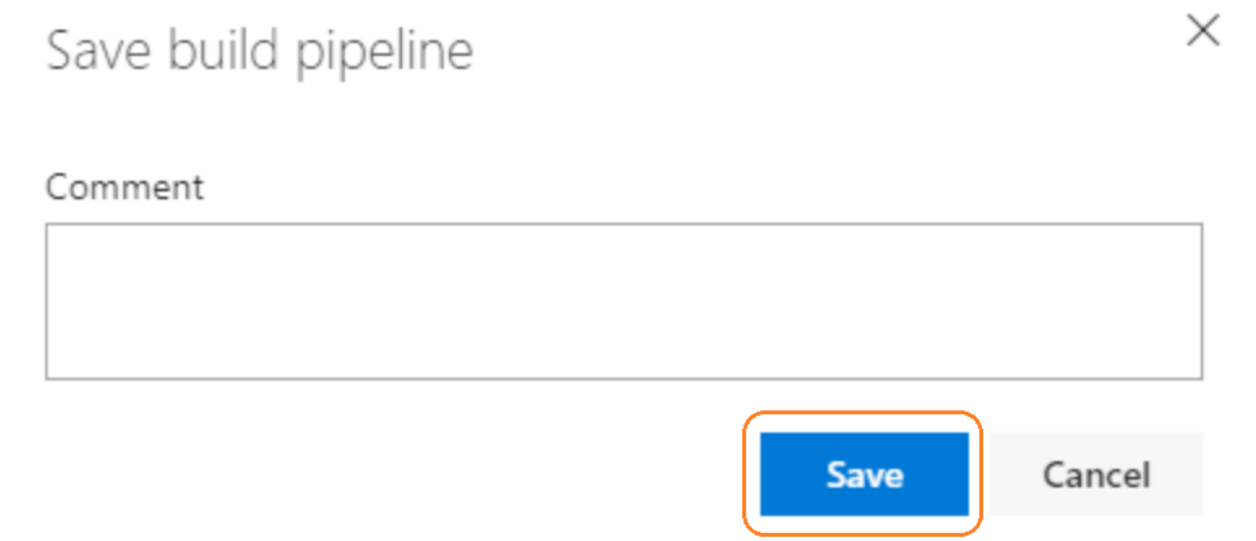


5. To be able to save the change in the build definition, we need to fix all the issues. Select the **Tasks** tab and select **windows-latest** in the dropdown for **Agent Specification**.



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





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

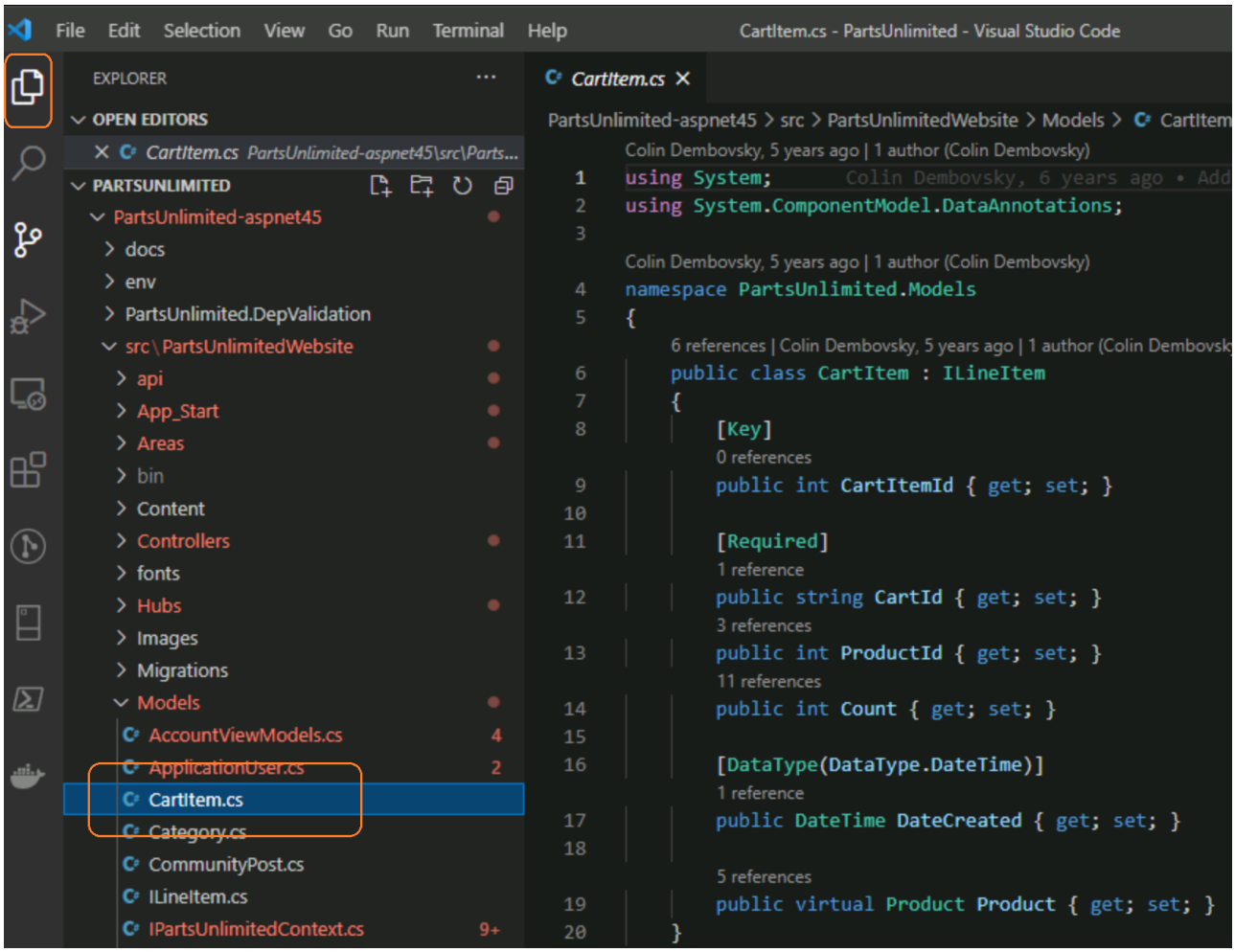
Tasks

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

---

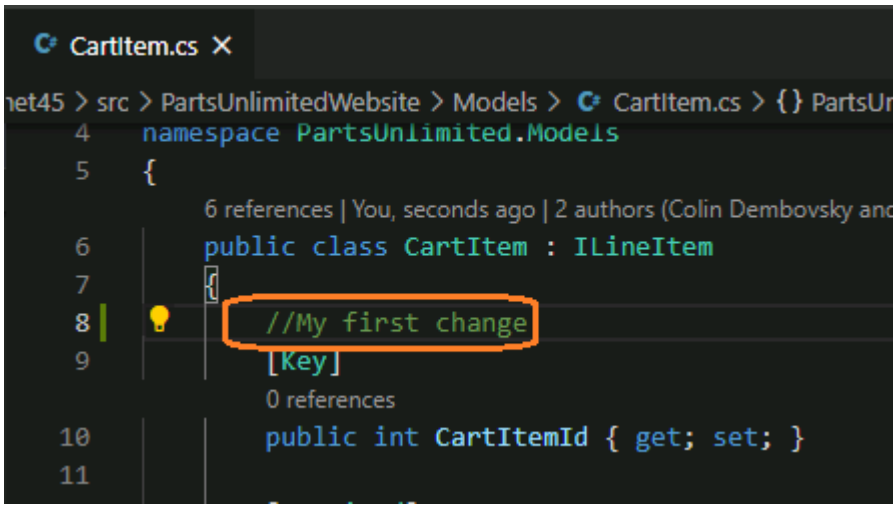
Task 1: Committing Changes

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

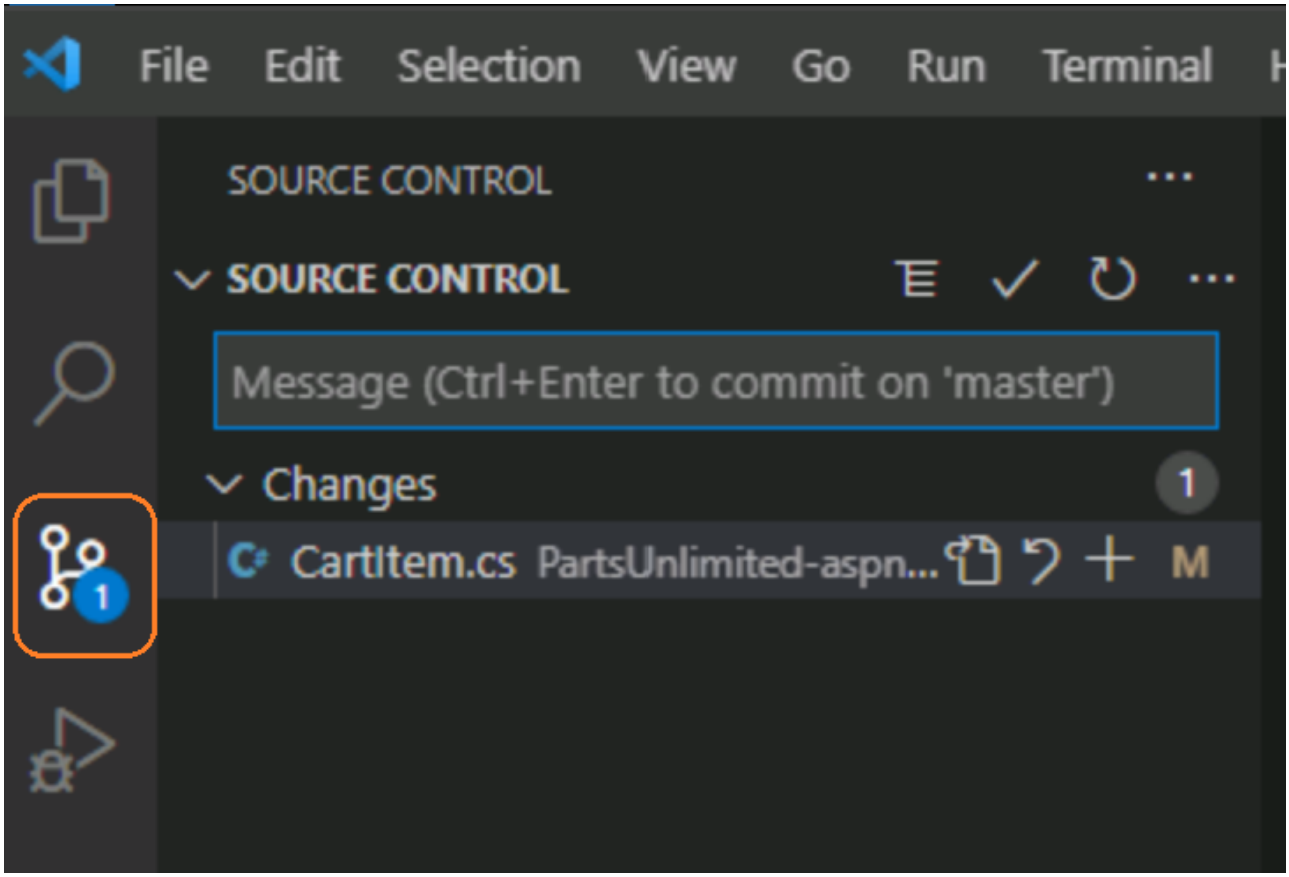


2. Add a comment to the file: **//My first change**. Press **Ctrl+S** to save the file. You can also save the file by navigating to **File -> Save**

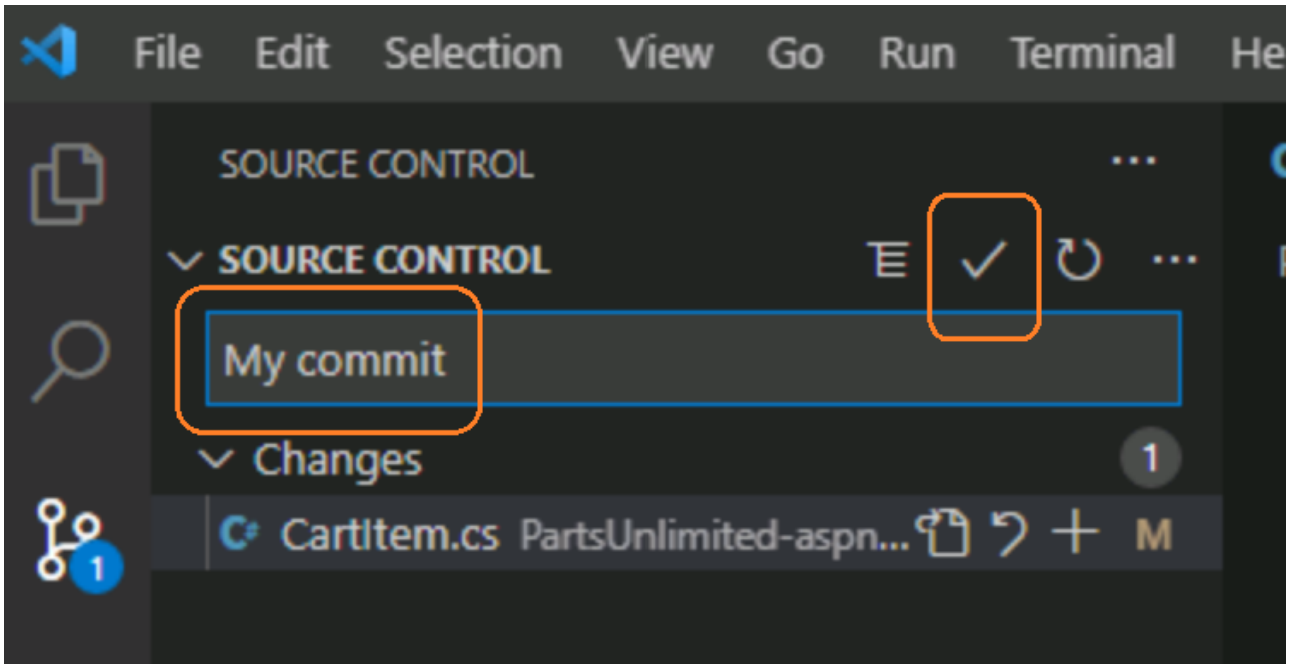
> It doesn't really matter what the comment is and where you add it in the file since the goal is just to make a change in the file.



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

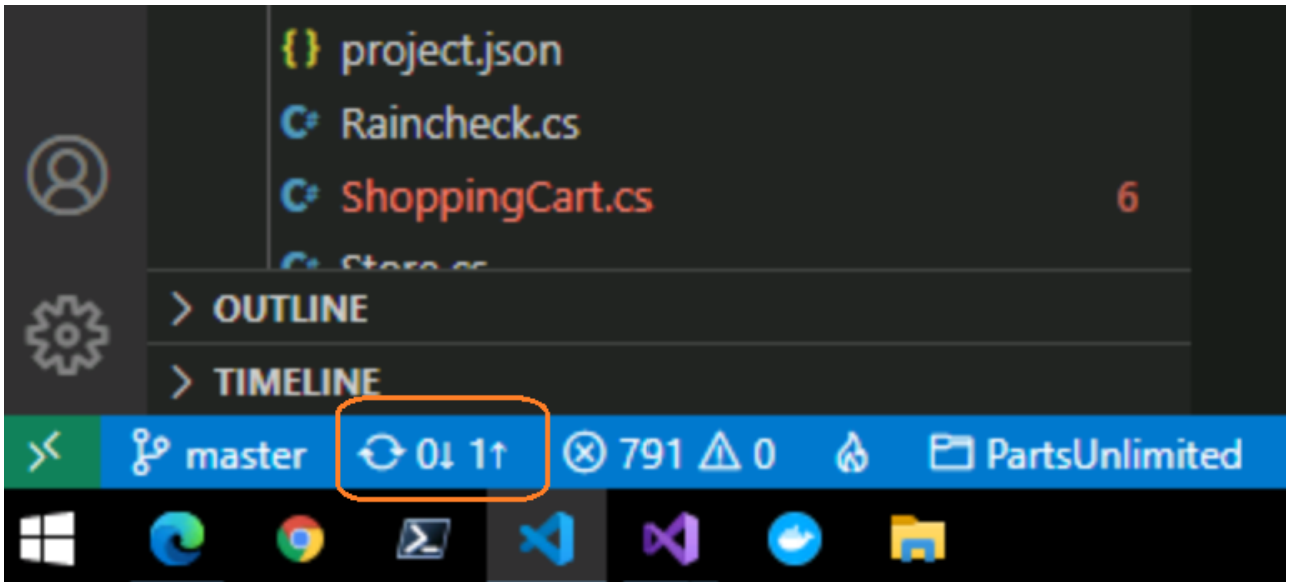


4. Enter a commit message of **My commit** and press **Ctrl+Enter** to commit it locally. Alternatively, you can click on **Commit**.



> 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.



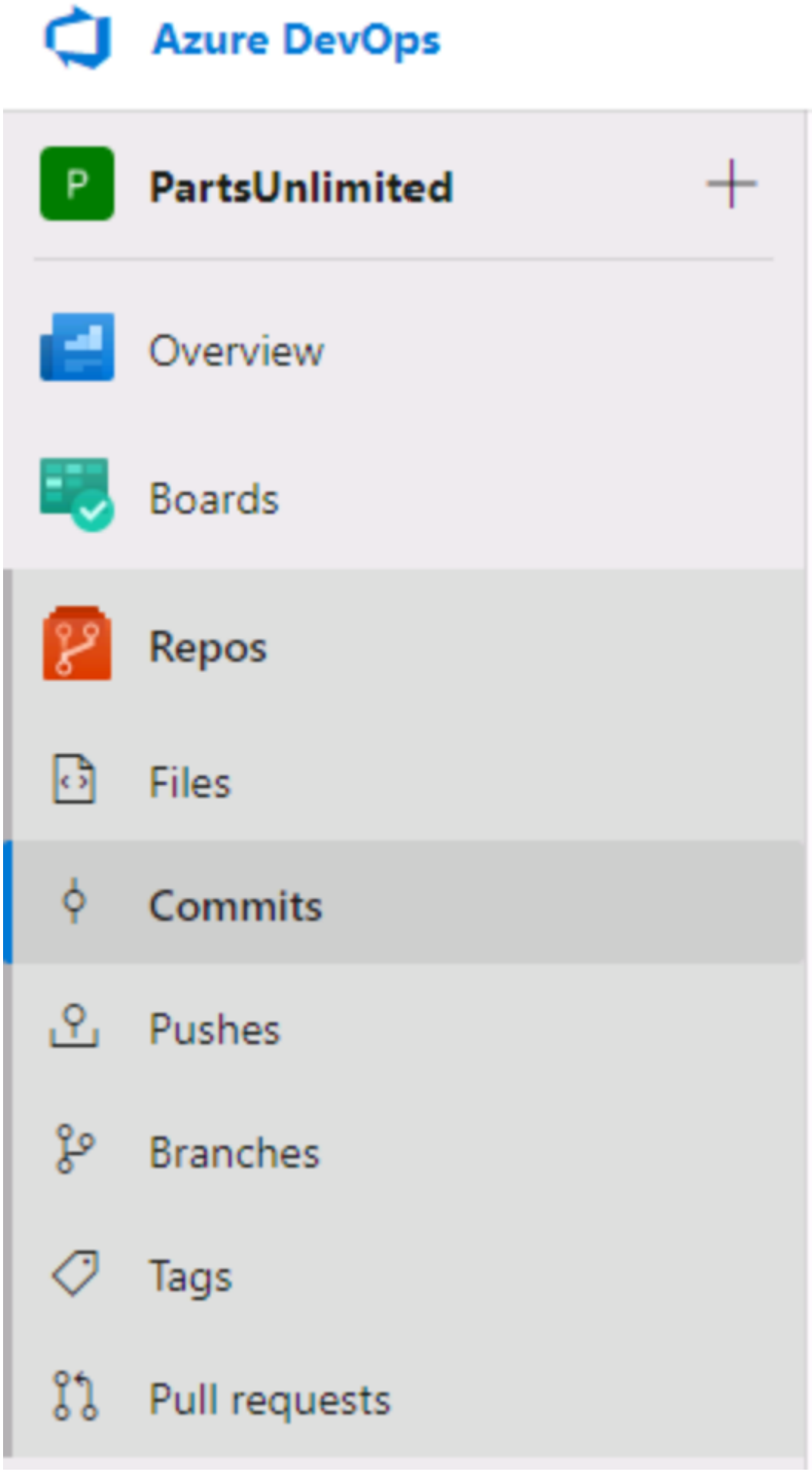
> Since you have made one commit locally that we need to push to the remote repository, the sync icon reflects this one change to be uploaded.

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

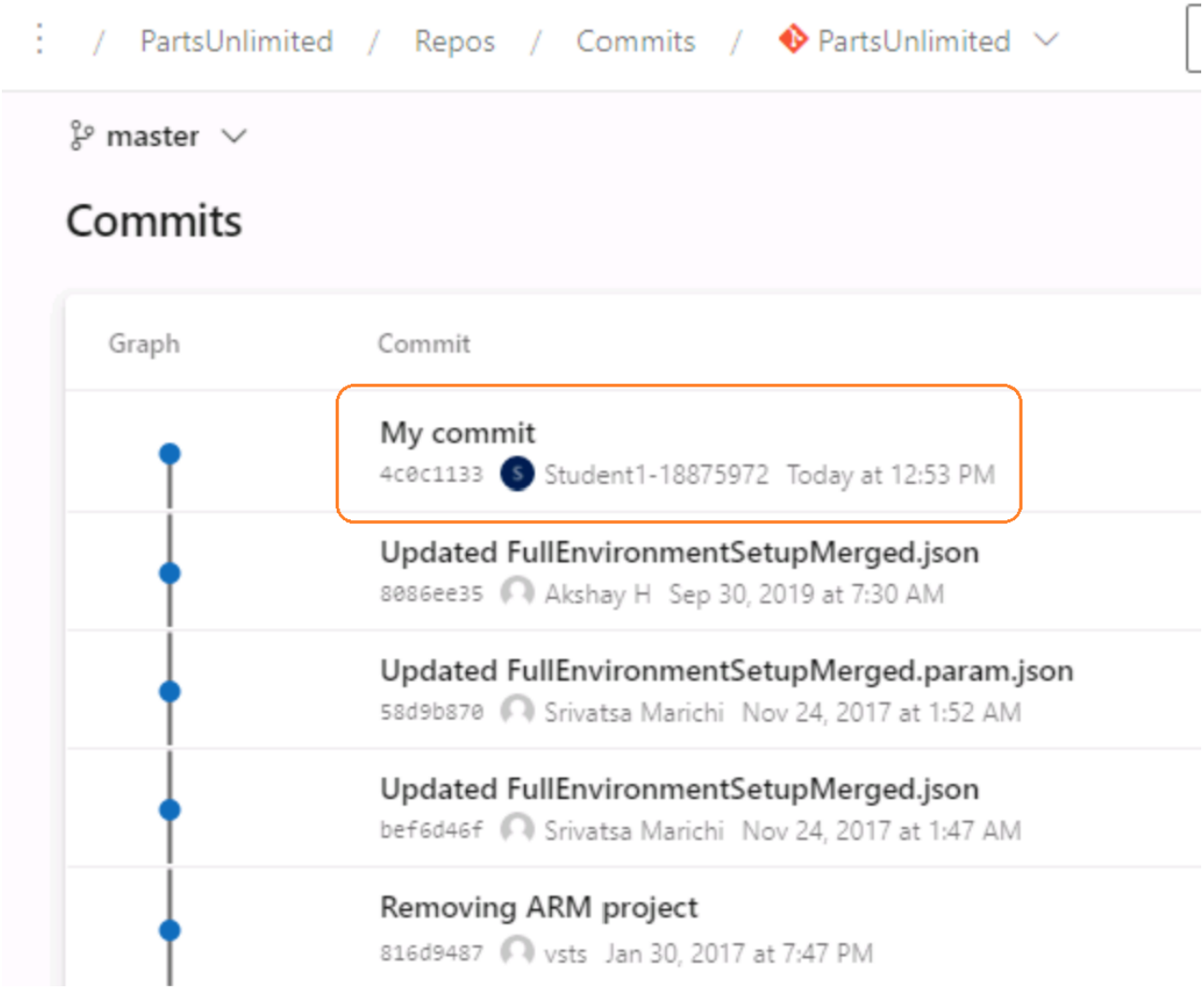
---

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.



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



> 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.

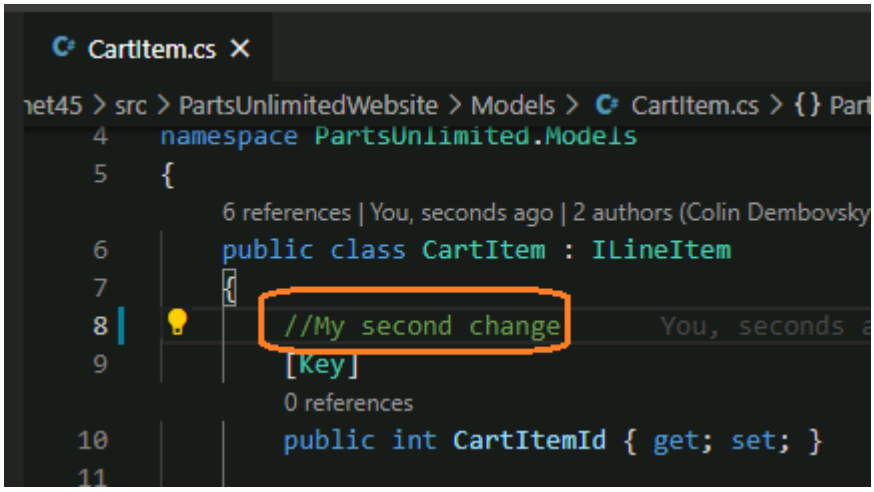
Module 3: **Azure Repos with Git**, **Lab 1: Getting Started with Git**, Exercise 3: Saving Work Locally with Commits

---

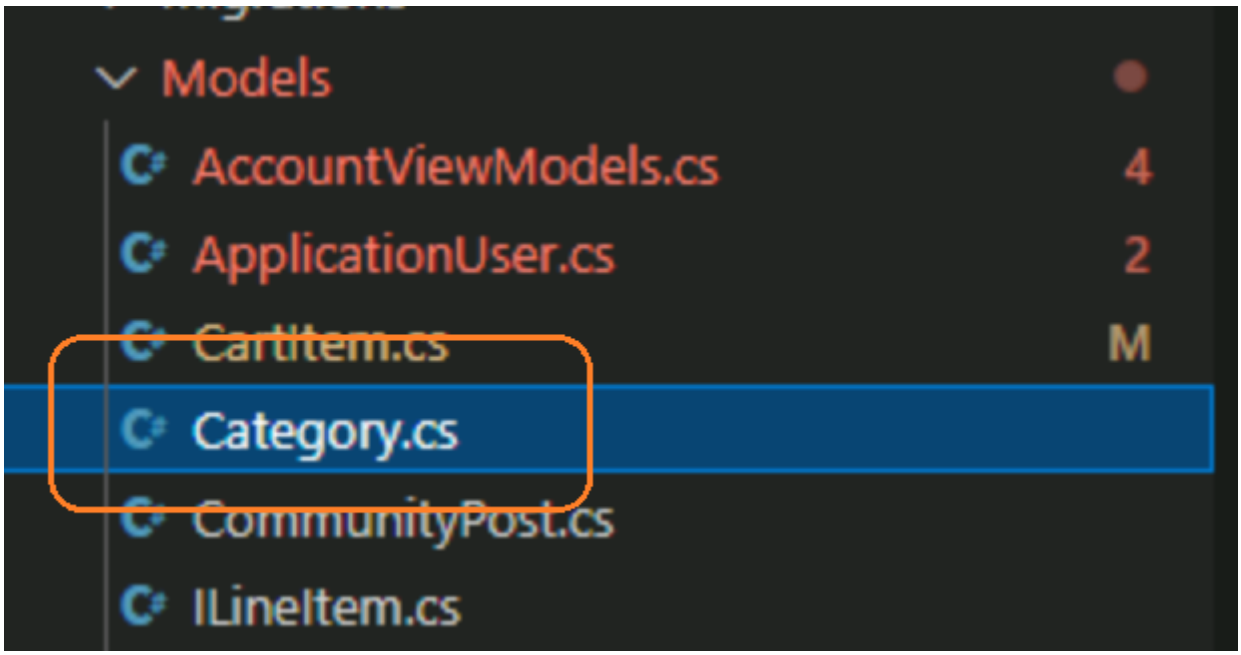
Task 3: Staging Changes

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

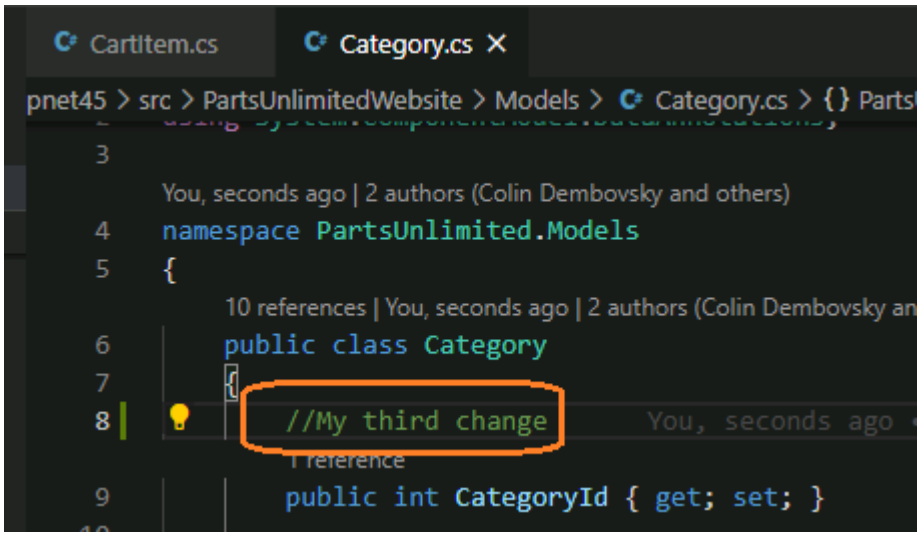
1. Return to **Visual Studio Code** with **PartsUnlimited** project open in it.
2. Edit the **CartItem.cs** file by editing the comment you made earlier and changing it to **//My second change**. **Save** the changes.



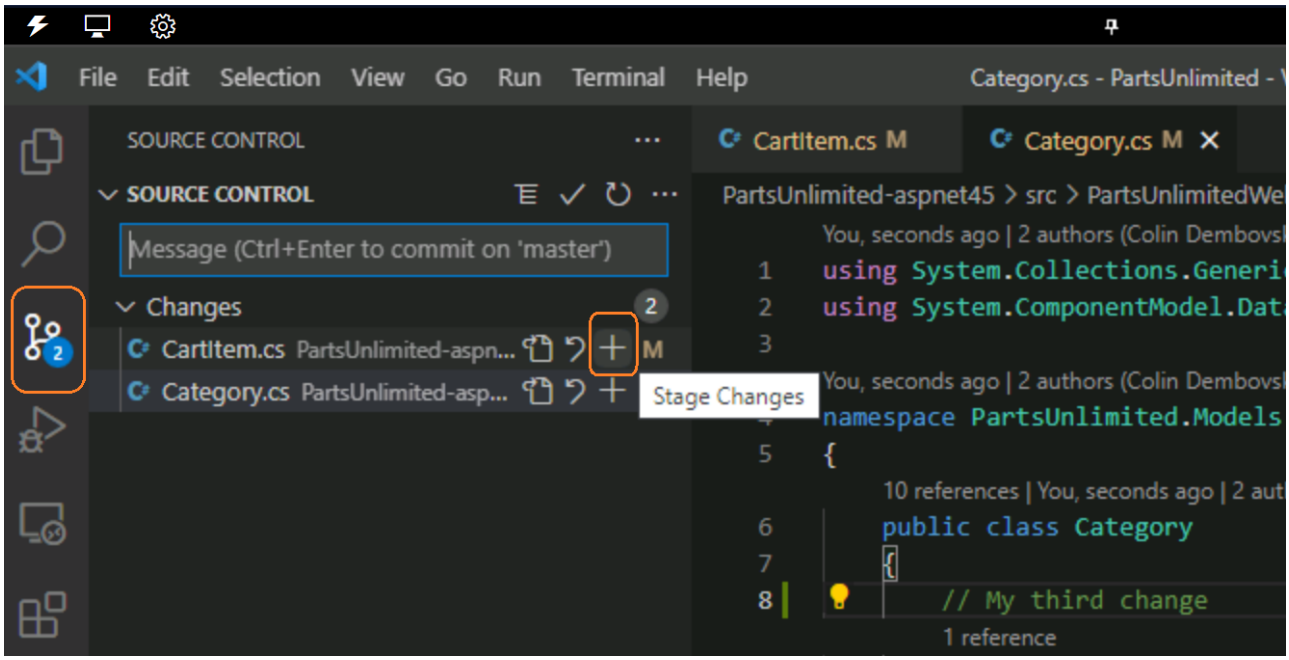
3. Click on the **Explorer** in the left tab of Visual Studio Code and open **Category.cs** file.



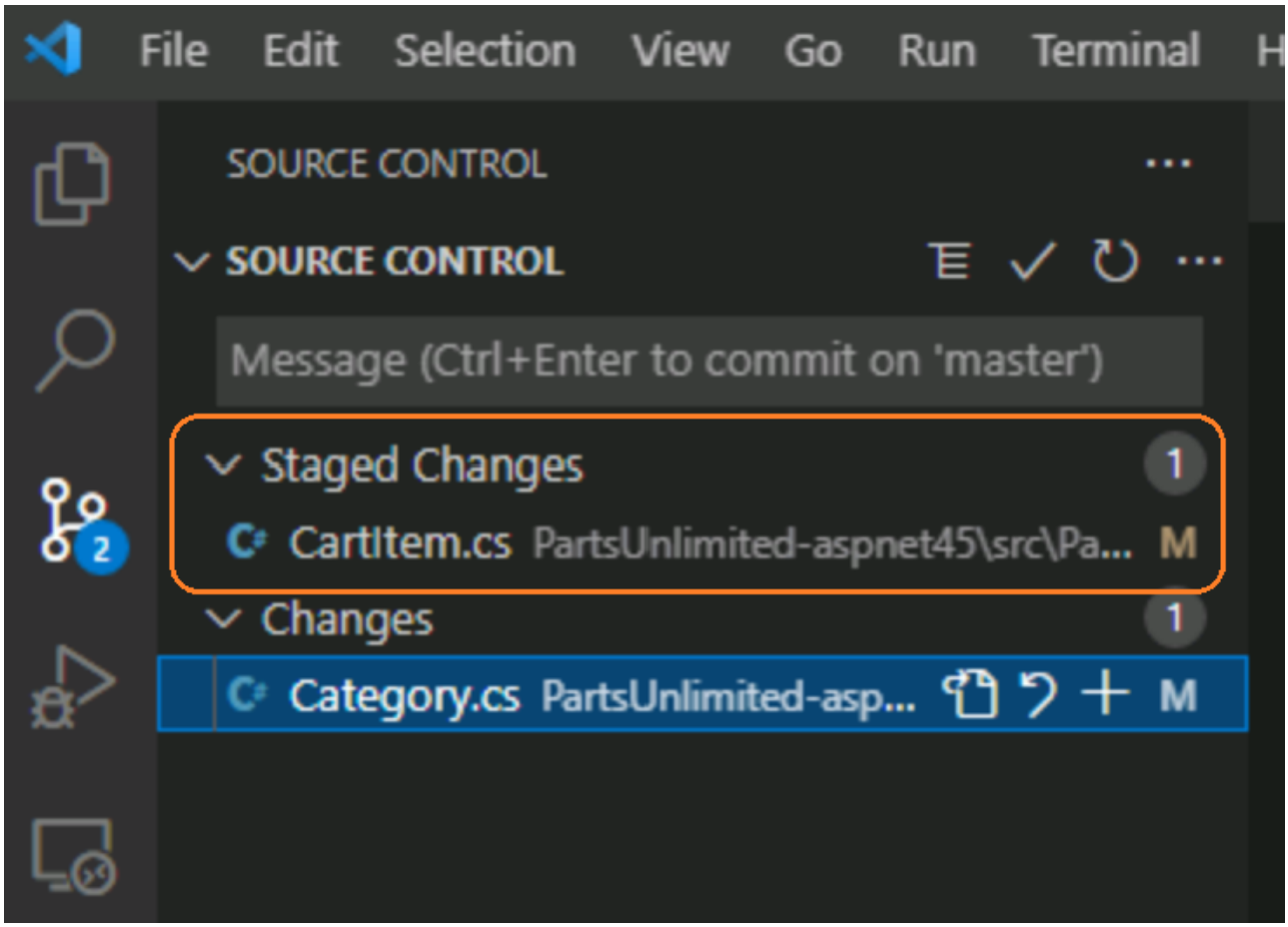
4. Add a new comment to Category.cs file: **//My third change** so there will be two files with changes. **Save** the file.



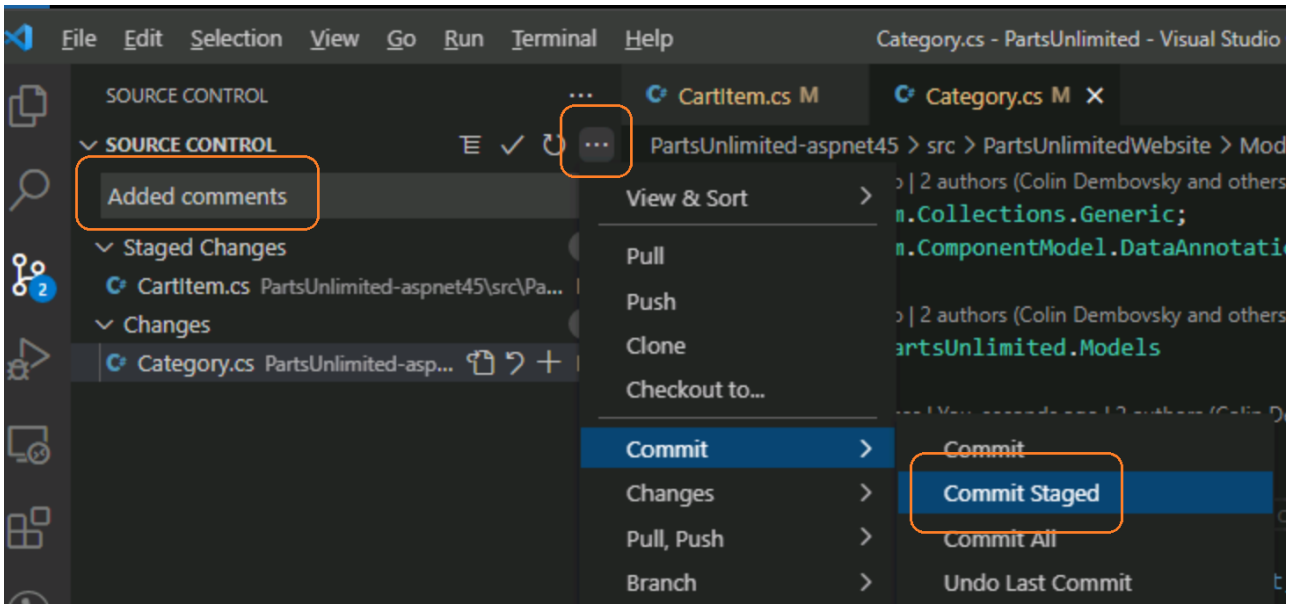
5. From the **Source Control** tab, click the **Stage Changes (+)** button for **CartItem.cs**.



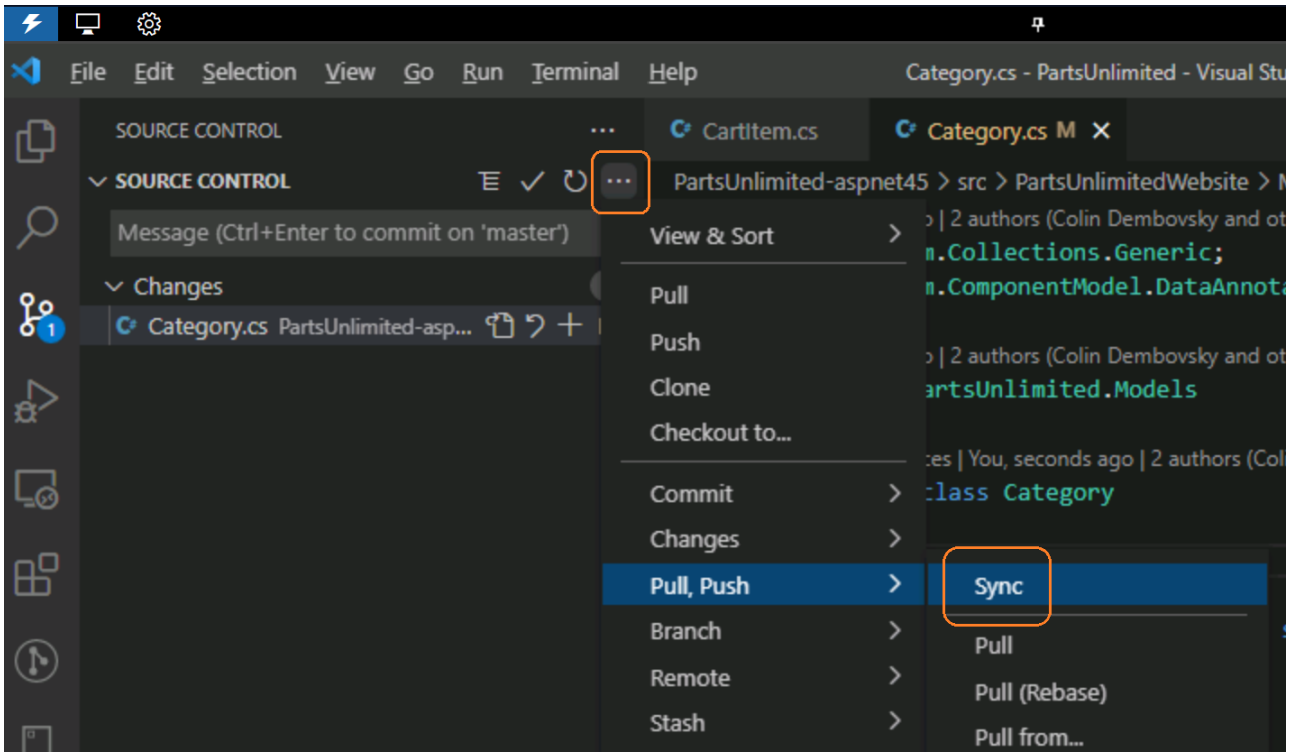
6. This will prepare CartItem.cs for committing without Category.cs.



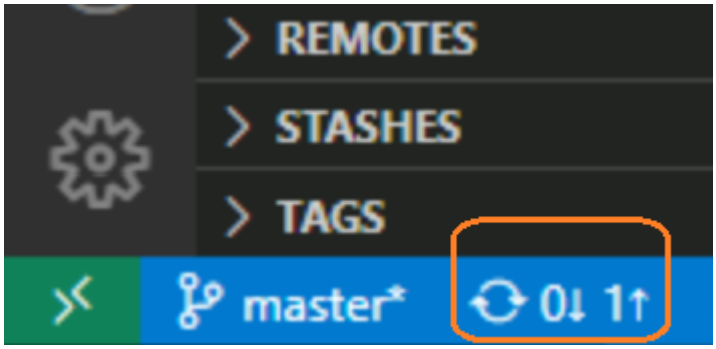
7. Enter a commit message **Added comments**. From the **More Actions** dropdown, select **Commit Staged**.



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

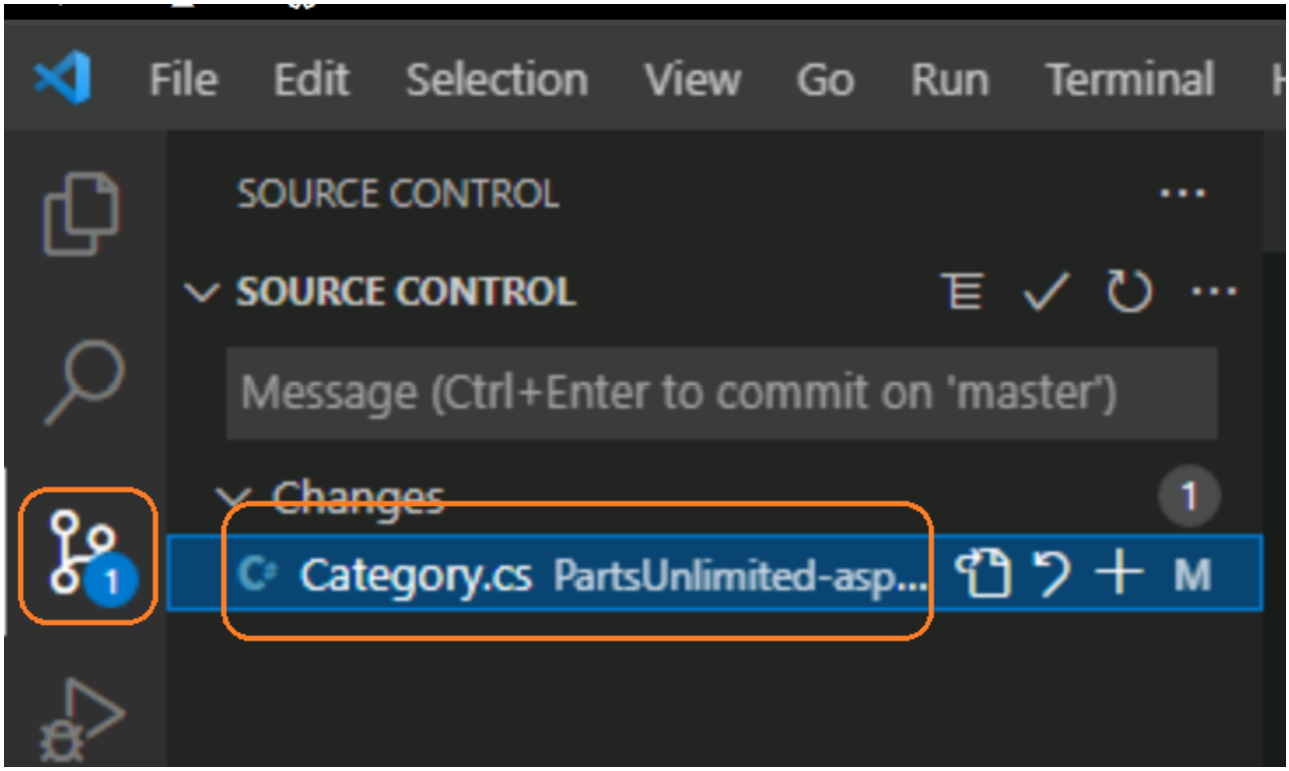
- Task 1: Comparing Files

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

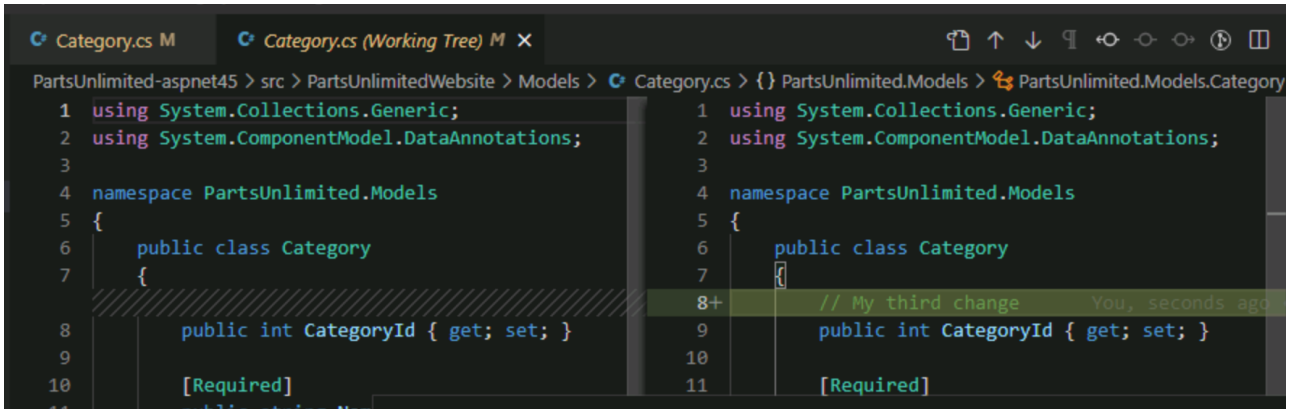
---

Task 1: Comparing Files

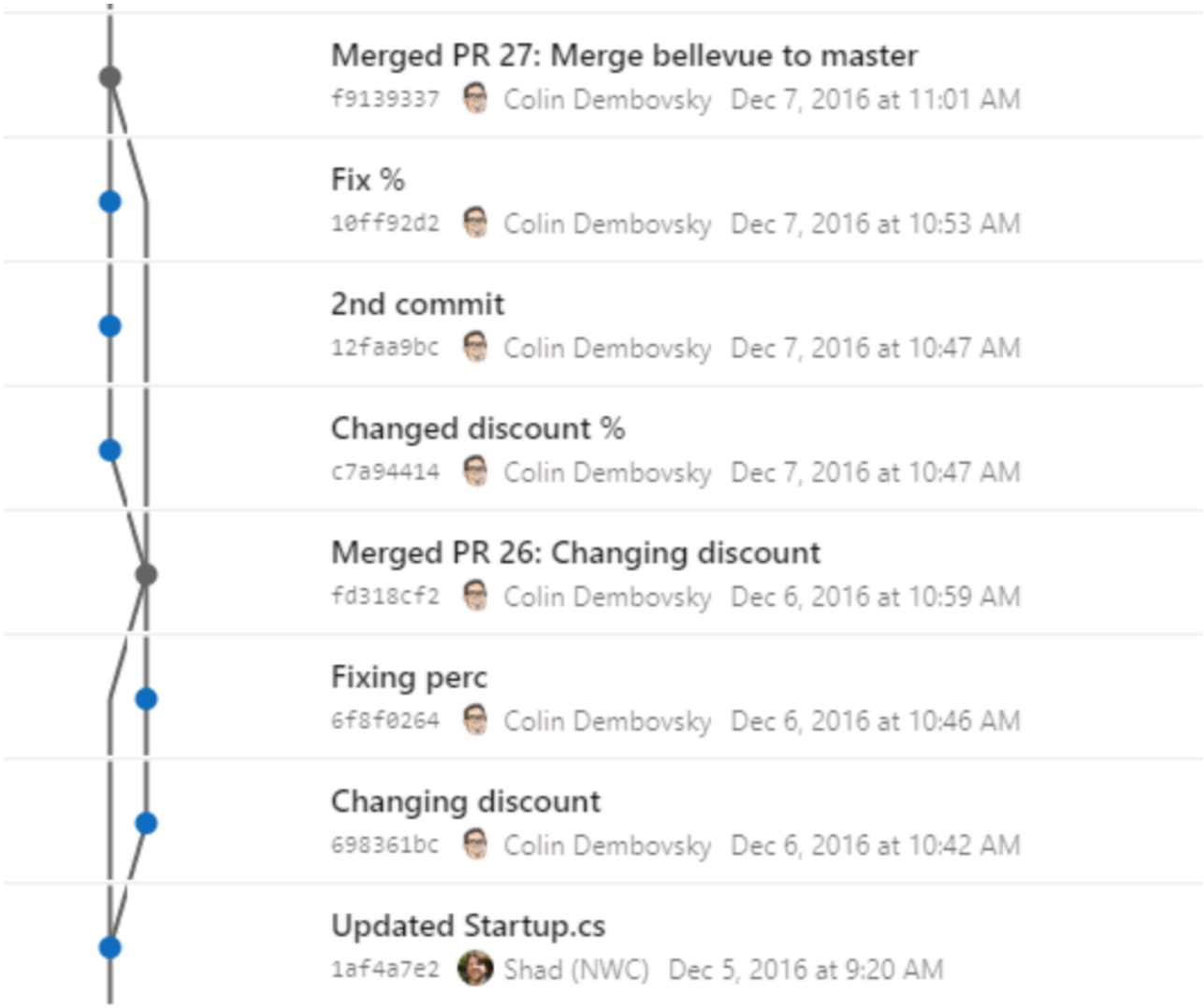
1. In **Visual Studio Code**, click on the **Source Control** tab and then click on **Category.cs** file.



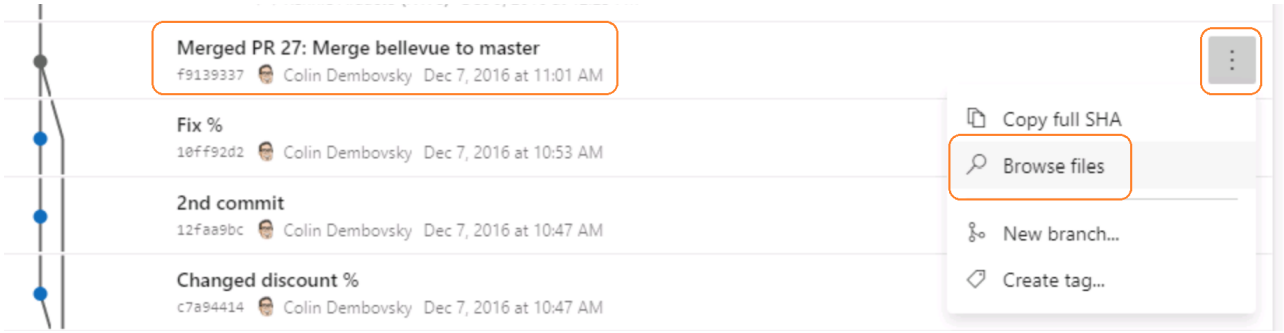
2. A comparison view is opened to enable you to easily locate the changes you've made. In this case, it's just the one comment.



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.

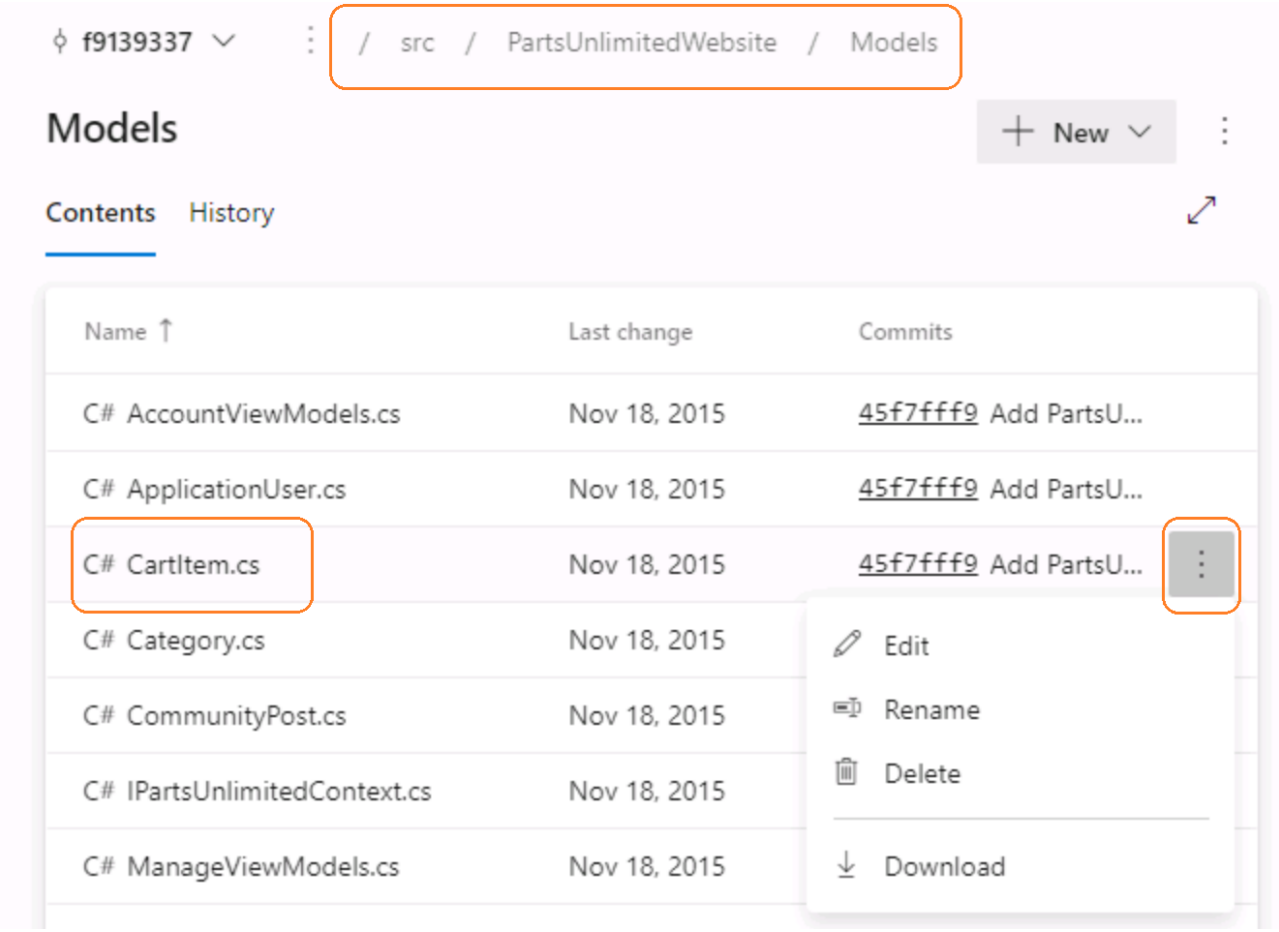


4. Locate **Merged PR 27: Merge bellevue to master** and click on **More options** and select **Browse files**.



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
45f7fff9 Add PartsU...
C# ApplicationUser.cs
Nov 18, 2015
45f7fff9 Add PartsU...
C# CartItem.cs
Nov 18, 2015
45f7fff9 Add PartsU...
C# Category.cs
Nov 18, 2015
C# CommunityPost.cs
Nov 18, 2015
C# IPartsUnlimitedContext.cs
Nov 18, 2015
C# ManageViewModels.cs
Nov 18, 2015
Edit
Rename
Delete
Download

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.

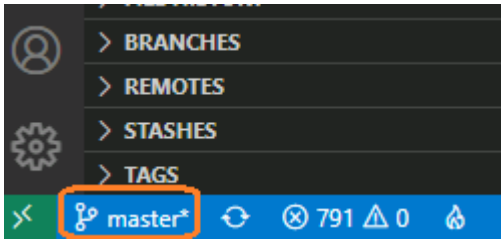
Tasks

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

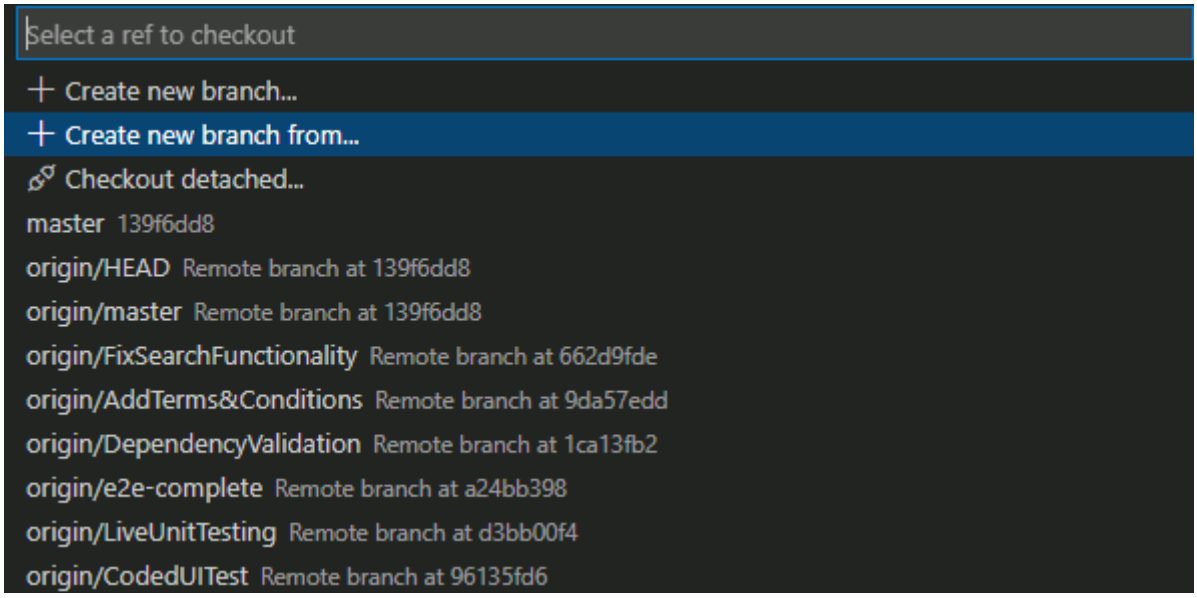
---

Task 1: Create a New Branch in Your Local Repository

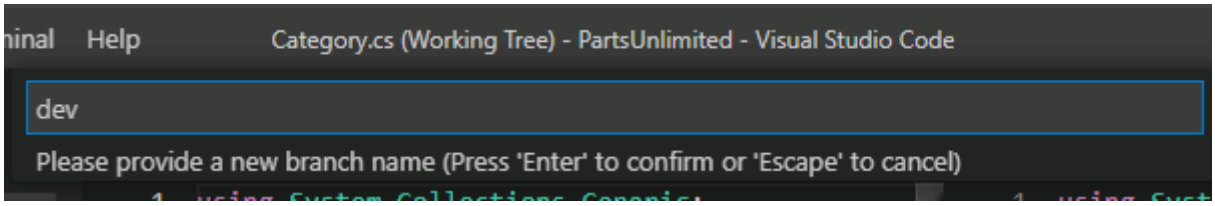
1. Return to **Visual Studio Code** and click the **master** branch from the bottom-left corner.



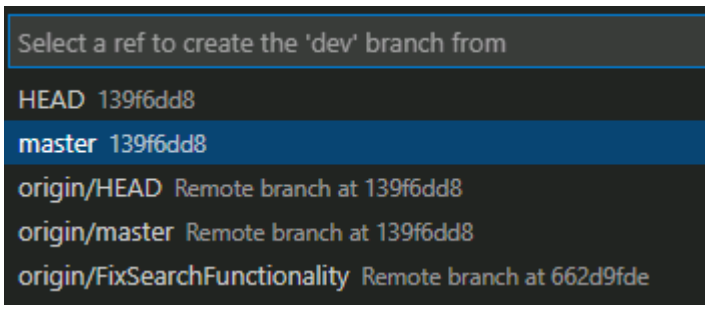
2. Select **+ Create new branch from...**.



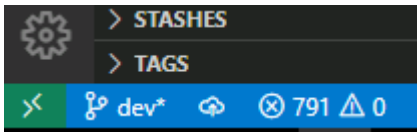
3. Enter the **Branch name** as **dev** for the new branch and press **Enter**.



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.



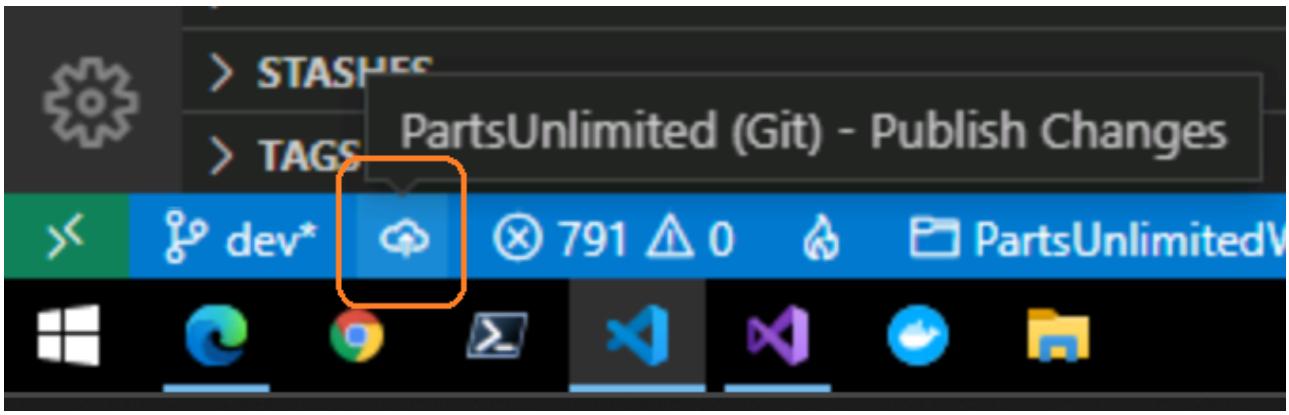
Module 3: **Azure Repos with Git**, **Lab 1: Getting Started with Git**, Exercise 5: Managing Branches Locally from Visual Studio Code

---

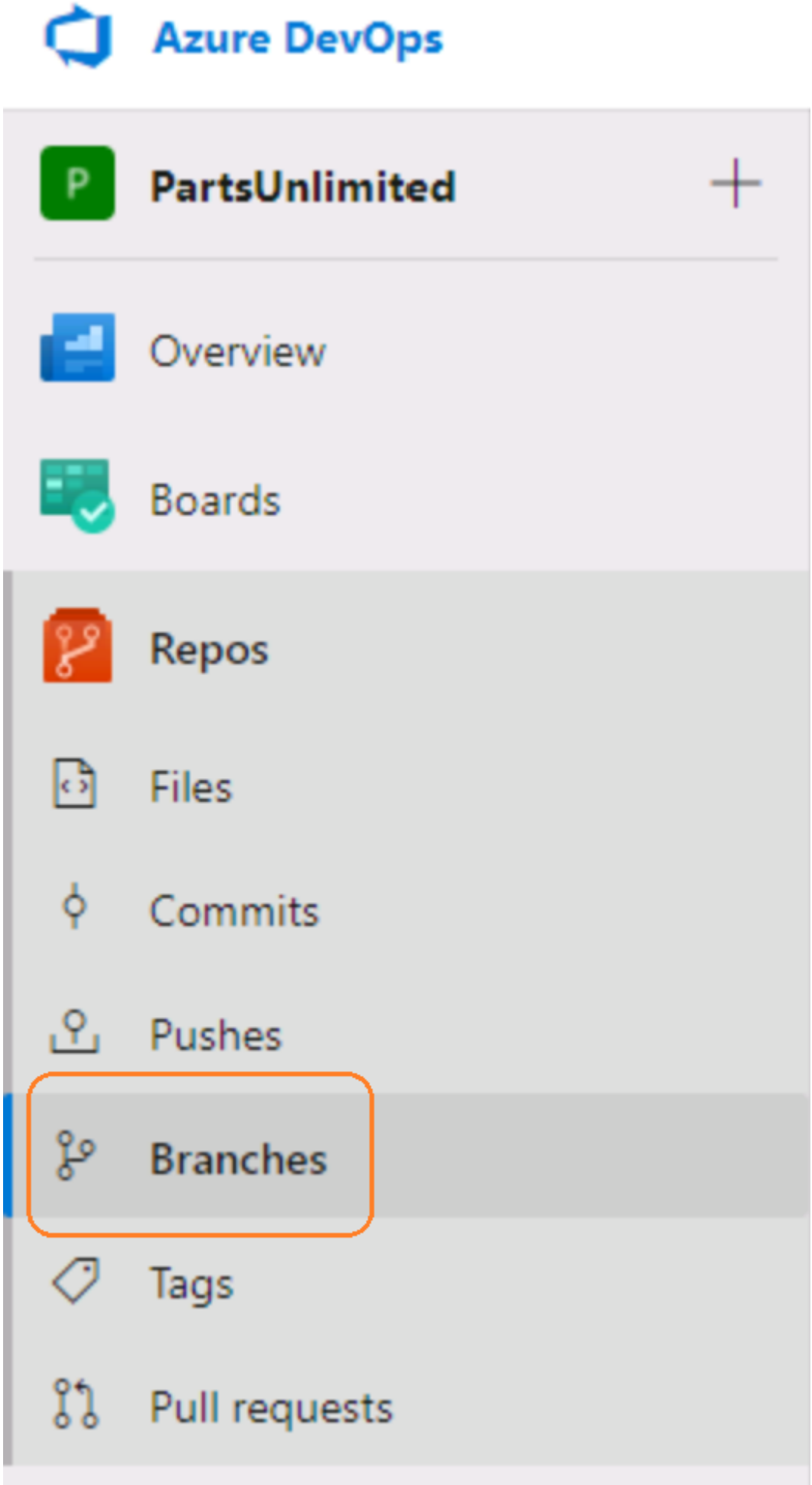
Task 2: Checking Out and Publishing Branches

> 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.

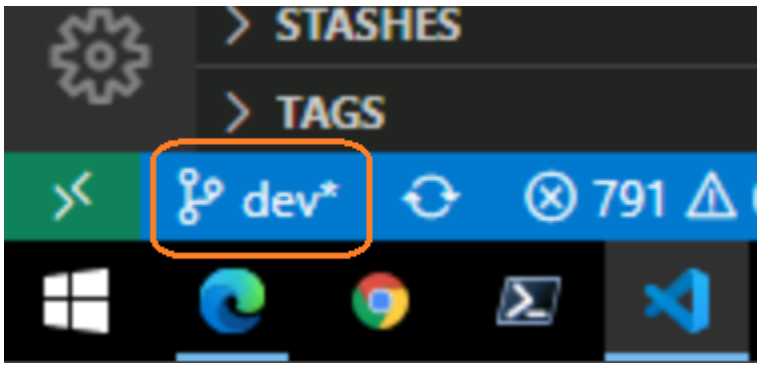


2. In the web browser go to **Azure DevOps Services**. Navigate to **Repos** hub and click **Branches**.

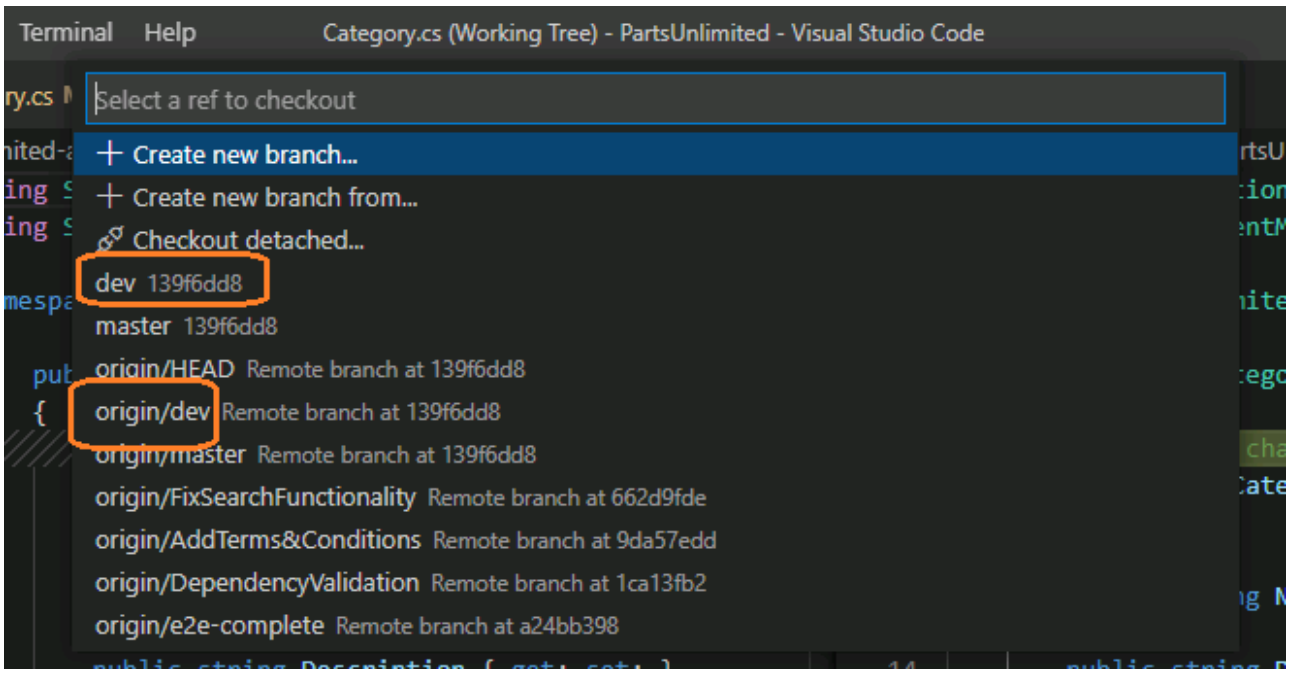

Azure DevOps
PartsUnlimited
Overview
Boards
Repos
Files
Commits
Pushes
Branches
Tags
Pull requests

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.

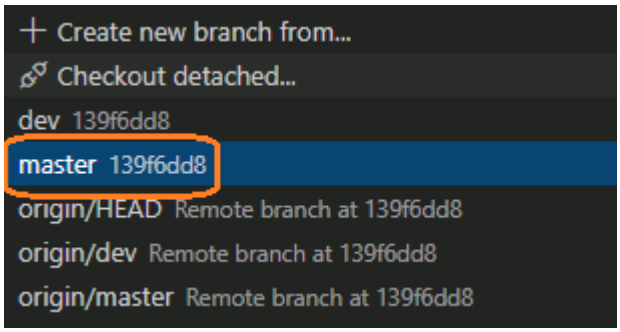
4. Return to **Visual Studio Code** and click the **dev** branch.

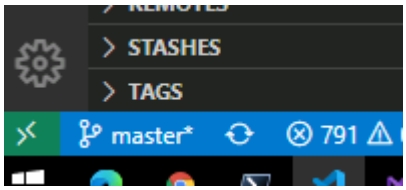


5. Note that there are two dev branches listed. The local **dev** branch is there because it's not deleted when the remote branch is deleted. The remote **origin/dev** is there because it hasn't been pruned.



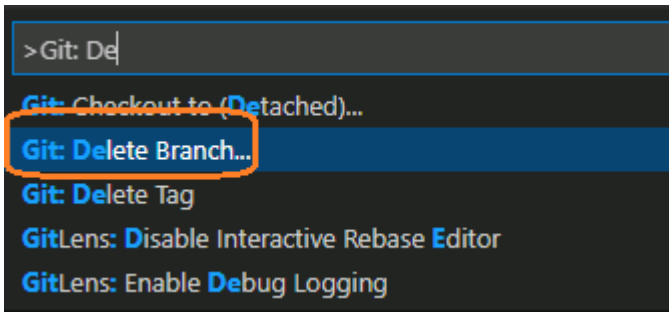
6. Select the **master** branch to check it out. Now you should see **master** branch checked out at the bottom-left corner of Visual Studio Code.



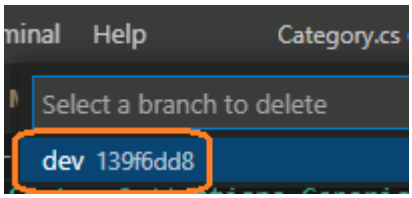


7. Press **Ctrl+Shift+P** to open the **Command Palette**.

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

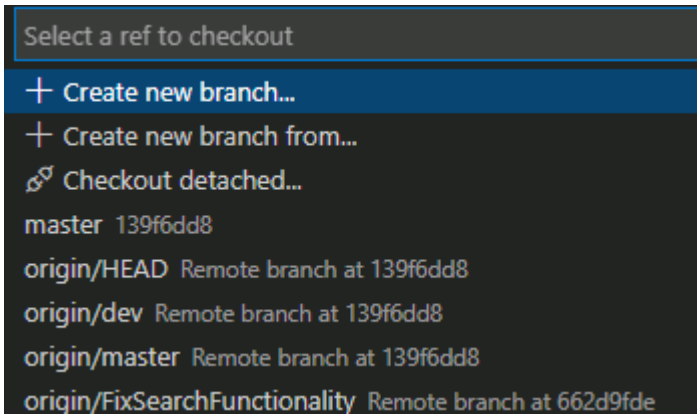


9. There is only one local branch (**dev**) to delete, so select this **dev** branch.



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.

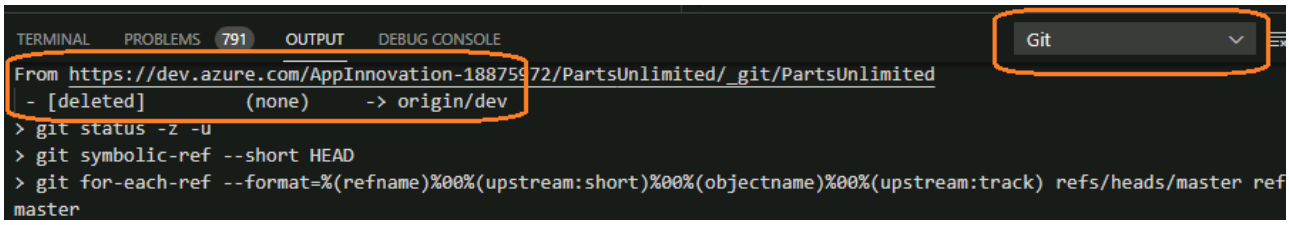


11. Press **Ctrl+Shift+P** to open the **Command Palette**.

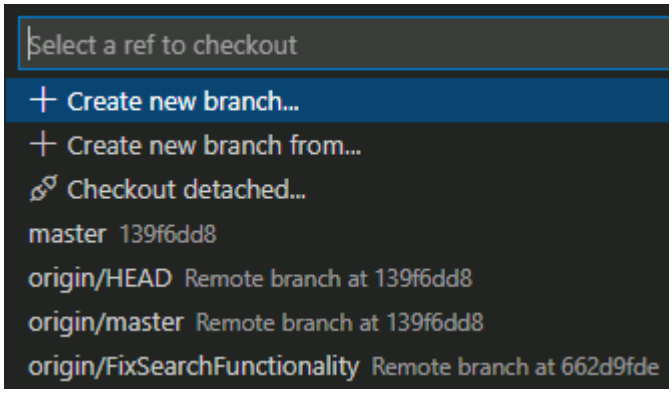
12. Start typing **Git: Fetch** and select **Git: Fetch (Prune)** when it becomes visible.

    > 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.



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 .

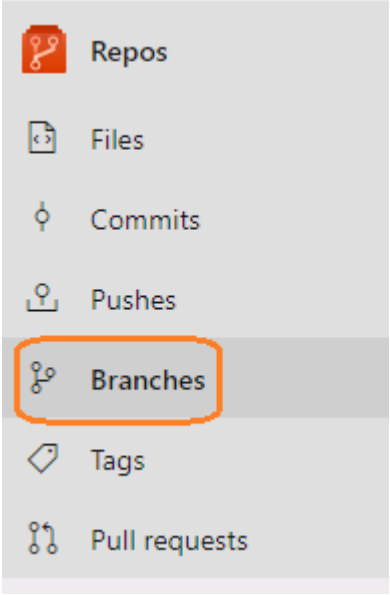
Tasks

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

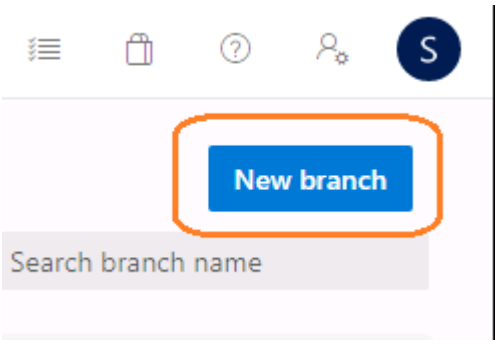
---

Task 1 Creating a New Branch

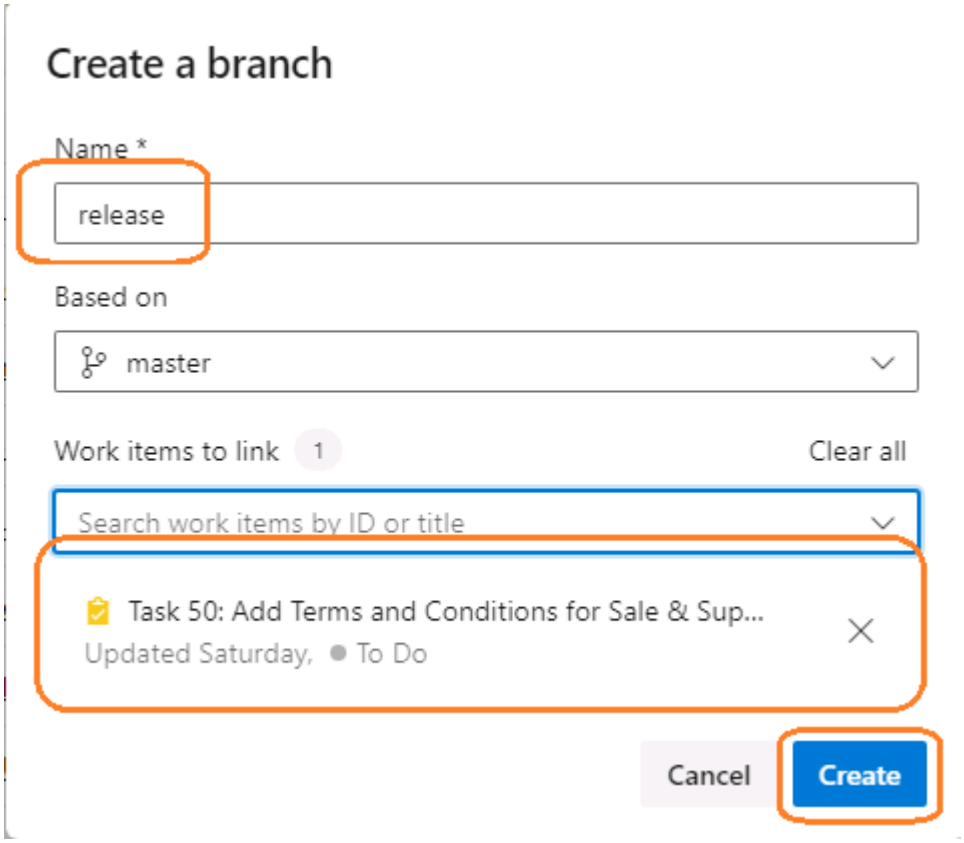
1. Switch to the **Azure DevOps Services** in the web browser.
2. Navigate to **Repos** hub and click on **Branches**



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.

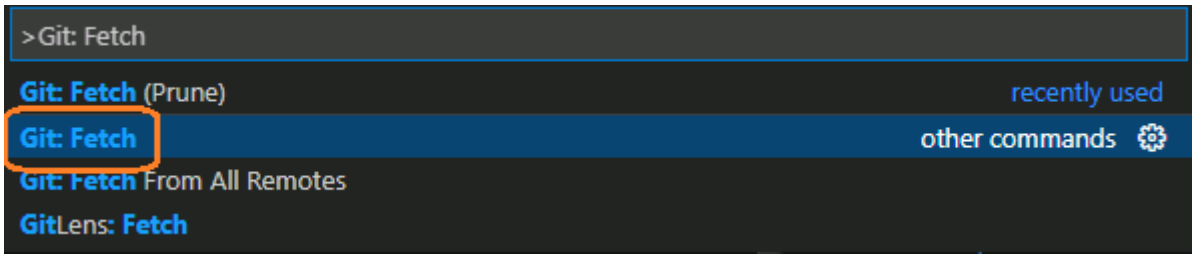


5. After the branch has been created, it will be available in the list.

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

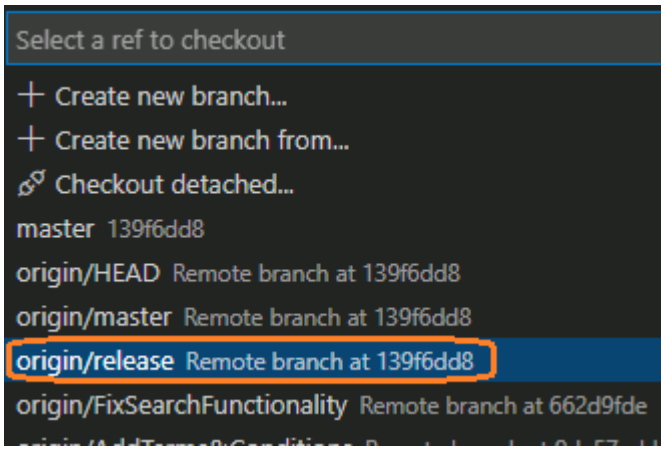
7. Press **Ctrl+Shift+P** to open the **Command Palette**.

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.


Select a ref to checkout
Create new branch...
Create new branch from...
Checkout detached...
master 139f6dd8
origin/HEAD Remote branch at 139f6dd8
origin/master Remote branch at 139f6dd8
origin/release Remote branch at 139f6dd8
origin/FixSearchFunctionality Remote branch at 662d9fde

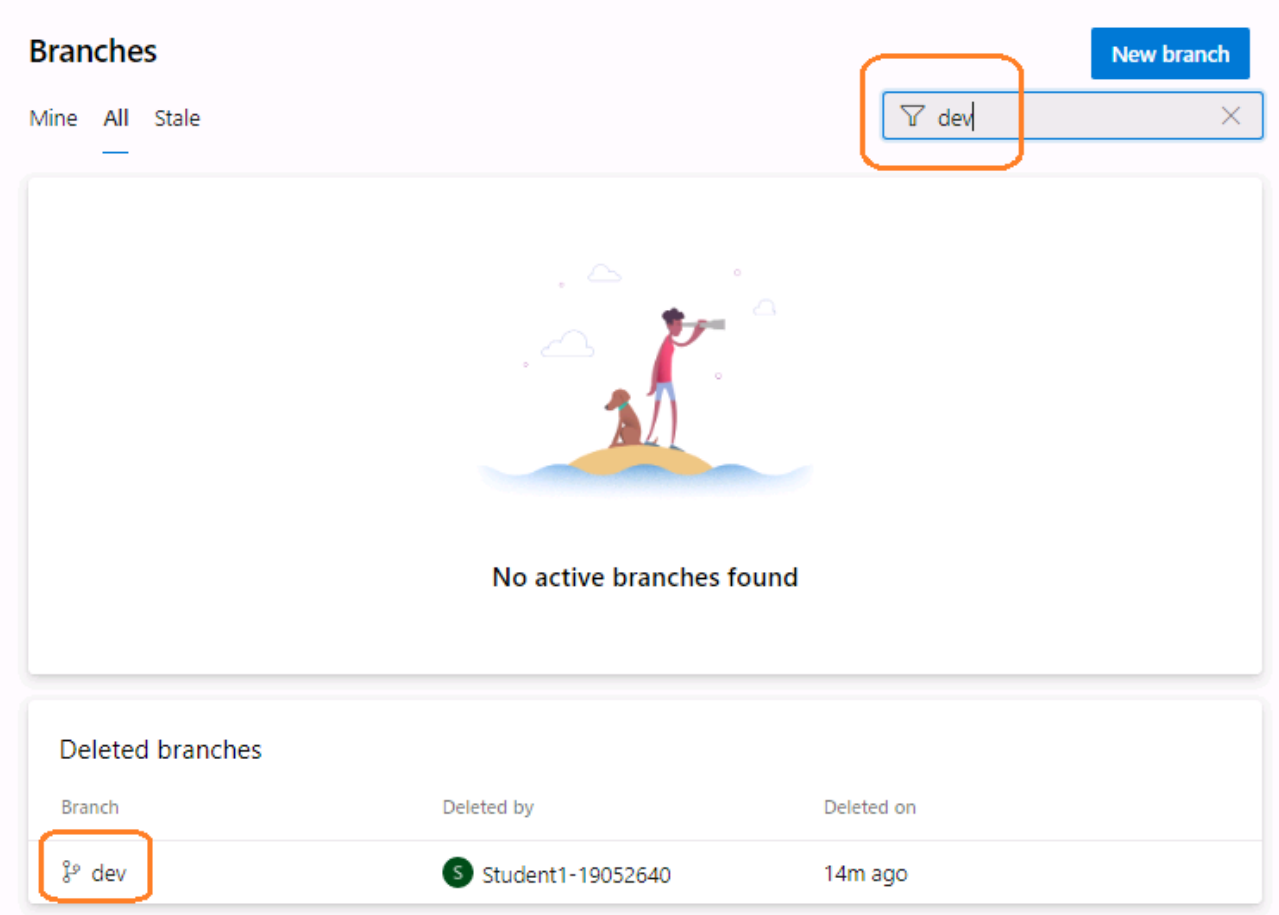
Module 3: **Azure Repos with Git**, **Lab 1: Getting Started with Git**, Exercise 6: Managing Branches from Azure DevOps Services

---

Task 2: Restoring a Branch

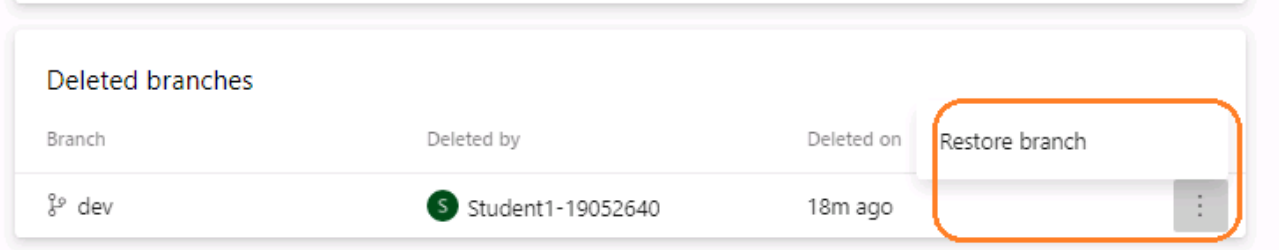
1. Remember that you deleted the **dev** branch earlier from Azure DevOps Services? You can restore this branch, as long as you remember the exact name of the branch.

2. Navigate to **Azure DevOps Services** and **Repos** hub for the PartsUnlimited project. Click on **Branches**. In the **Search branch name**, enter **dev**.



> 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.



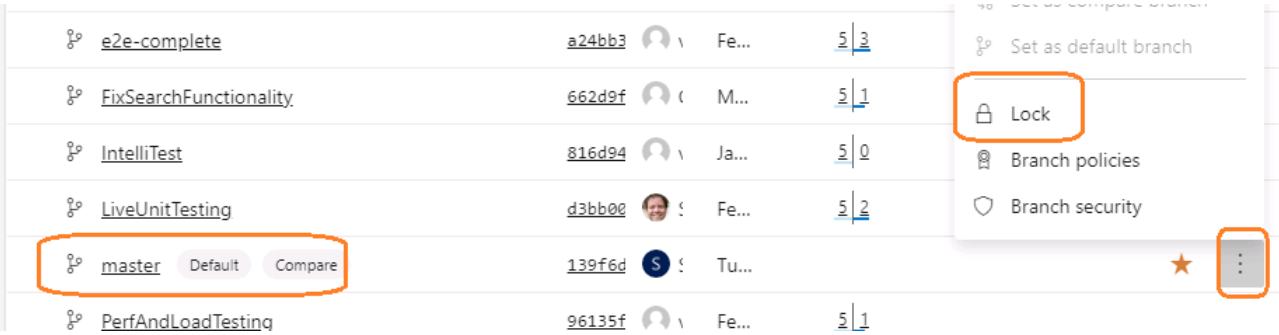
Module 3: **Azure Repos with Git**, **Lab 1: Getting Started with Git**, Exercise 6: Managing Branches from Azure DevOps Services

---

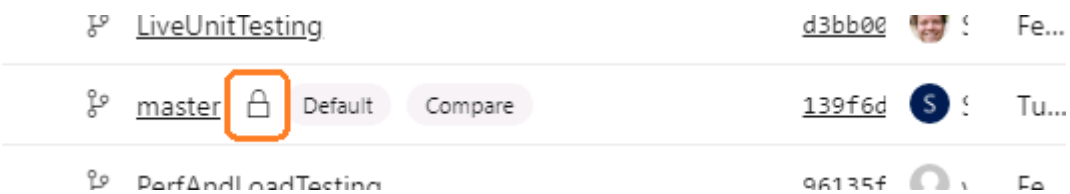
Task 3: Locking a Branch

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**



3. The master branch is now locked.



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

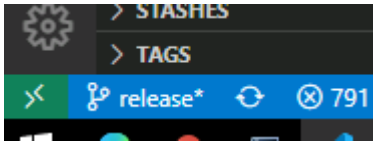
- Task 1: Creating a New Pull Request from Visual Studio Code
- Task 2: Managing Pull Requests
- Task 3: Tagging a Release

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

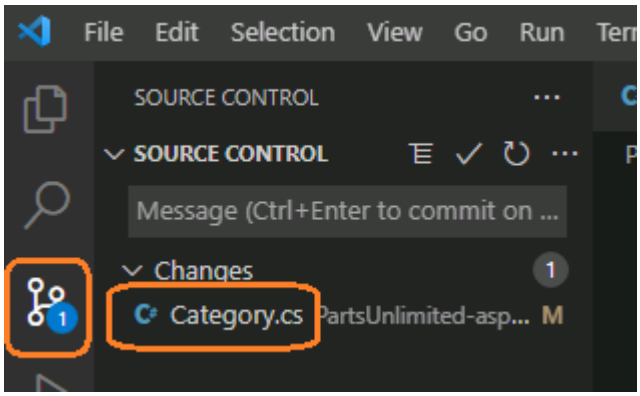
---

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

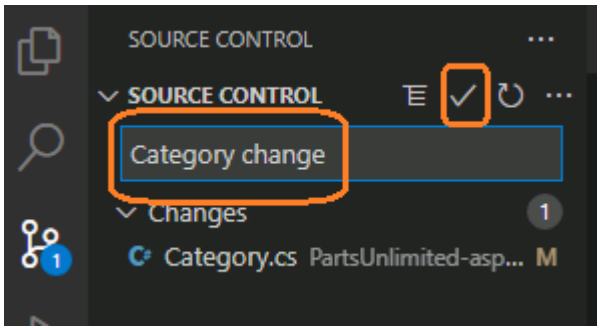
1. Return to **Visual Studio Code** and confirm that you have **release** branch selected.



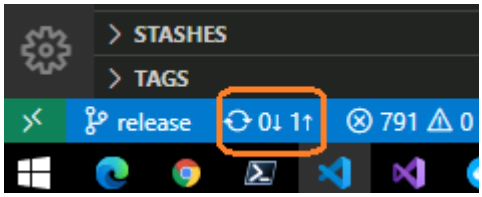
2. Select the **Source Control** tab. It should recognize that you have uncommitted changes to Category.cs.



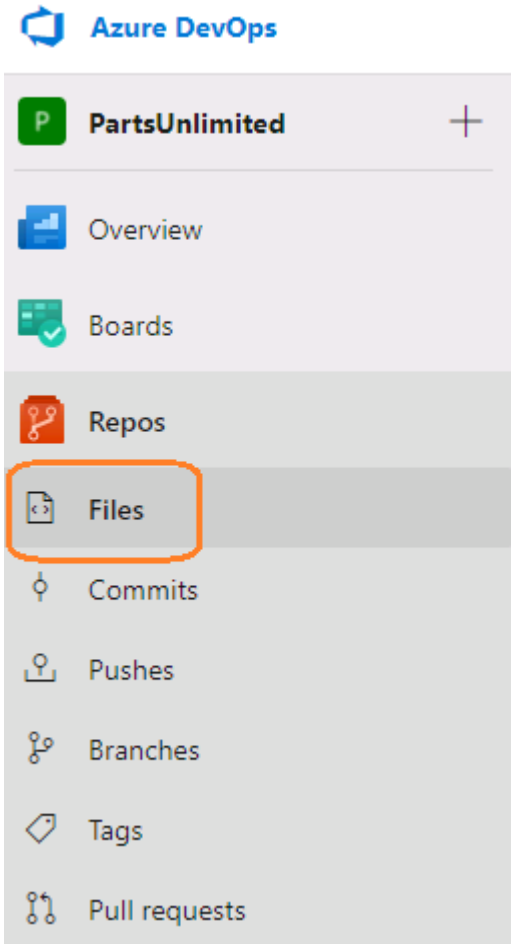
3. Enter a **commit message Category change** and press **Ctrl+Enter** to commit the change to the local release branch. You can also commit by clicking the **commit** button.



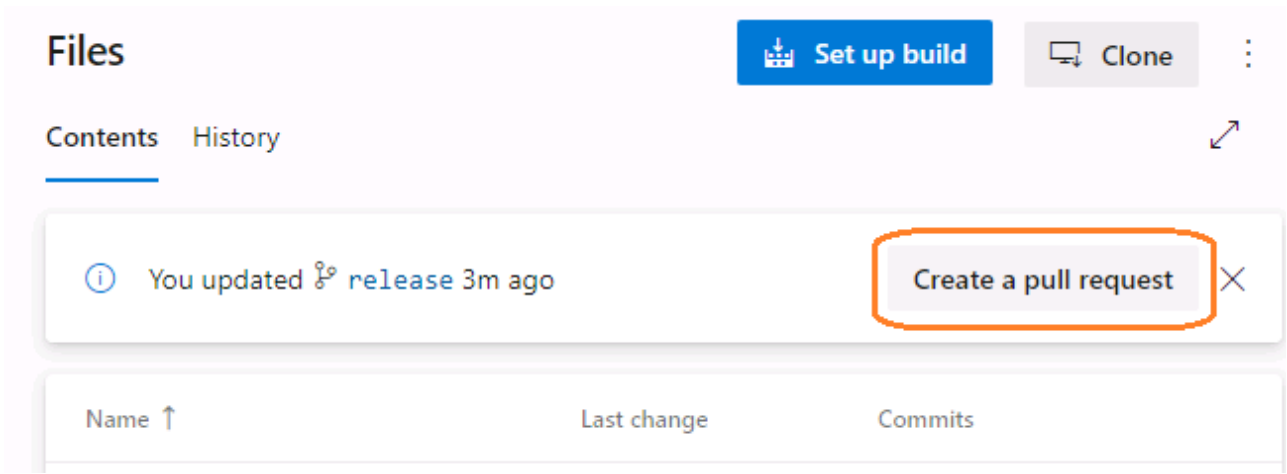
4. Click the **Synchronize Changes** button at the bottom-left corner of the VS Code to push the commit to the remote repository.



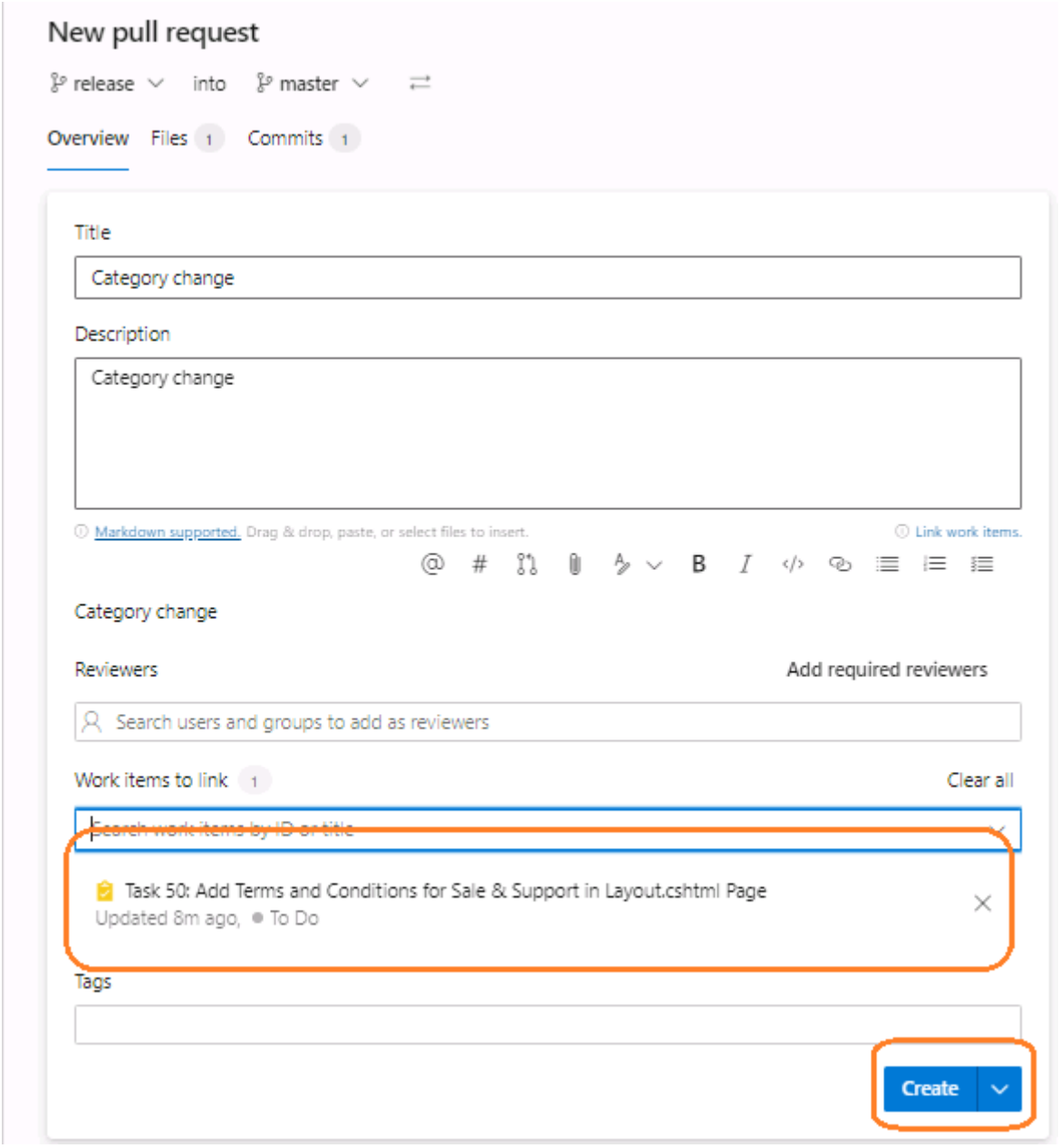
5. Navigate back to the web browser with Azure DevOps Services open. Click on the **Files** tab from **Repos**.



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



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**.

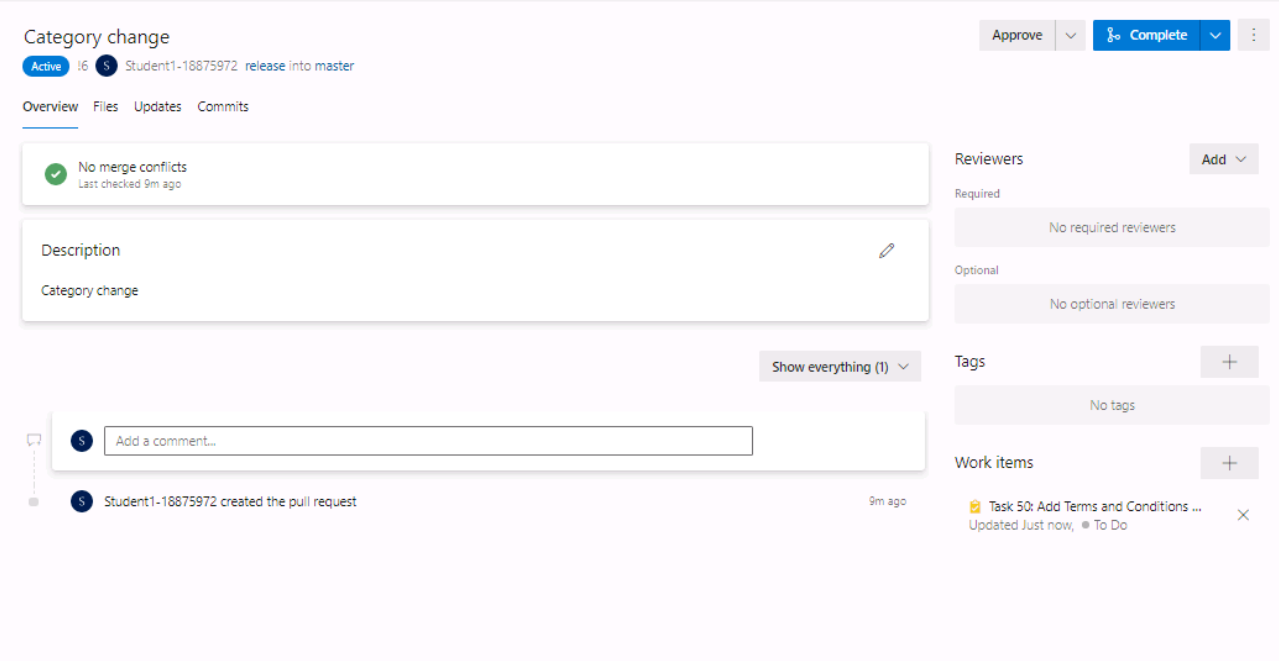

New pull request
release
into
master
Overview
Files 1
Commits 1
Title
Category change
Description
Category change
Markdown supported. Drag & drop, paste, or select files to insert.
Link work items.
Category change
Reviewers
Add required reviewers
Search users and groups to add as reviewers
Work items to link 1
Clear all
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

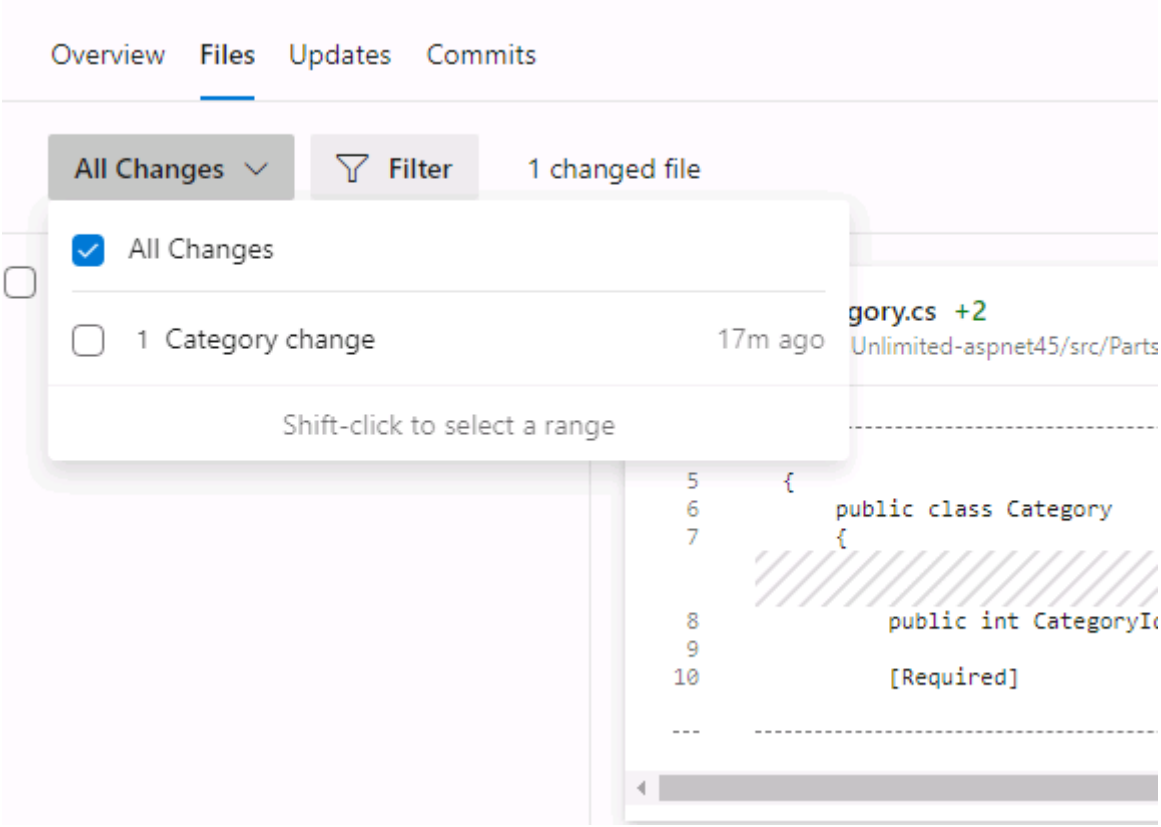
1. The **Overview** tab of the Pull Request contains all of the key information specified in the creation form, as well as options to approve and complete the request.



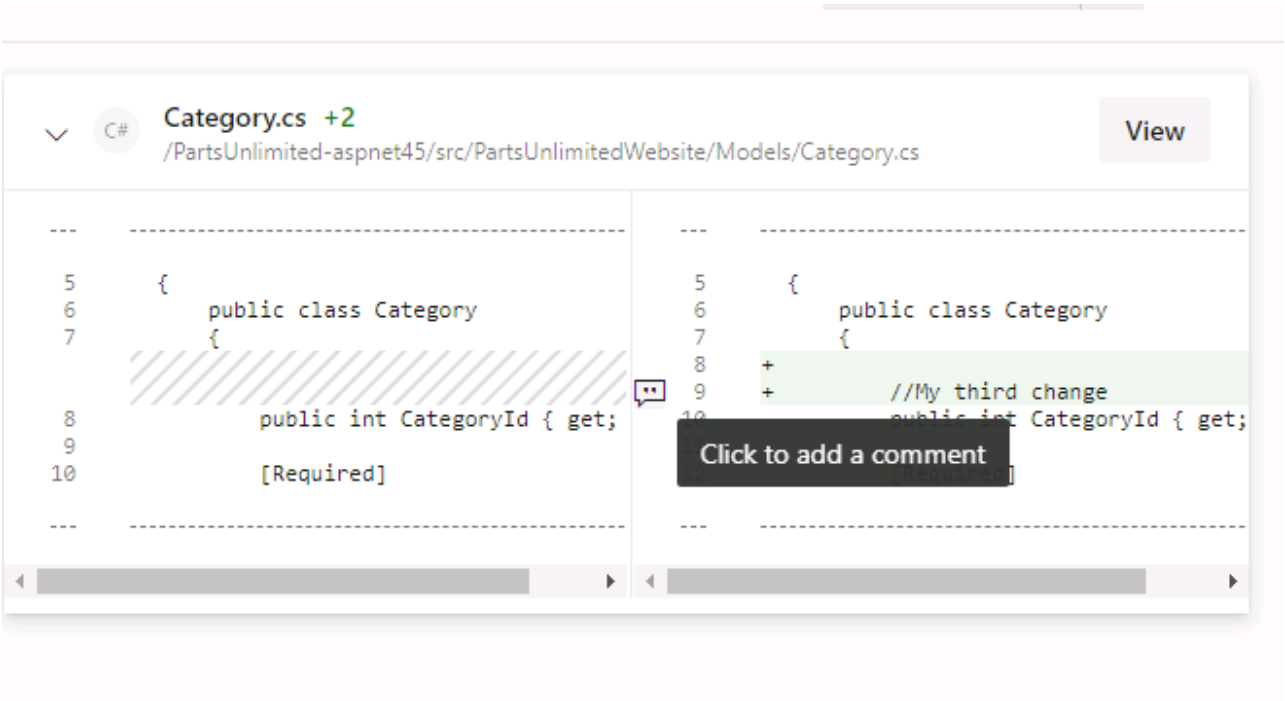
2. Select the **Files** tab of the Pull Request to review the files involved in the commit.

> 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.

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

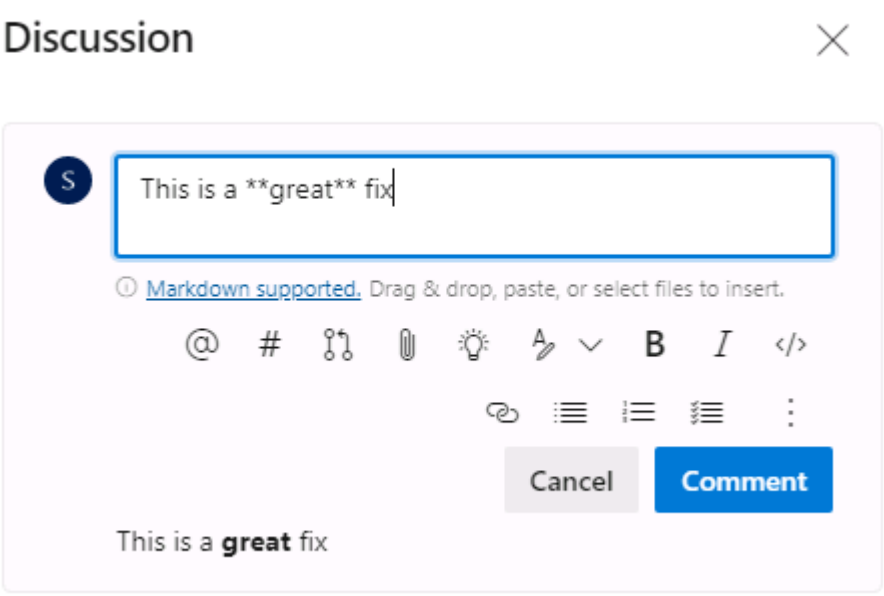


4. You can add a comment in the file that is displayed in the Pull Request. Click on **Click to add comment**.

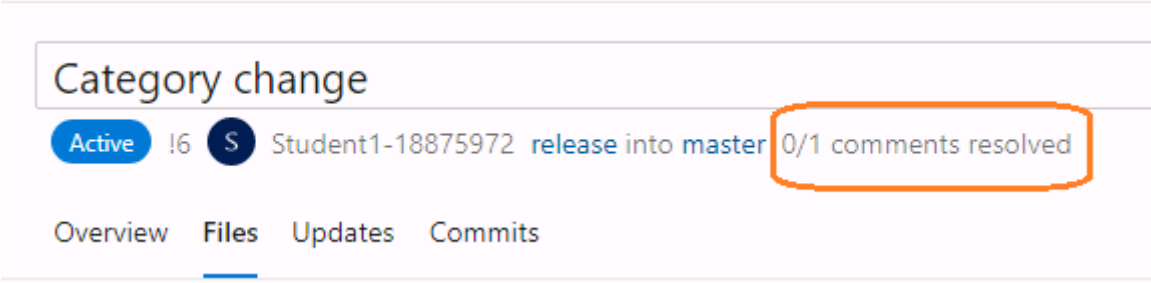


5. Enter a comment using markdown **This is a *_great_* fix** and click **Comment** to save it. Close the **Discussion** pane.

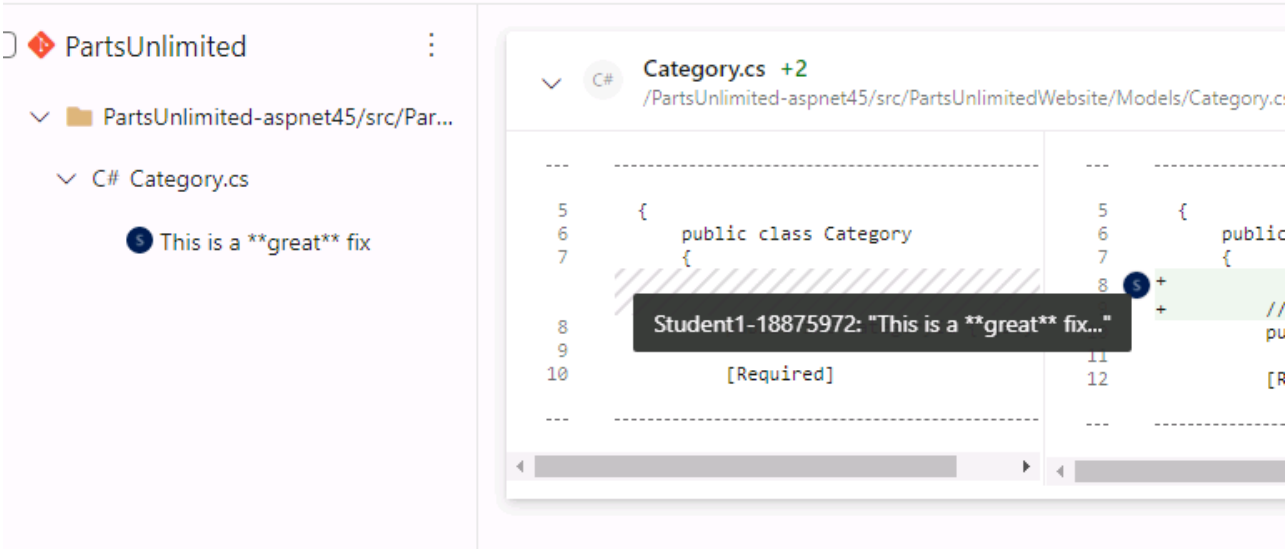
> Note the live preview of your comment before you commit to it.

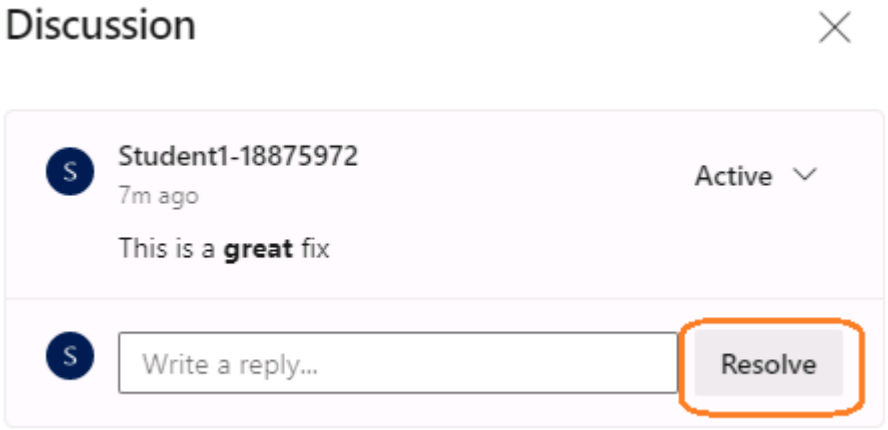


6. The new comment is tracked as part of the pull request. It's expected that every comment will be resolved before a pull request will be completed, so this convenient marker lets you know if there's anything else that needs to be reviewed.

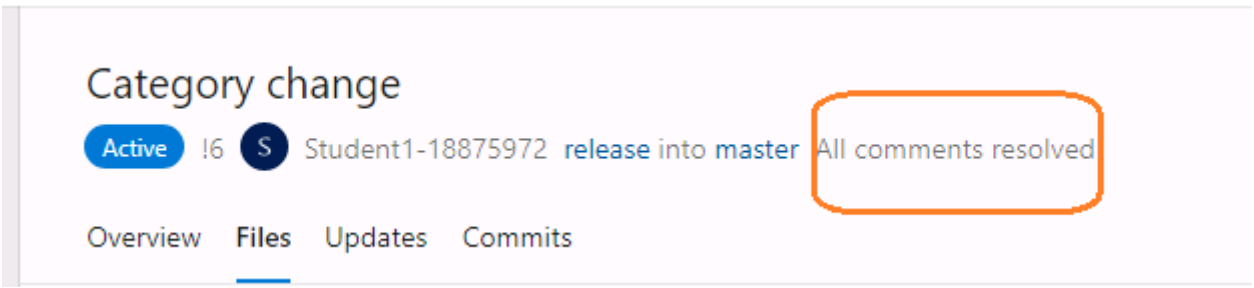


7. The comment is also placed in line with the code. **Click** on the comment to start the discussions about lines and sections of code within their proper context. Click **Resolve** to complete the discussion and **close** the Discussion pane.





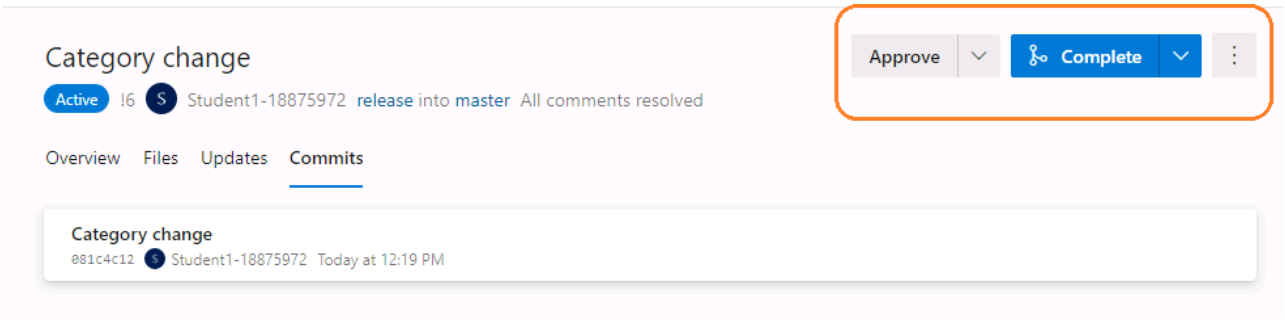
8. Note that this now updates the tracking as well.



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

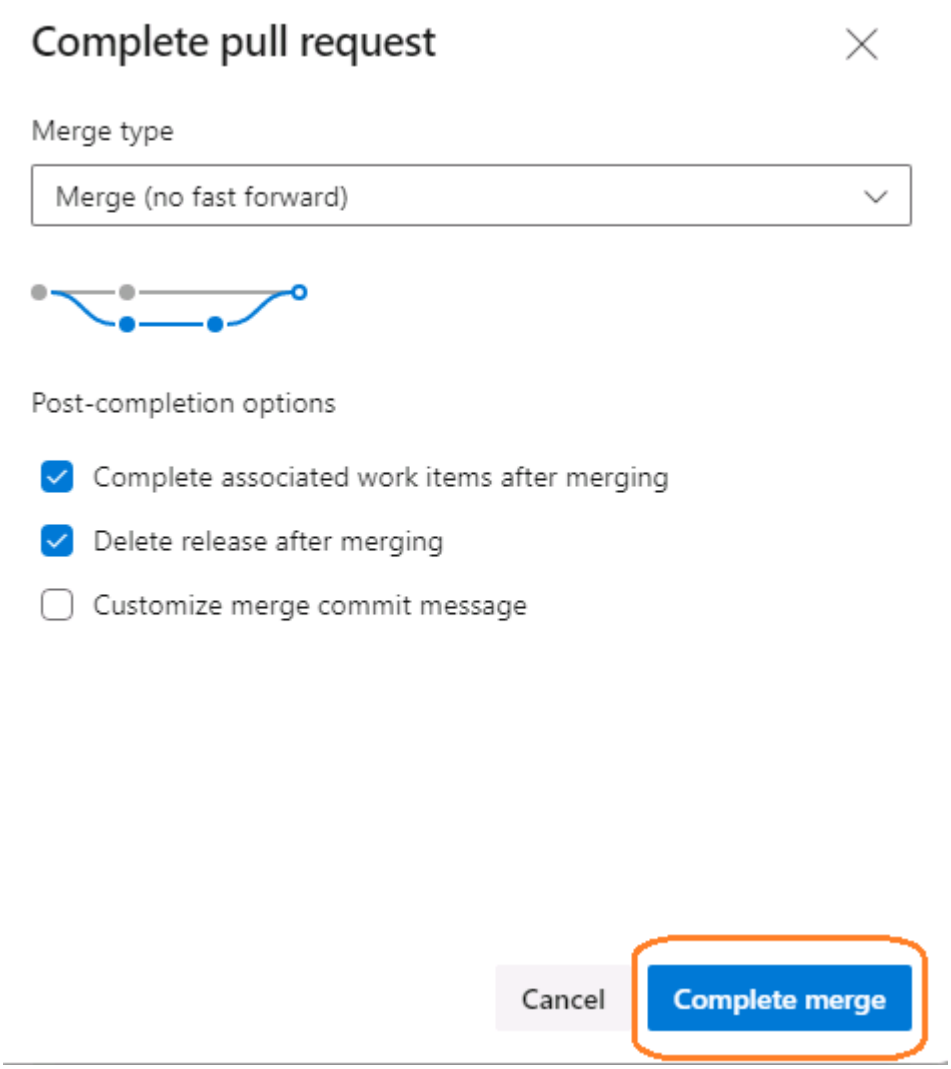
10. Select the **Commits** tab, where you can review the commits made to the branch.

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

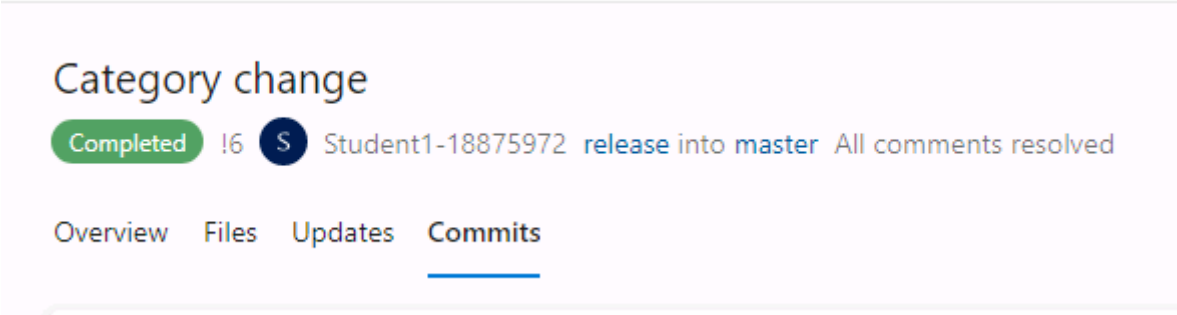


12. You can accept the default options in the **Complete pull request** dialog. The first option is to complete the work items linked to the branch being merged. The second checkbox indicates that the release branch will deleted after merging. Click **Complete merge**.

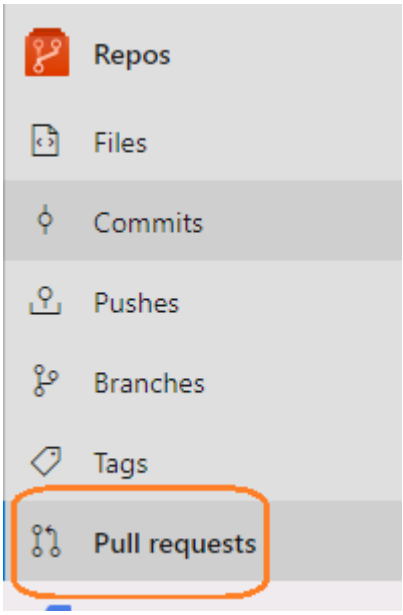
> Please see Complete, abandon, or revert pull requests - Azure Repos for different Merge Type options.



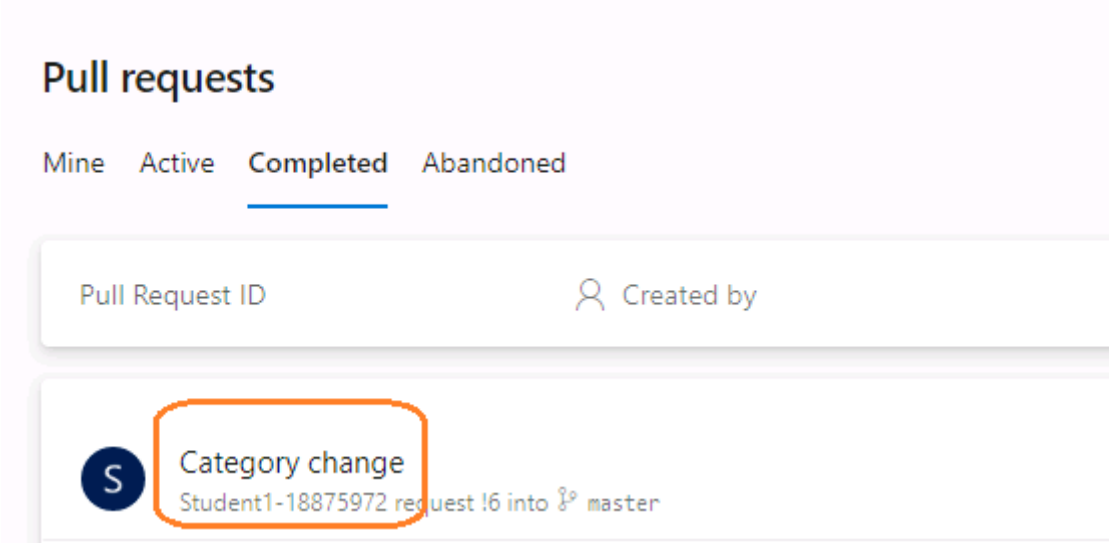
13. When the merge completes, the pull request should be marked as *Completed*.



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

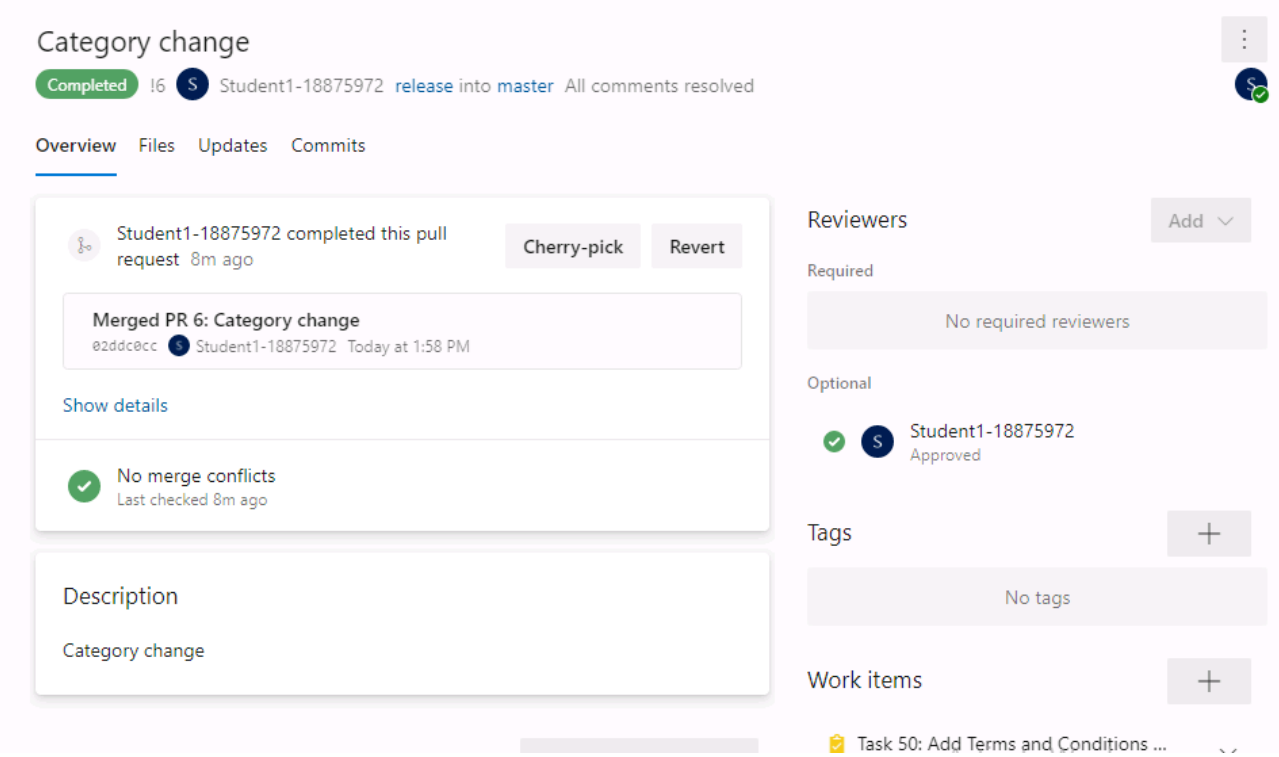


15. Select the **Completed** tab and click the pull request as though you were visiting it fresh.

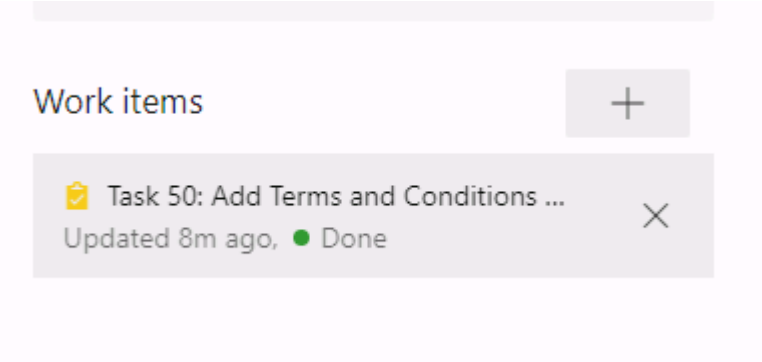


16. **Click** on the pull request to review it. You can *Cherry-pick* or *Revert* if needed.

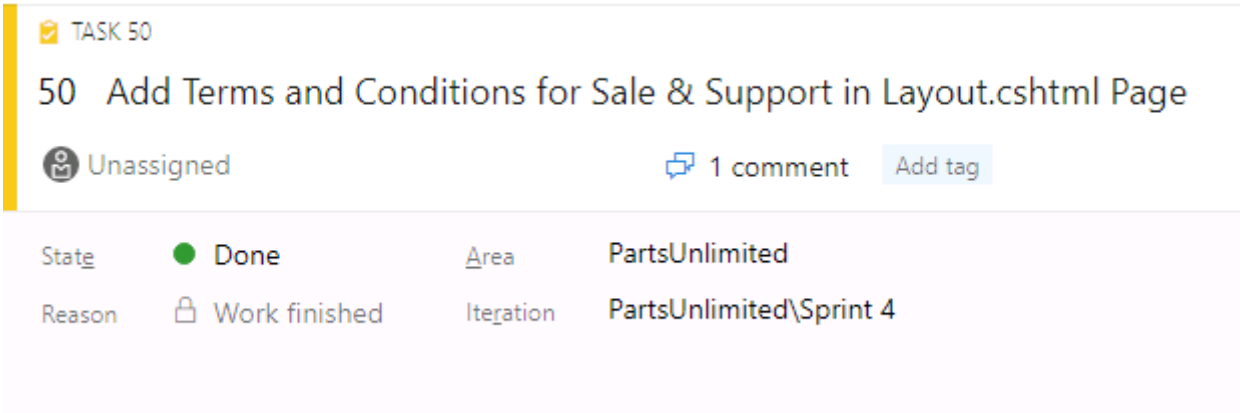
> Cherry-picking is the process of selecting specific commits from one branch to apply to another, conceptually similar to a copy/paste operation.



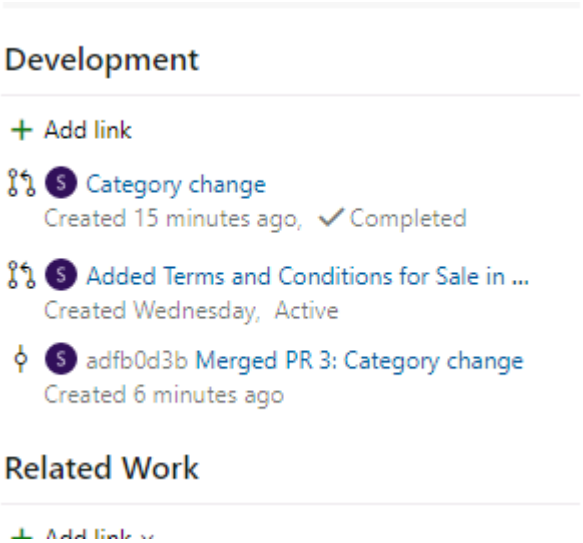
17. Under Work Items, click one of the linked work items.



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



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

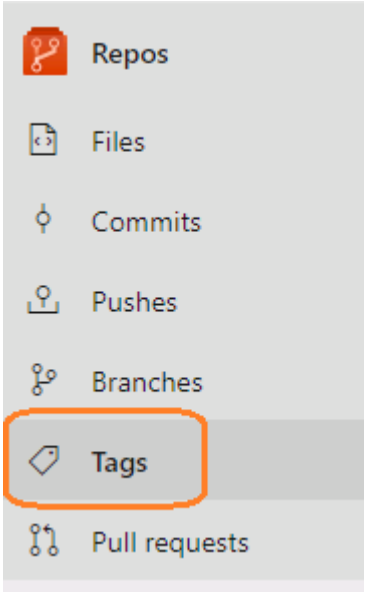

Development
+ Add link
Category change
Created 15 minutes ago, Completed
Added Terms and Conditions for Sale in ...
Created Wednesday, Active
adfb0d3b Merged PR 3: Category change
Created 6 minutes ago
Related Work

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

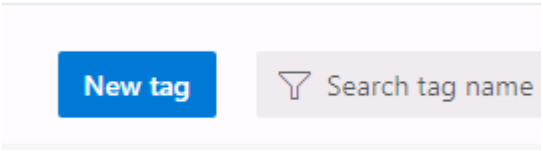
---

Task 3: Tagging a Release

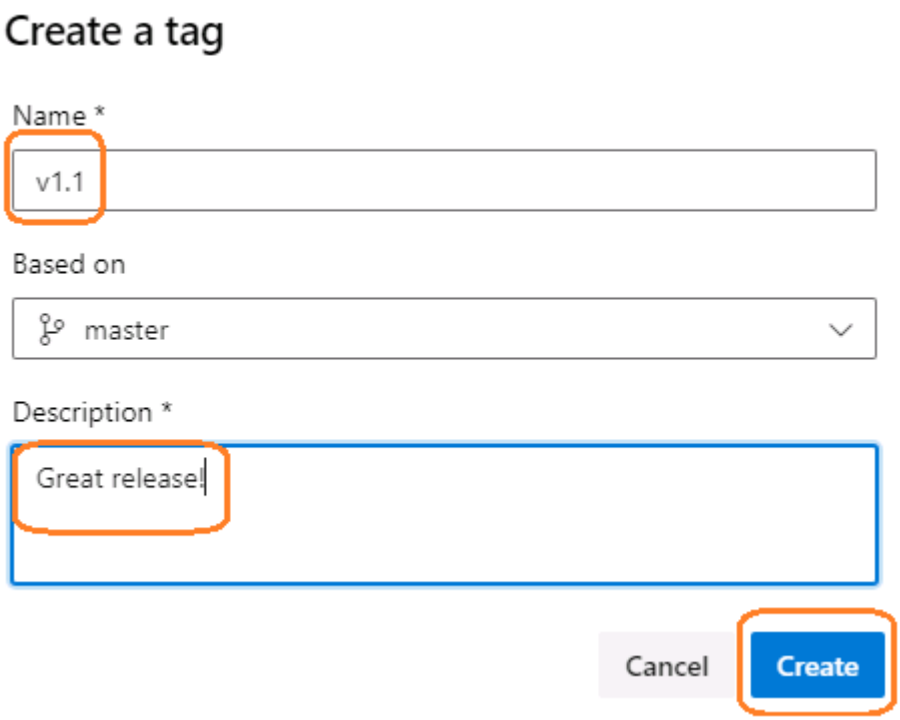
1. Let's assume that this version of the site is exactly what's needed for v1.1. You can add a tag mentioning this. Navigate to **Repos** hub and click on **Tags** tab.



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**.



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

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

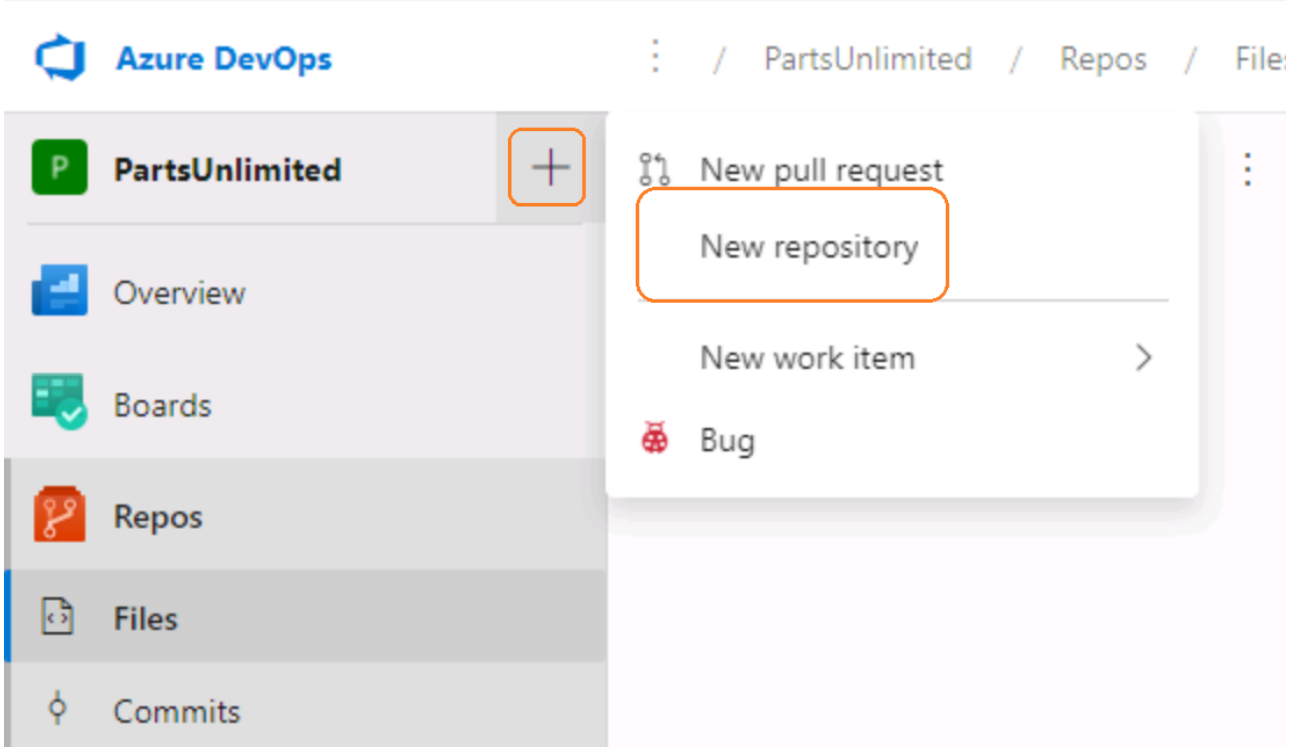
---

Task 1: Creating a New Repo from Azure DevOps Services

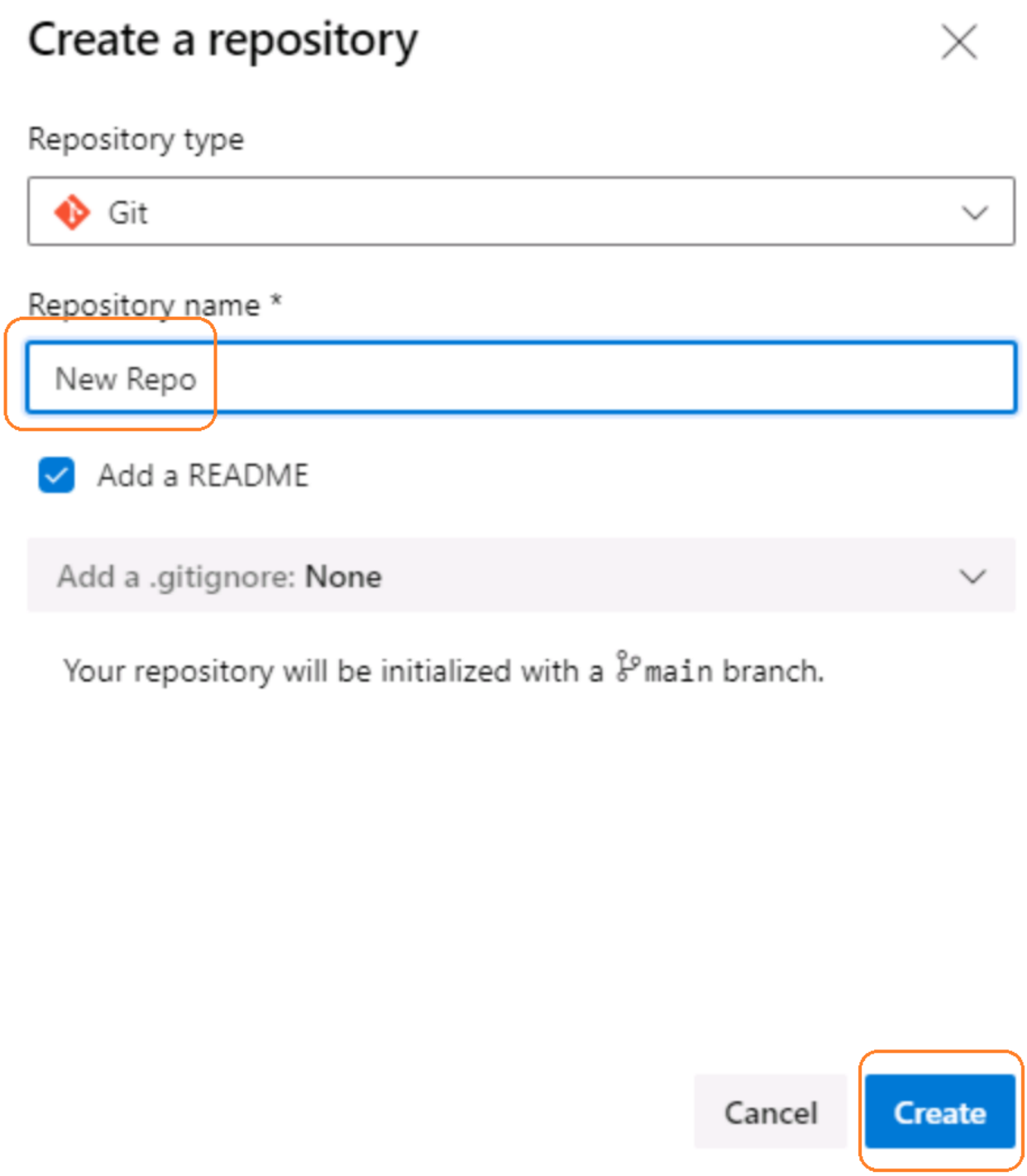
1. In the web browser, navigate to **Azure DevOps Services**. From the **PartsUnlimited** project **Add** dropdown, select **New repository**.

   > If you don't see the **New Repositoy** 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 / Partslimitted** from the top-center and select **+ New repository**



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



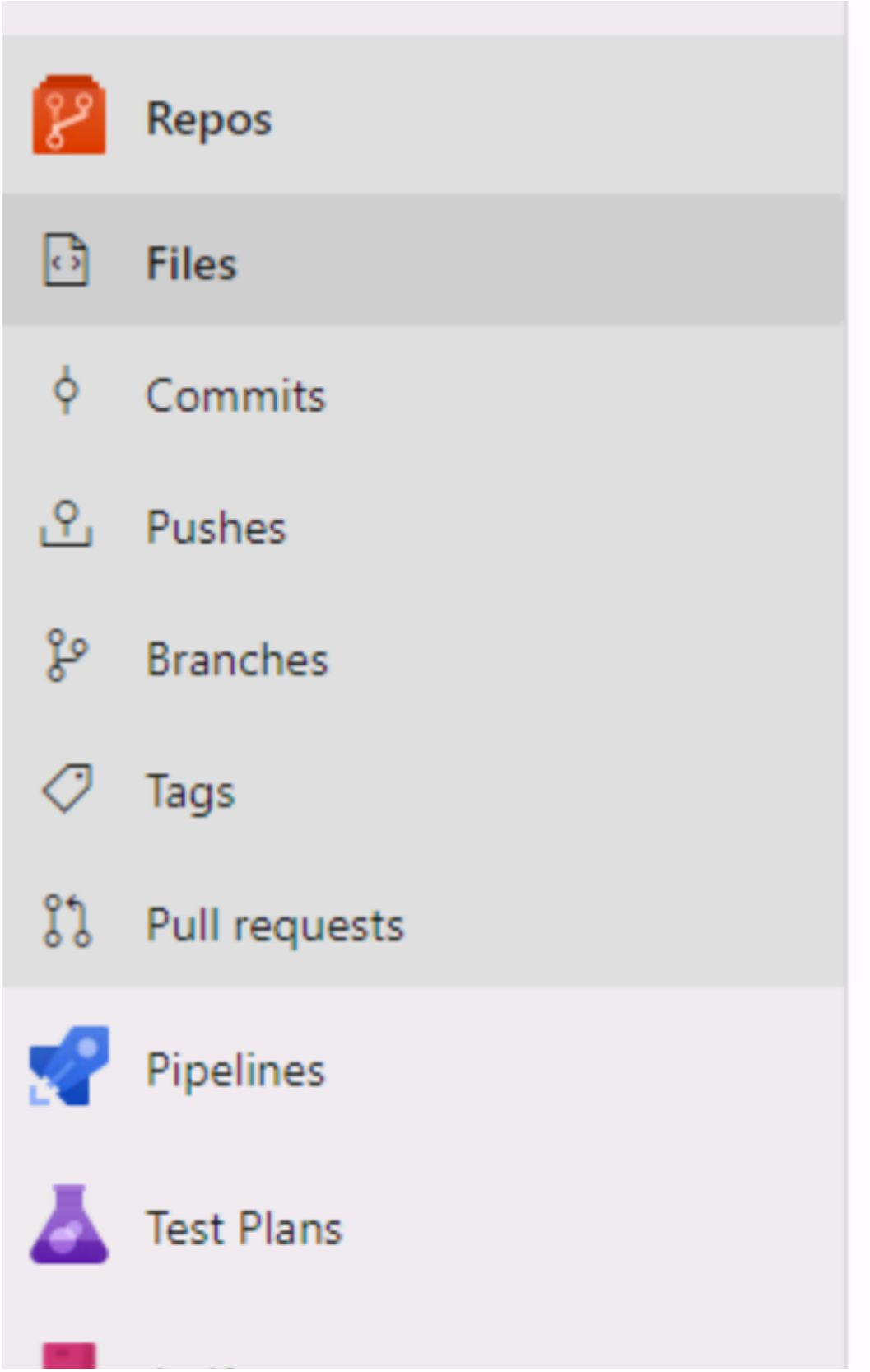
3. That's it. Your repo is ready. You can now clone it with Visual Studio Code or tool of your choice.

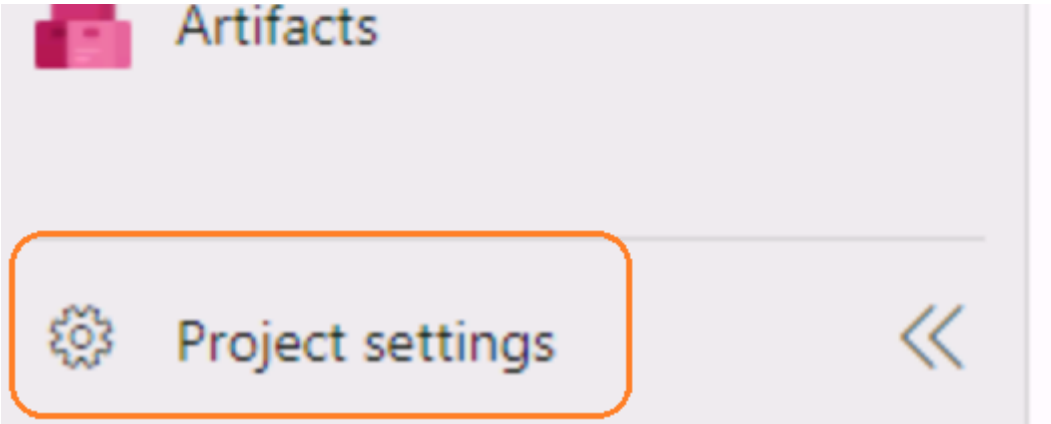
Module 3: **Azure Repos with Git**, **Lab 1: Getting Started with Git**, Exercise 8: Managing Repositories in Azure Repos

---

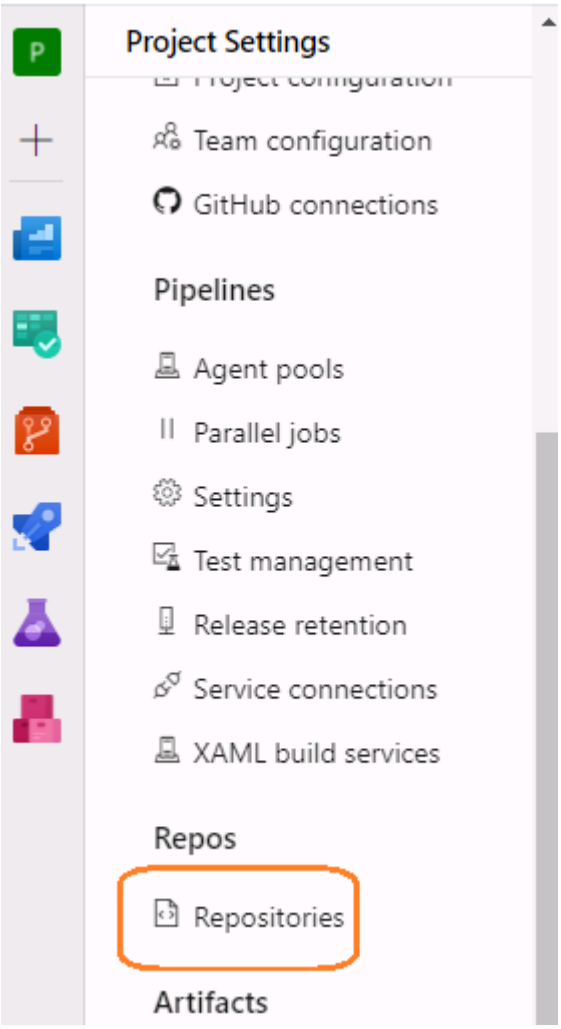
Task 2: Deleting and Renaming Git Repos

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.

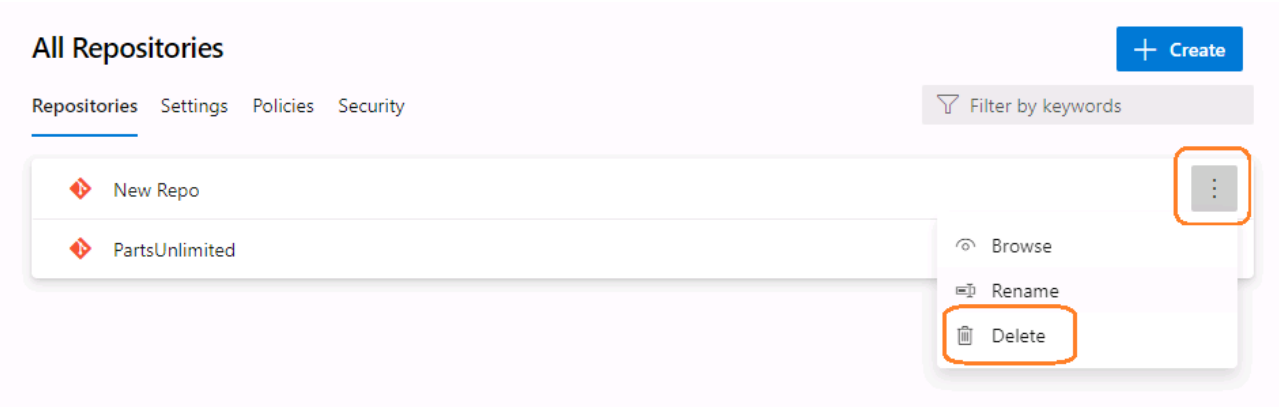




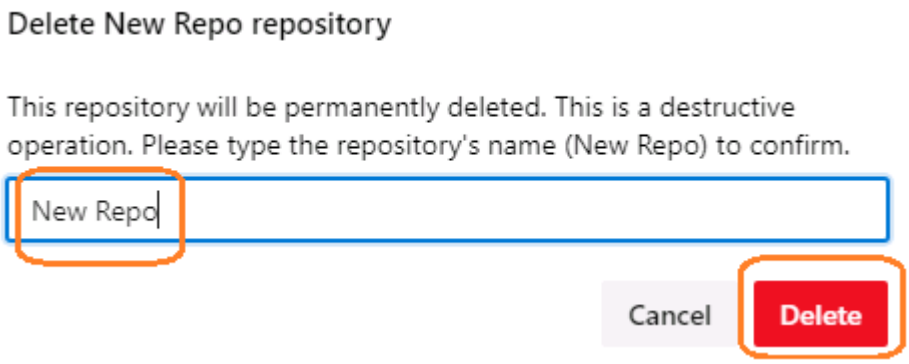
2. Select **Repositories** under **Repos**.



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.



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



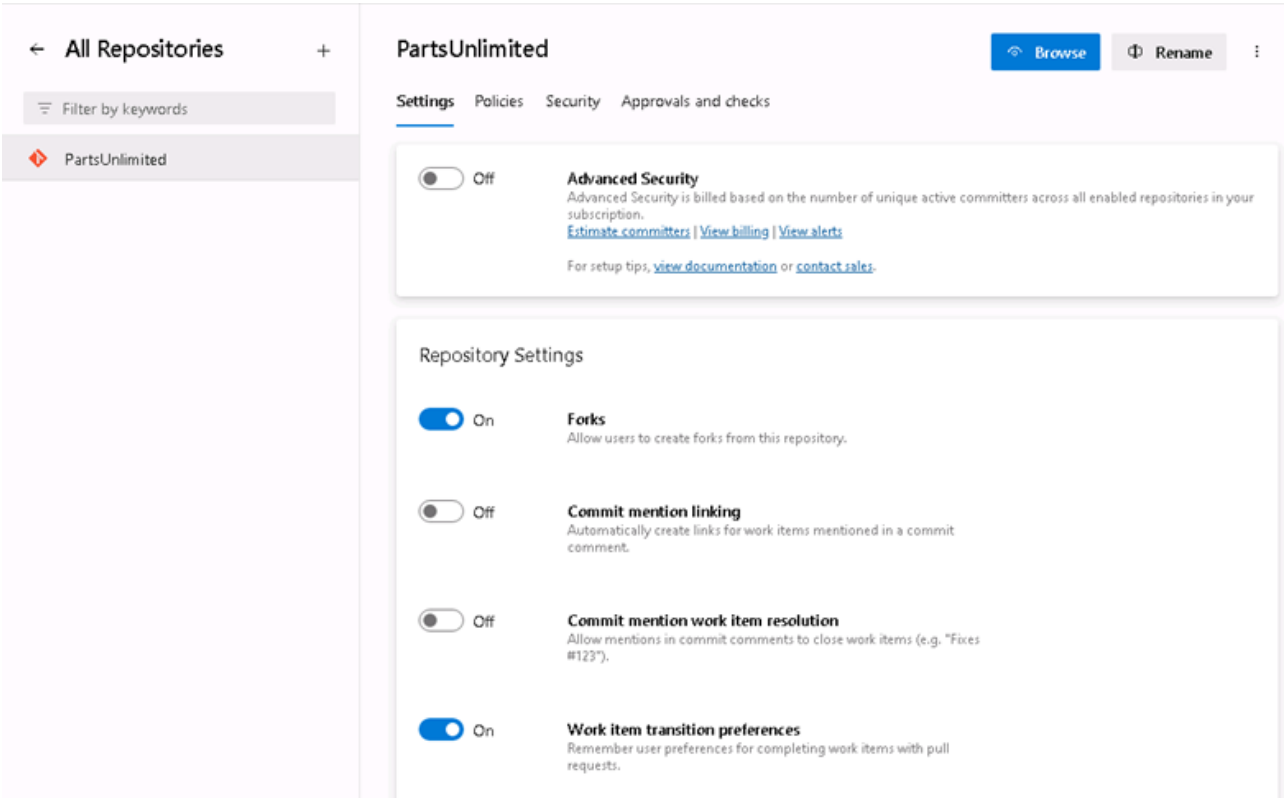
Module 3: **Azure Repos with Git**, **Lab 1: Getting Started with Git**, Exercise 8: Managing Repositories in Azure Repos

---

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.

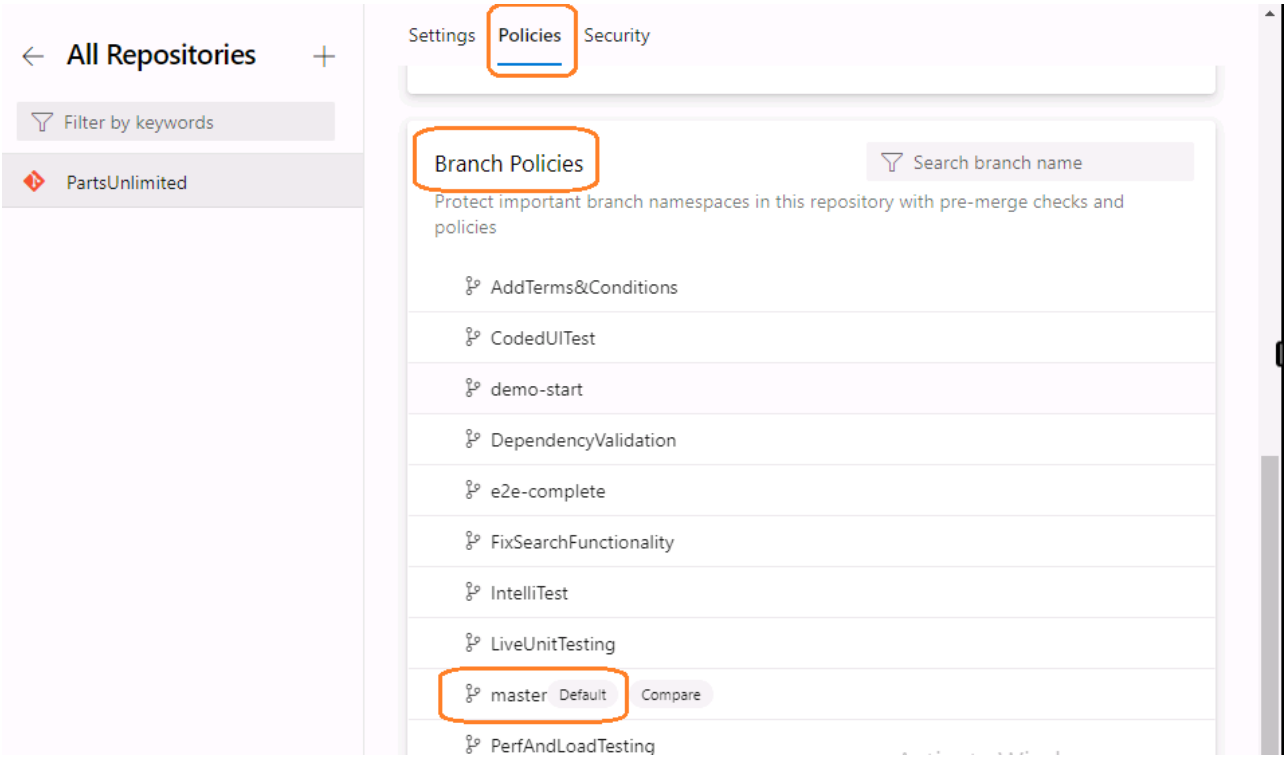


2. Click **Policies**. You can define a wide variety of policies for the repositories and branches to enforce.

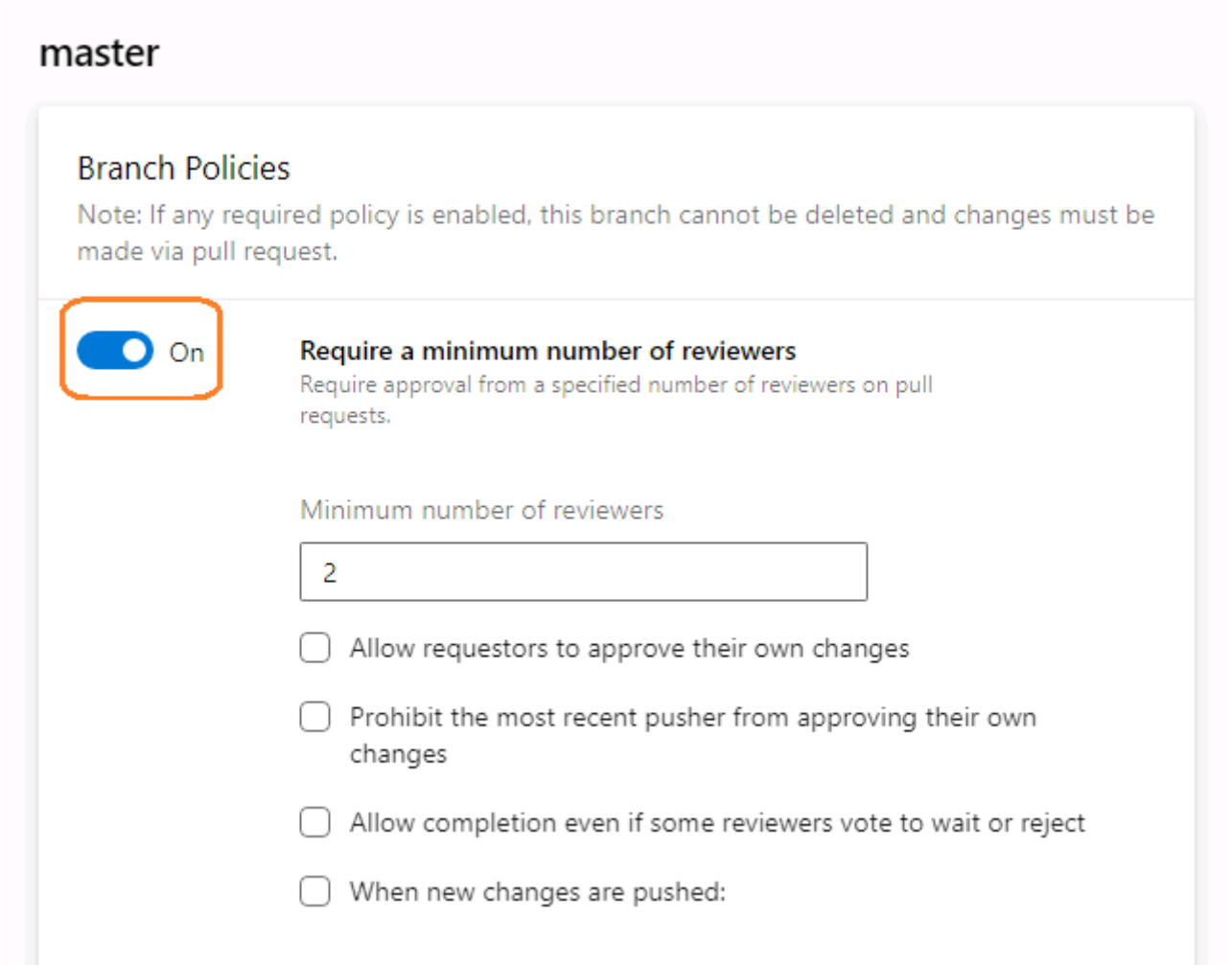
   > Azure DevOps Services branch policies are very effective in enforcing a level of quality control in the repo.

   > 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.

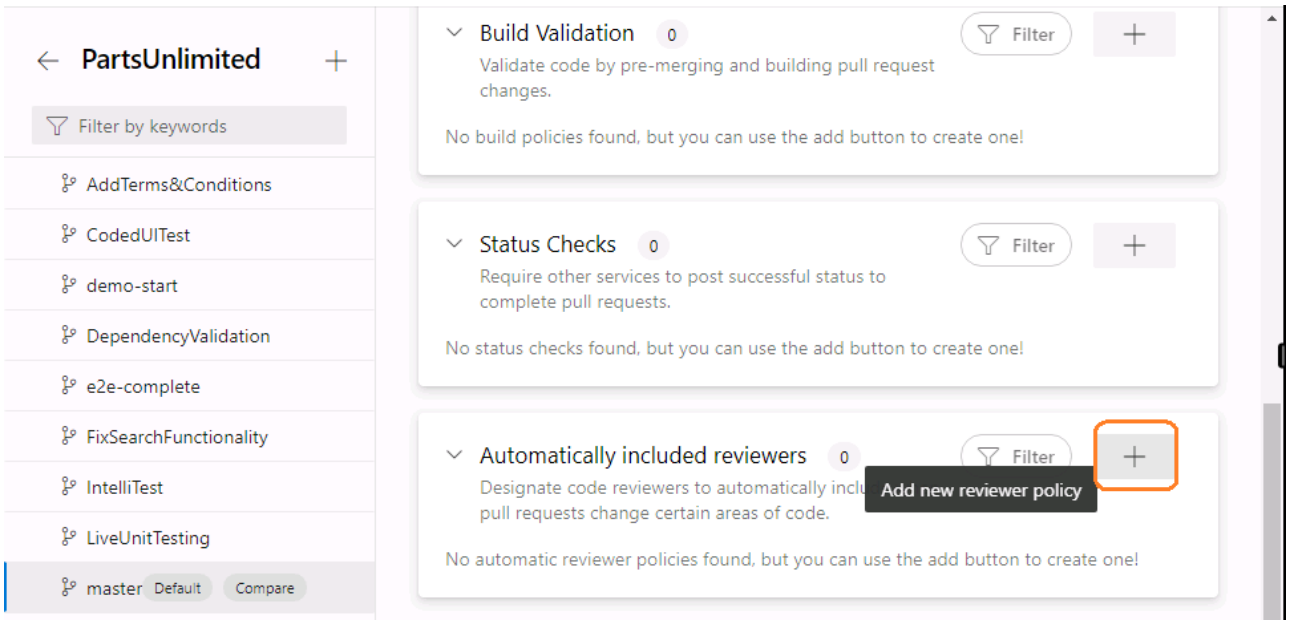
3. **Scroll down** to select **master** branch from **Branch Policies**.



4. 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.

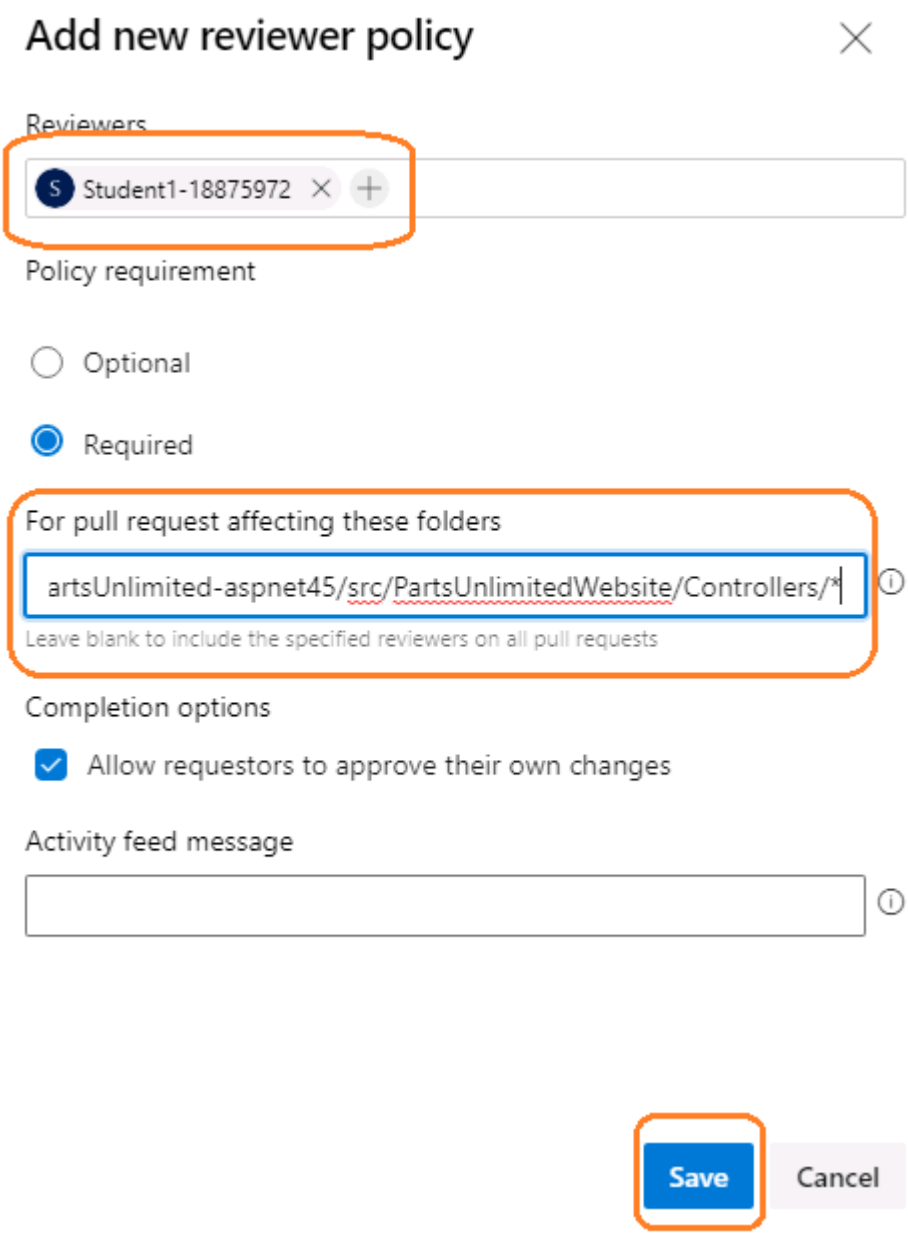


5. **Scroll down** and look for **Automatically included reviewers** section. Click **+** to **Add new reviewer policy**.



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

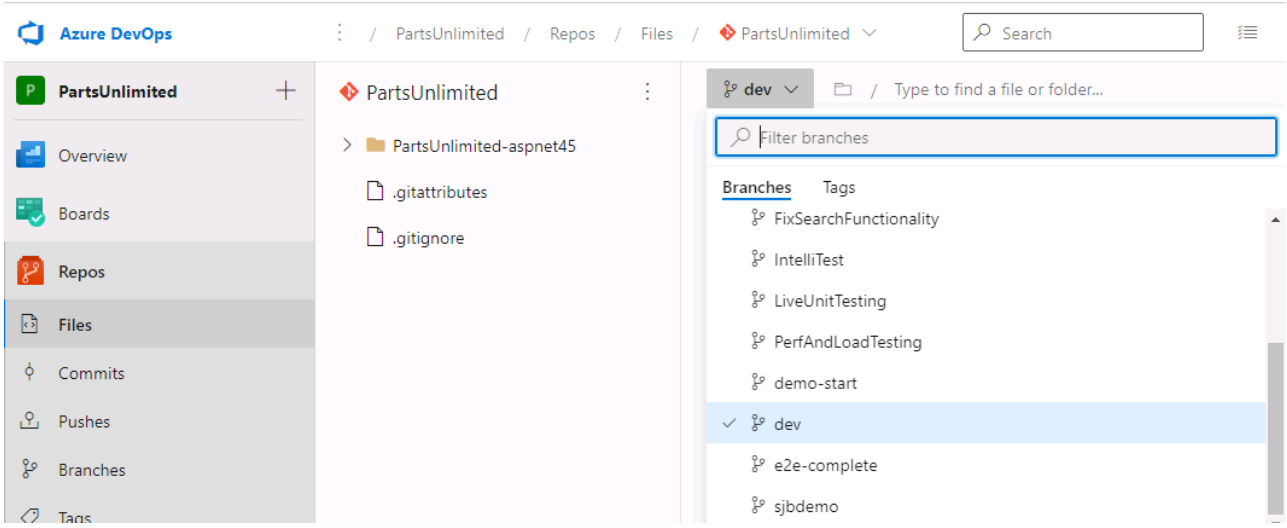
    - **Reviewers**: Add yourself as the reviewer. If asked to authenticate, enter your credentials.
    - **Policy requirement**: **Required**. This ensures that above reviewer will be required to sign off on any changes propsed in the path mentioned below.
    - **For pull request affecting these folders**: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



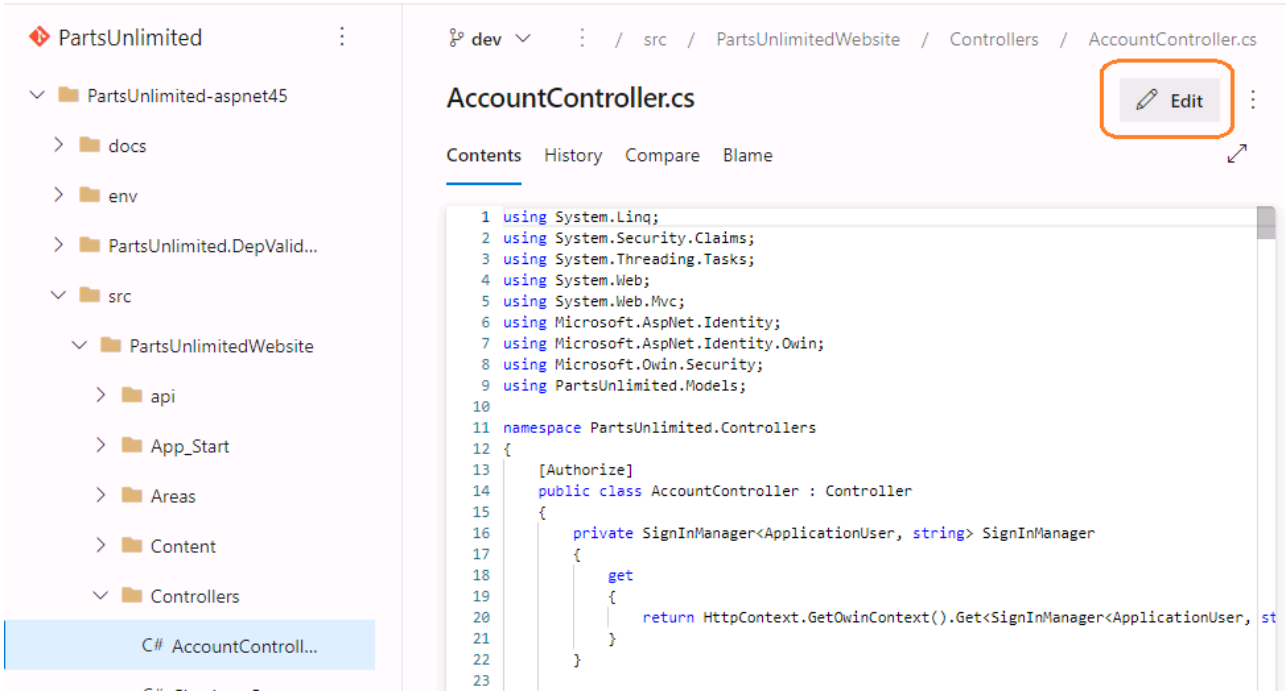
> You will now test the **Automatically included reviewer** policy we implemented above.

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

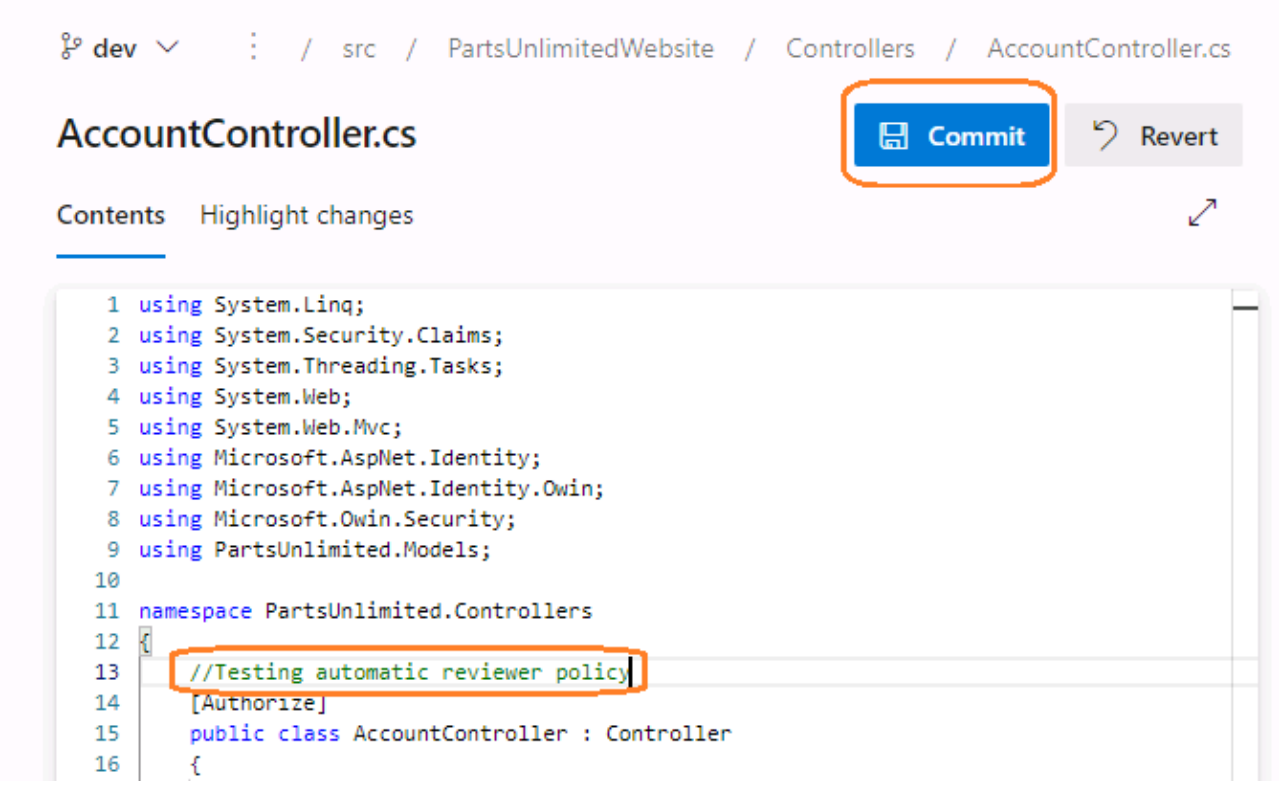
> If **Repository "New Repo" not found** message is displayed, click on **Switch to the default PartsUnlimited repository**.



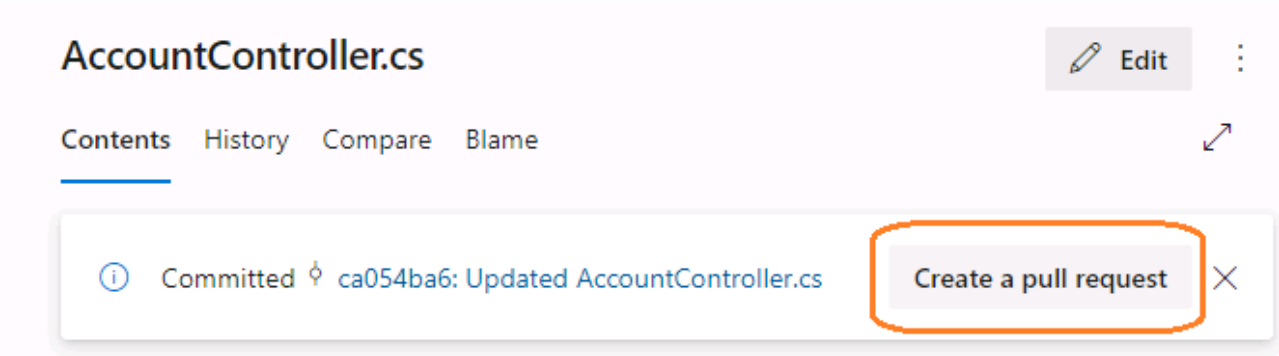
8. Navigate to **PartsUnlimited-aspnet45/src/PartsUnlimitedWebsite/Controllers/AccountController.cs** and click **Edit**.



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

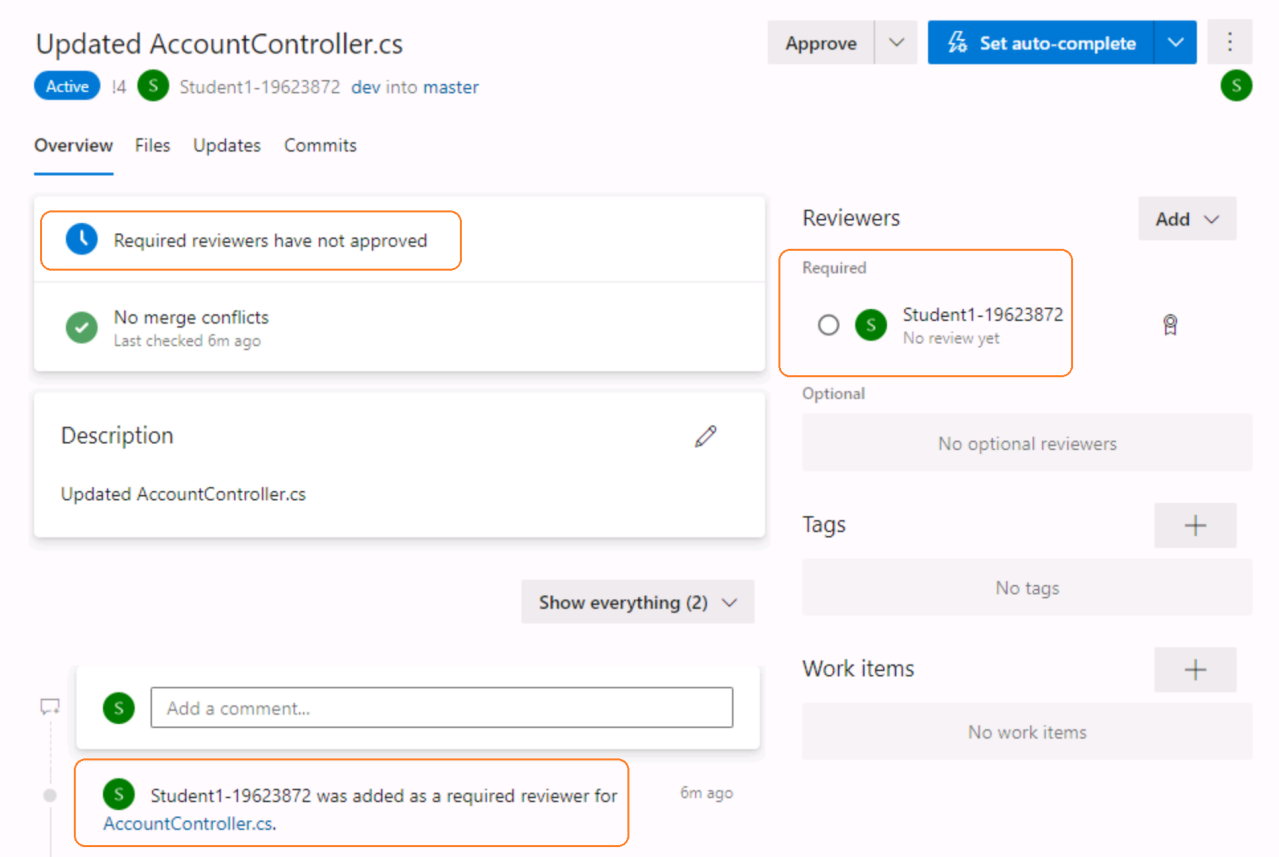


10. Click on the **Create a pull request** to create a new pull request to merge this change to the master branch.

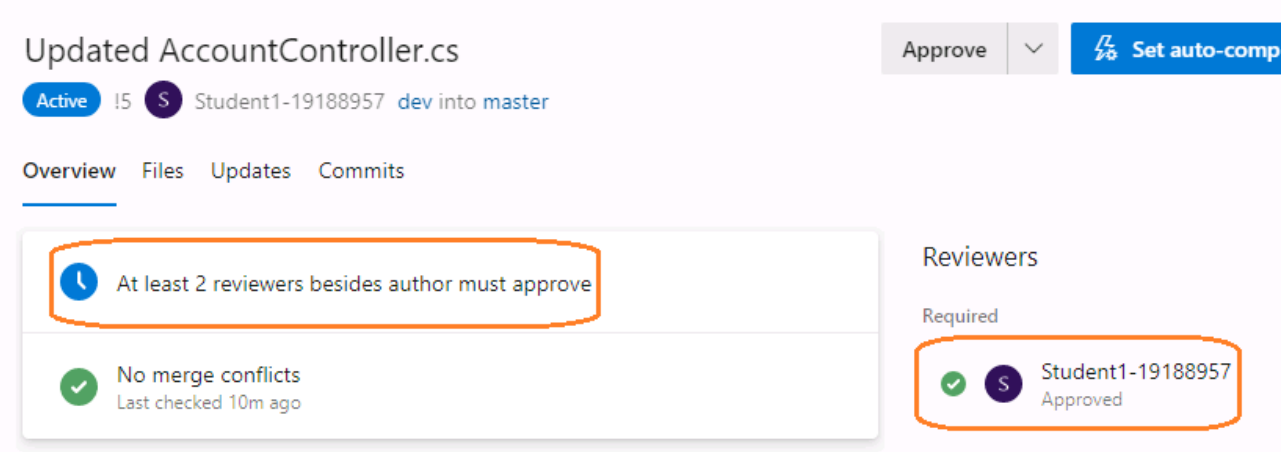


11. Click **Create** to create a pull request with the default values.

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.



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*.



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

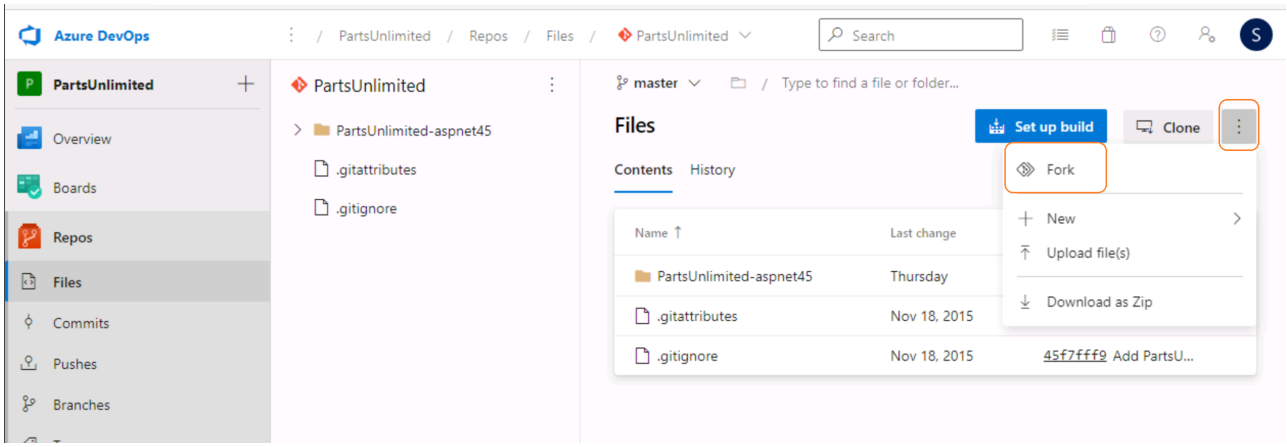
Module 3: **Azure Repos with Git**, **Lab 1: Getting Started with Git**, Exercise 8: Managing Repositories in Azure Repos

---

Task 4: Forking Repos

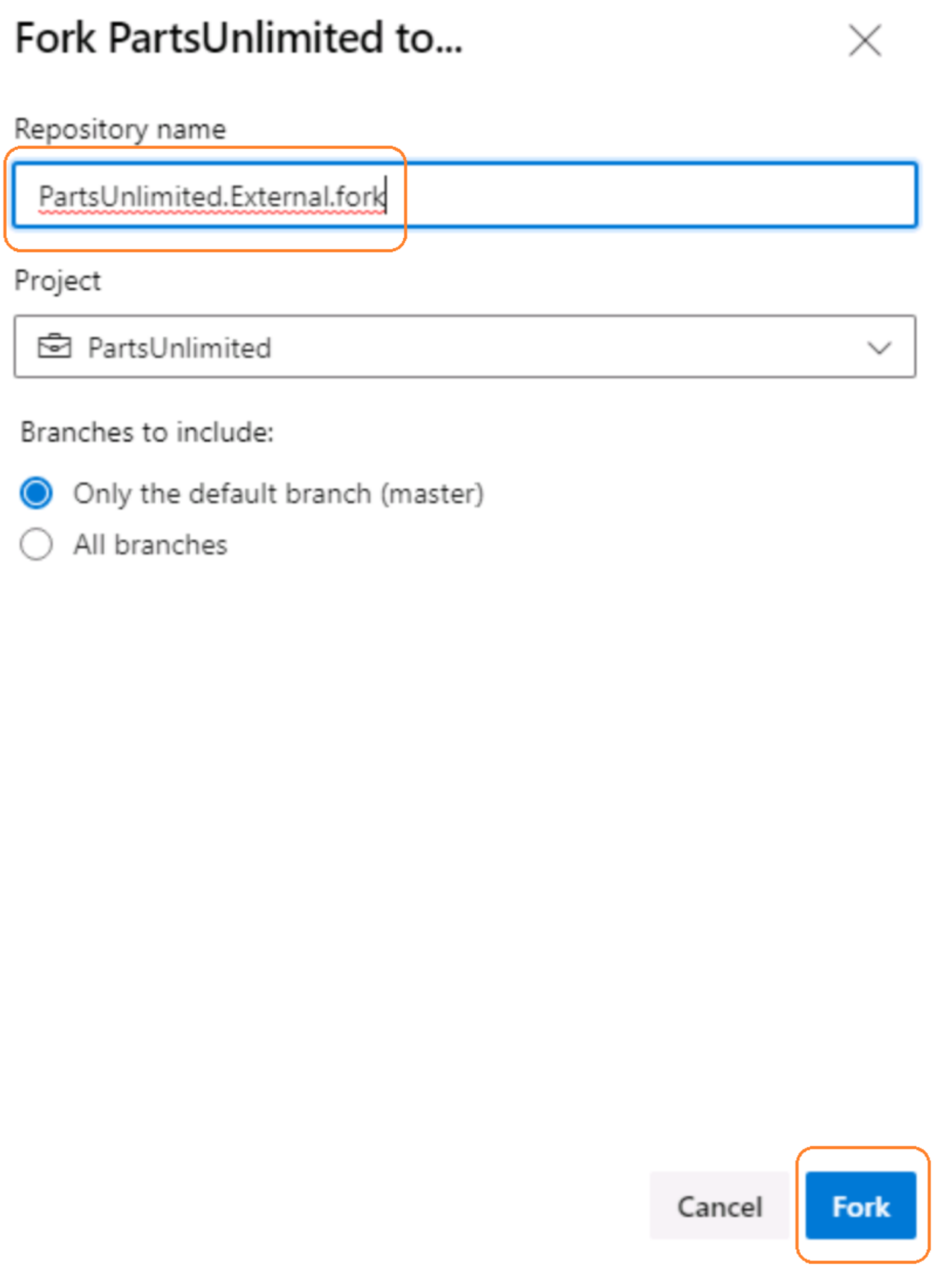
1. Click on **Repos | Files**.

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.



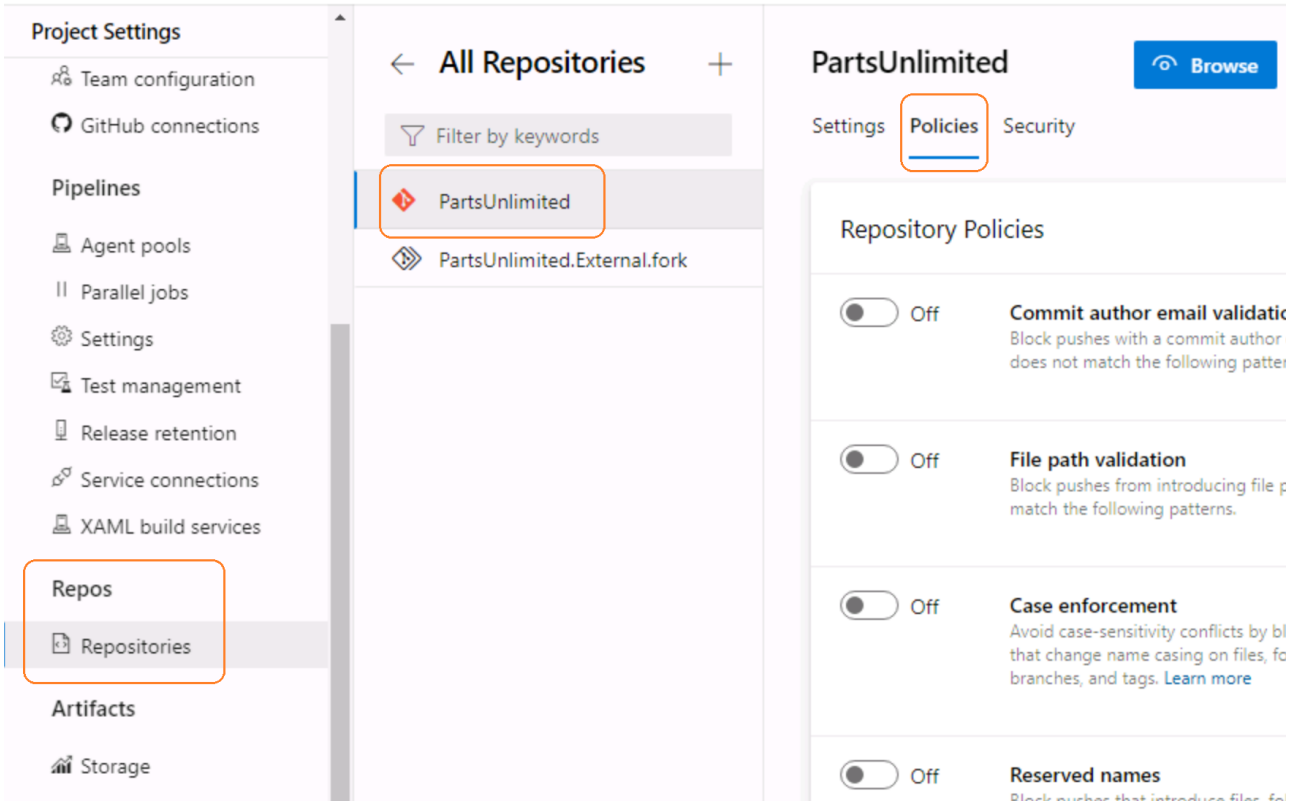
4. You can now work with the repo you forked.

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

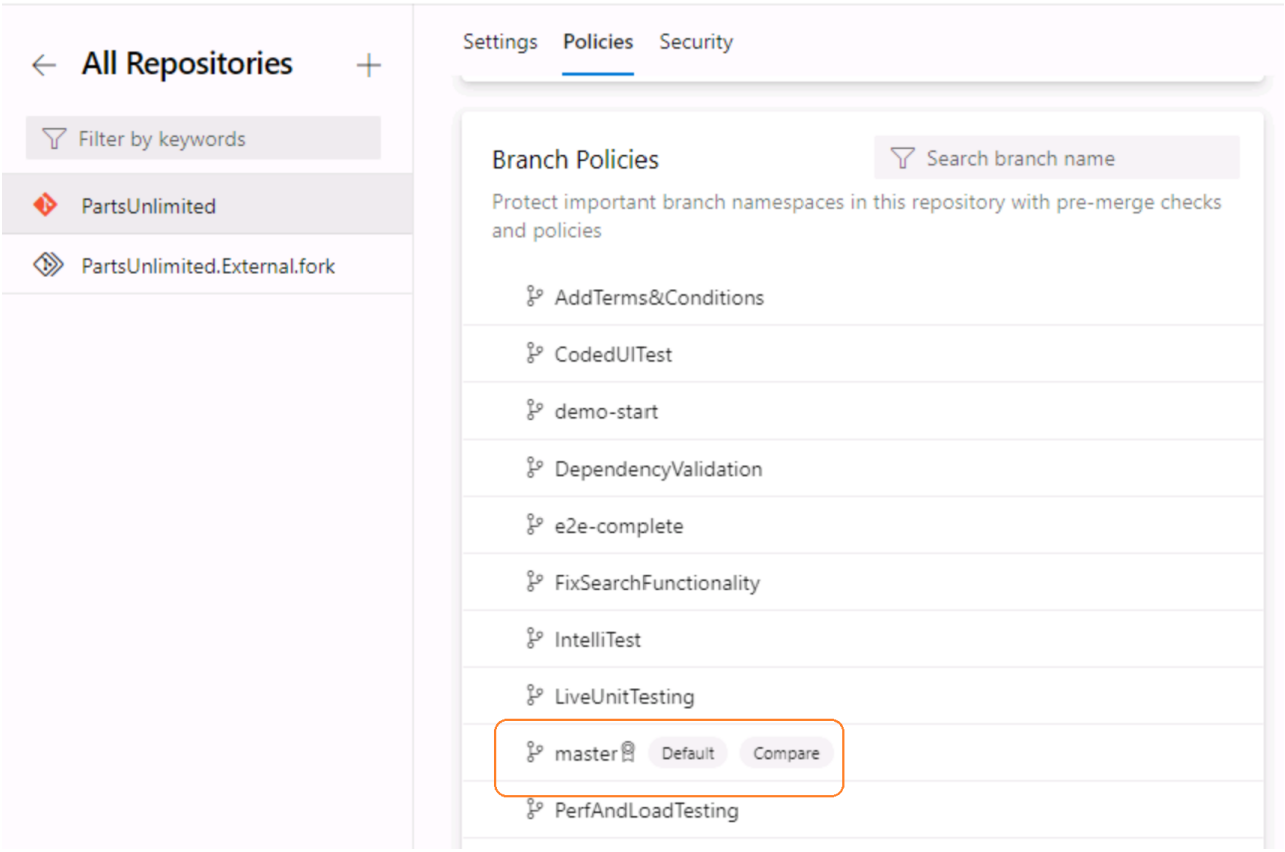
---

Task 5: Remove Branch Policies

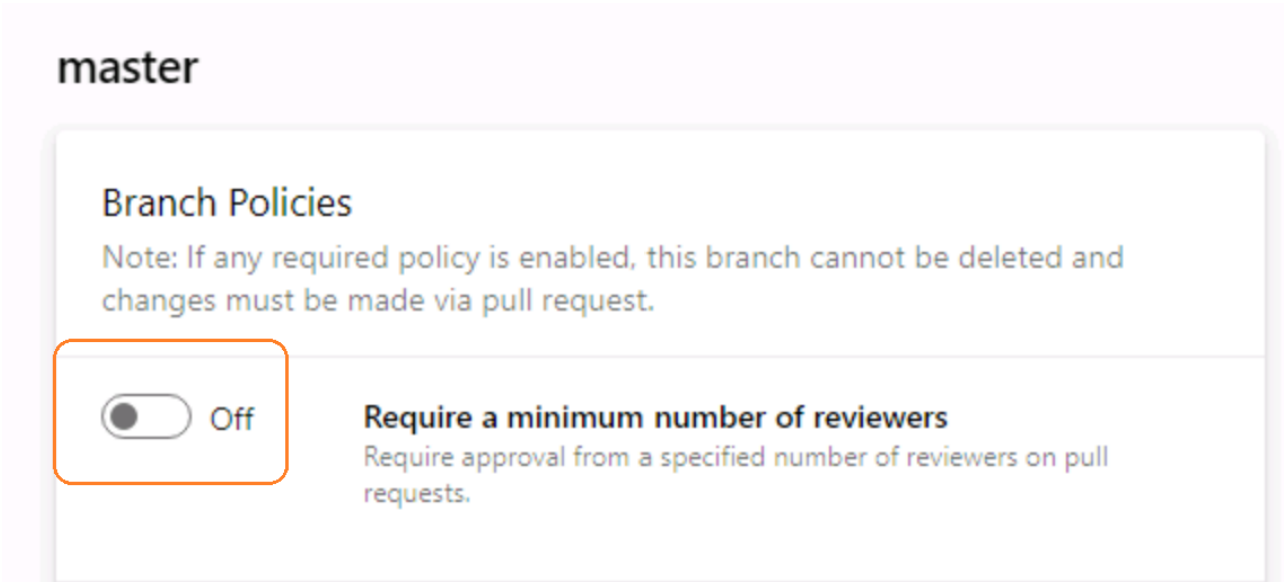
1. In Azure DevOps Services, open **Project settings** from the bottom-left corner.
2. Select **Repositories** under **Repos**.
3. Select the **PartsUnlimited** repo. Click **Policies**.



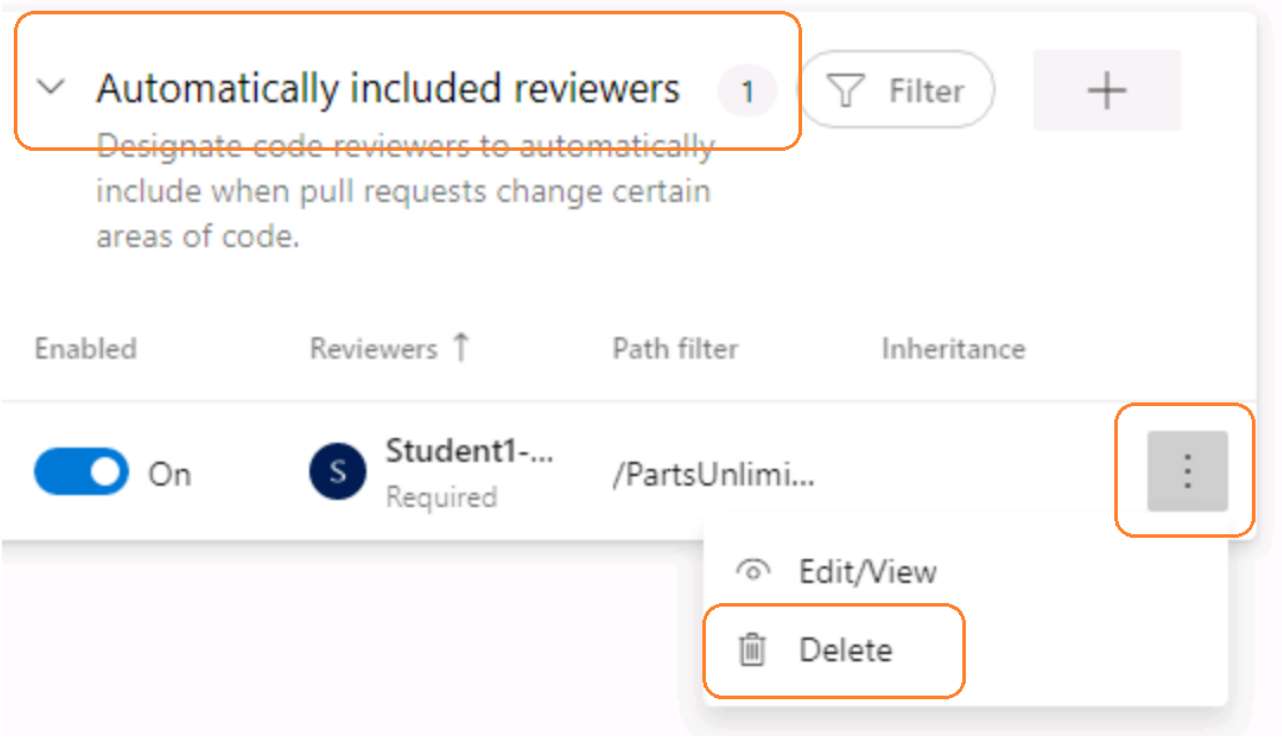
4. Scroll down and select the **master** branch from the Branch Policies section.



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



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



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