Description
In this video, you will learn about source control principles and source control systems. You will also learn about Azure repositories, migrating strategies and authentication options.

(A)What is Source Control?
Source control (or version control) is the practice of tracking and managing changes to code. Source control management (SCM) systems provide a running history of code development and help to resolve conflicts when merging contributions from multiple sources. For most software teams, the source code is a repository of the invaluable knowledge and understanding about the problem domain that the developers have collected and refined through careful effort. Version control protects source code from both catastrophe and casual degradation of human error and unintended consequences.

Without version control, you're tempted to keep multiple copies of code on your computer. This is dangerous-it's easy to change or delete a file in the wrong copy of code, potentially losing work. Version control systems solve this problem by managing all versions of your code but presenting you with a single version at a time.

(B)Benefits of Source Control?
" Code doesn't exist unless it's committed into source control … Source control is the fundamental enabler of continuous delivery."

Whether you are writing code professionally or personally, you should always version your code using a source control management system. Some of the advantages of using source control are,
1. **Create workflows**
Version control workflows prevent the chaos of everyone using their own development process with different and incompatible tools. Version controls systems provide process enforcement and permissions, so everyone stays on the same page.

2. **Work with versions**
Every version has a description in the form of a comment. These descriptions helps you follow changes in your code by version instead of by individual file changes. Code stored in versions can be viewed and restored from version control at any time as needed. This makes it easy to base new work off any version of code.

3. **Collaboration**
Version control synchronizes versions and makes sure that changes doesn't conflict with other changes from your team. Your team relies on version control to help resolve and prevent conflicts, even when people make changes at the same time.

4. **Maintains history of changes**
Version control keeps a history of changes as your team saves new versions of your code. This history can be reviewed to find out who, why, and when changes were made. History give you the confidence to experiment since you can roll back to previous good version at any time. History lets you base work from any version of code, such as to fix a bug in a previous release.

5. **Automate tasks**
Version control automation features save your team time and generate consistent results. Automate testing, code analysis and deployment when new versions are saved to version control.

(C)Type of Source Control Systems

(a) **Git (distributed)**

Git is a distributed version control system. Each developer has a copy of the source repository on their dev machine. Developers can commit each set of changes on their dev machine and perform version control operations such as history and compare without a network connection. Branches are lightweight. When you need to switch contexts, you can create a private local branch. Yu can quickly switch from one branch to another to pivot among different variations of your codebase. Later, you can merge, publish, or dispose of the branch.

(b) **TFVC (centralized)**

Team Foundation Version Control (TFVC) is a centralized version control system. Typically, team members have only one version of each file on their dev machines. Historical data is maintained only on the server. Branches are path-based and created on the server.

TFVC has two workflow models:

(i) **Server workspaces** – Before making changes, team members publicly check out files. Most operations require developers to be connected to the server. This system facilitates locking workflows. Other systems that work this way include Visual Source Safe, Perforce, and CVS. With server workspaces, you can scale up to very large codebases with millions of files per branch and large binary files.

(ii) **Local workspaces** – Each team member takes a copy of the latest version of the codebase with them and works offline as needed. Developers check in their changes and resolve conflicts, as necessary. Another system that works this way is Subversion.

(D) Migrate from TFVC to Git

Most teams wish they could reorganize their source control structure – typically the structure the team is using today was set up by a well-meaning developer a decade ago but it's not really optimal. Migrating to Git could be a good opportunity to restructure your repo. In this case, it probably doesn't make sense to migrate history anyway, since you're going to restructure the code (or break the code into multiple repos). The process is simple: create an empty Git repo (or multiple empty repos), then get-latest from TFS and copy/reorganize the code into the empty Git repos. Then just commit and push and you're there! Of course if you have shared code you need to create builds of the shared code to publish to a package feed and then consume those packages in downstream applications, but the Git part is really simple.

(E)Authenticate the Git Credential Manager

Git Credential Managers simplify authentication with your Azure DevOps Services/TFS Git repos. Credential Managers let you use the same credentials that you use for the Azure DevOps Services/TFS web portal and support multi-factor authentication through Microsoft Account (MSA) or Azure Active Directory (Azure AD). In addition to supporting multi-factor authentication with Azure DevOps Service, the credential managers also provide support two-factor authentication with Github repositories.

Azure DevOps Services provides IDE support for MSA and Azure AD authentication through Team Explorer in Visual Studio, IntelliJ and Android Studio with the Azure Repos Plugin for IntelliJ, and Eclipse (with the Team Explorer Everywhere plug-in). If your environment doesn't have integration available, configure your IDE with a Personal Access Token or SSH to connect with your repos.

Demo: Version Controlling with Git

Version Controlling with Git in Azure Repos - https://www.azuredevopslabs.com/azuredevops/git/

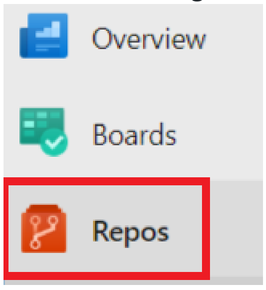
Prerequisites

- Visual Studio Code with the C# extension installed.
- This lab requires you to complete task 1 from the prerequisite instructions.

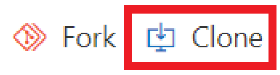
Exercise 1: Cloning an existing repository

Task 1: Cloning an existing repository

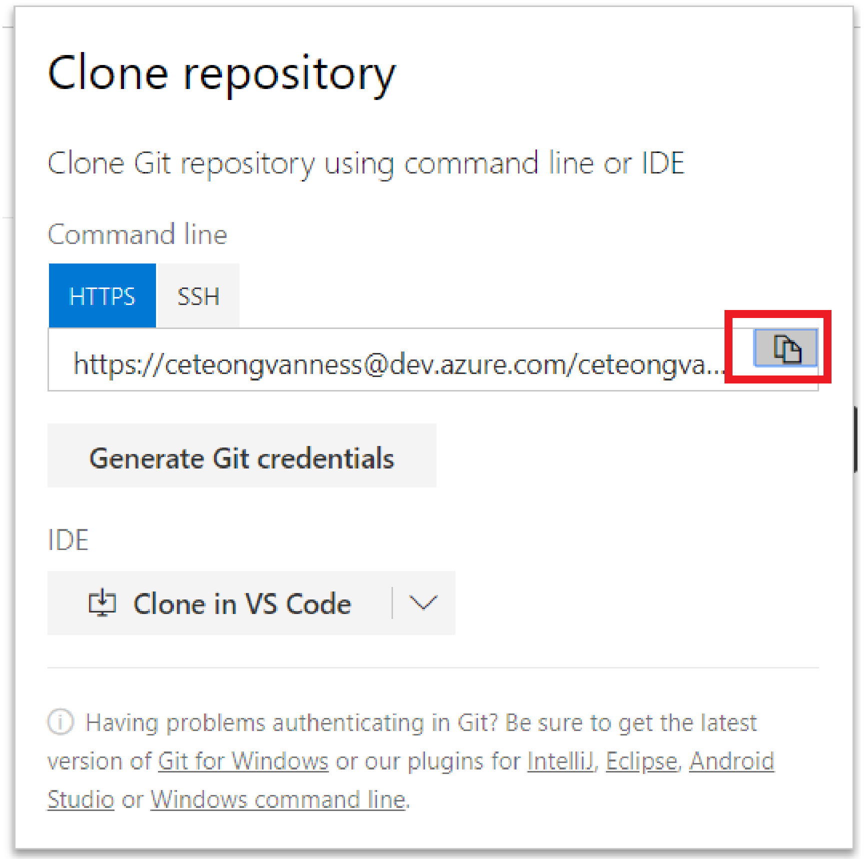
1. In a browser tab, navigate to your team project on Azure DevOps.
2. Getting a local copy of a Git repo is called "cloning". Every mainstream development tool supports this and will be able to connect to Azure Repos to pull down the latest source to work with. Navigate to the **Repos** hub.



3. Click **Clone in Visual Studio Code**.



4. Click the **Copy 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.



Clone repository

Clone Git repository using command line or IDE

Command line

HTTPS SSH

https://ceteongvanness@dev.azure.com/ceteongva...

Generate Git credentials

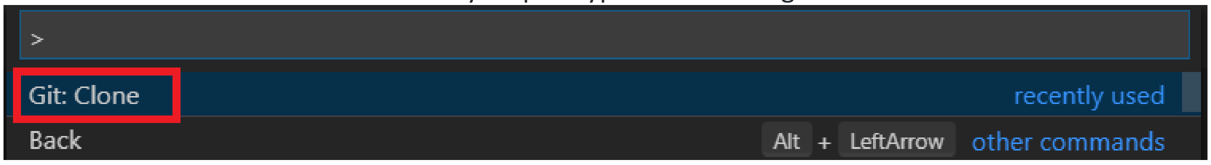
IDE

Clone in VS Code

Having problems authenticating in Git? Be sure to get the latest version of Git for Windows or our plugins for IntelliJ, Eclipse, Android Studio or Windows command line.

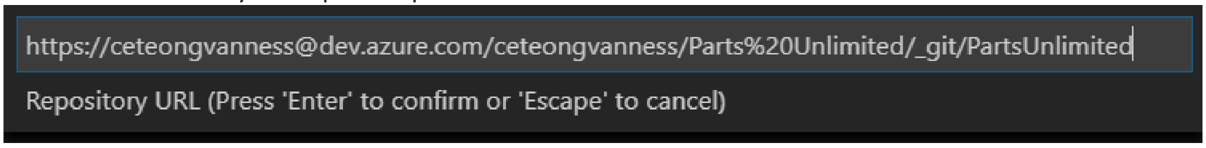
5. Open an instance of **Visual Studio Code**.

6. Press **Ctrl+Shift+P** to show the **Command Palette**. The Command Palette provides an easy and convenient way to access a wide variety of tasks, including those provided by 3rd extensions.
7. Execute the **Git:Clone** command. It may help to type "Git" to bring it to the shortlist.



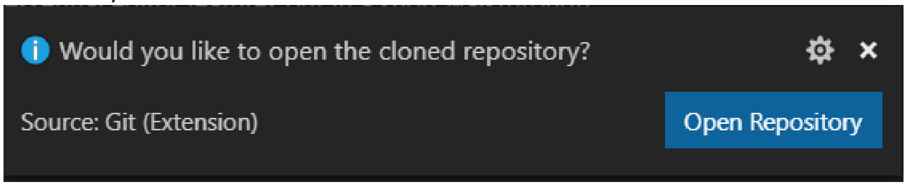
```
>
Git: Clone                                              recently used
Back                                 Alt + LeftArrow    other commands
```

8. Paste in the URL to your repo and press **Enter**.



```
https://ceteongvanness@dev.azure.com/ceteongvanness/Parts%20Unlimited/_git/PartsUnlimited
Repository URL (Press 'Enter' to confirm or 'Escape' to cancel)
```

9. Select a local path to clone the repo to.
10. When prompted, log in to your Azure DevOps account.
11. Once the cloning has completed, click **Open 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.

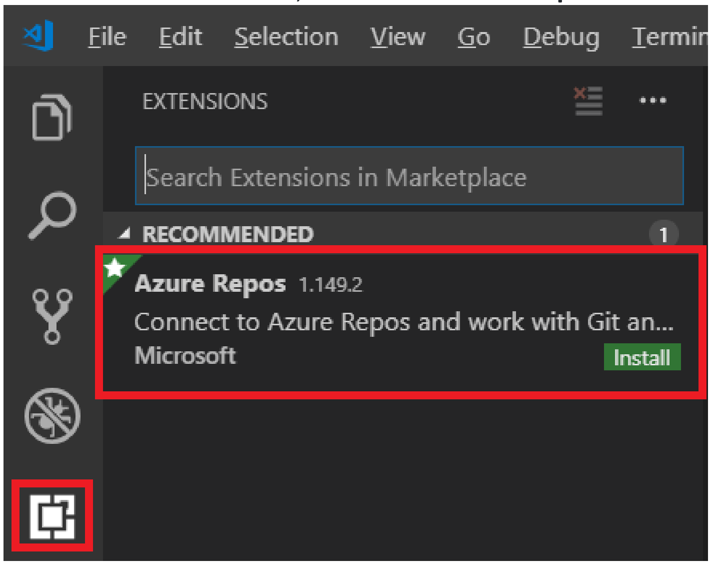


Would you like to open the cloned repository?

Source: Git (Extension)

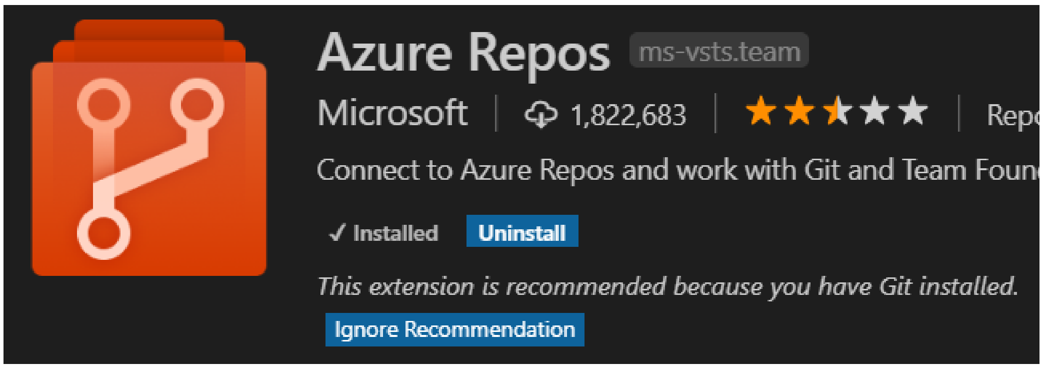
Task 2: Installing the Azure Repos extension for Visual Studio Code

1. The Azure Repos extension provides convenient access to many features of Azure DevOps. From the **Extensions** tab, search for "**Azure Repos**" and click **Install** to install it.



Azure Repos 1.149.2
Connect to Azure Repos and work with Git an...
Microsoft

2. Click **Reload** once the extension has finished installing. If this option is not available, reopen **Visual Studio Code**.


Azure Repos ms-vsts.team

Microsoft | 1,822,683 | Repo

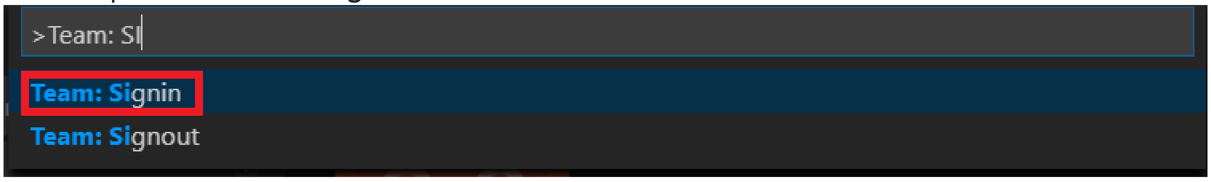
Connect to Azure Repos and work with Git and Team Foun

✓ Installed Uninstall

This extension is recommended because you have Git installed.

Ignore Recommendation

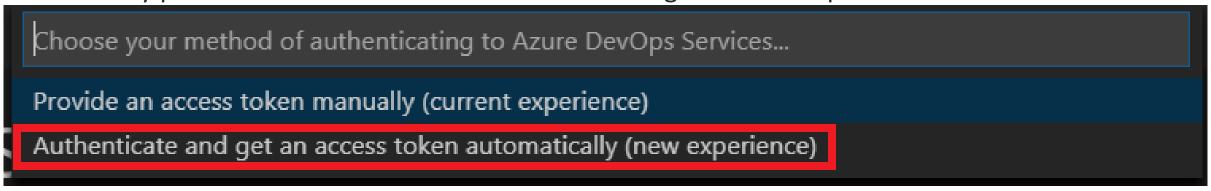
3. Press **Ctrl+Shift+P** to show the **Command Palette**.
4. Search for "**Team**" to see all the new commands that are now available for working with Azure Repos. Select **Team: Signin**.


>Team: SI

Team: Signin

Team: Signout

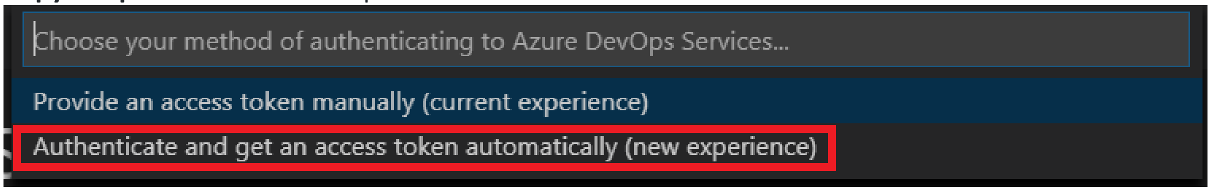
5. Select **Authenticate and get an access token automatically**. Note that you could alternatively provide the token create earlier if following the manual path.


Choose your method of authenticating to Azure DevOps Services...

Provide an access token manually (current experience)

Authenticate and get an access token automatically (new experience)

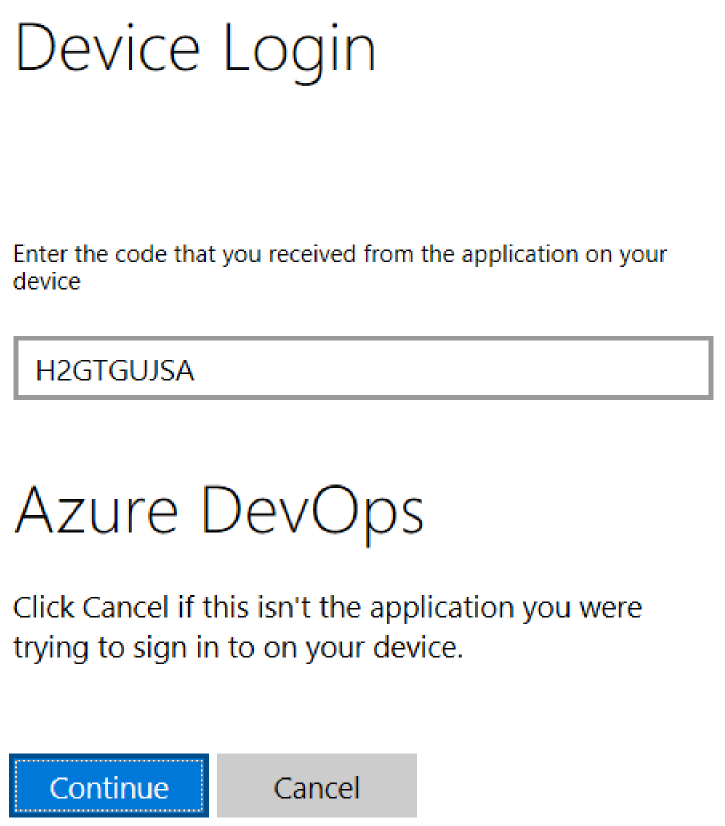
6. **Copy the provided token** and press **Enter** to launch a browser tab.


Choose your method of authenticating to Azure DevOps Services...

Provide an access token manually (current experience)

Authenticate and get an access token automatically (new experience)

7. Paste the code in to the login box and click **Continue**.


Device Login

Enter the code that you received from the application on your device

H2GTGUJSA

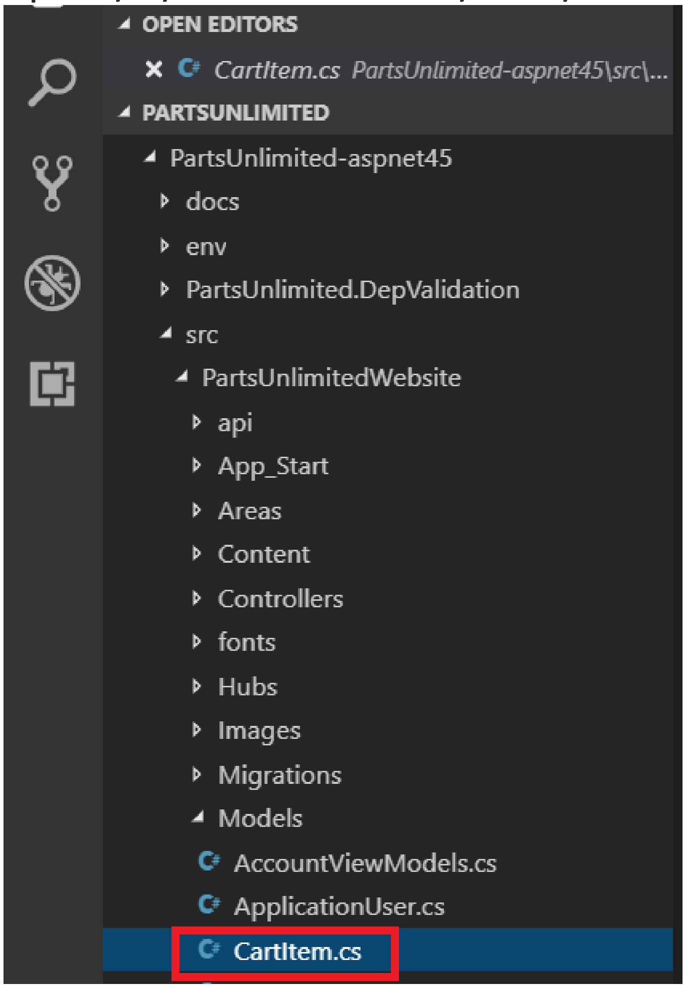
Azure DevOps

Click Cancel if this isn't the application you were trying to sign in to on your device.

8. Select the Microsoft account associated with your Azure DevOps account.
9. When the process has complete, close the browser tab.

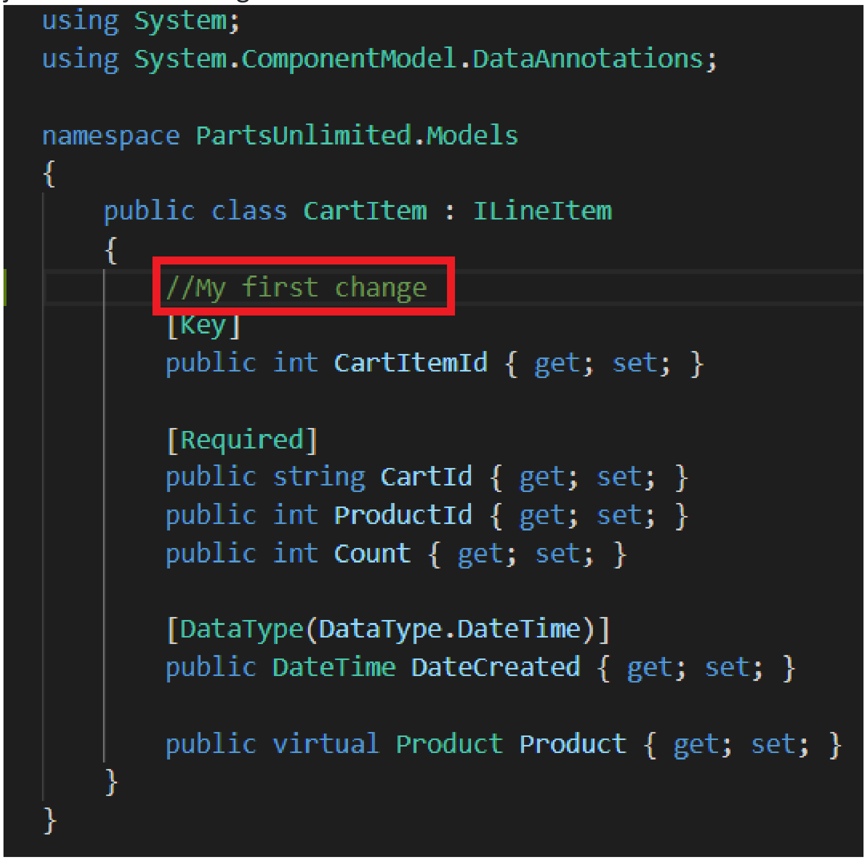
Exercise 2: Saving work with commits
Task 1: Committing changes

1. From the **Explorer** tab, open **/PartsUnlimited-aspnet45/src/PartsUnlimitedWebsite/Models/CartItem.cs**.



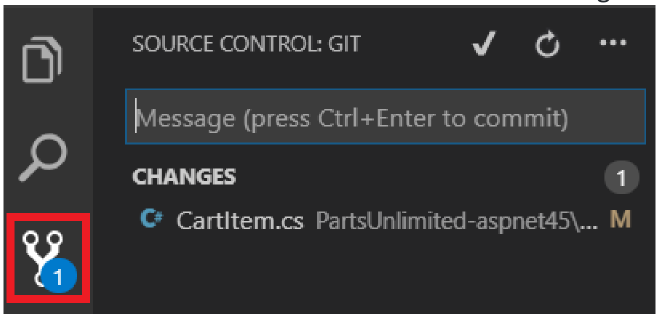
```
OPEN EDITORS
  × CartItem.cs PartsUnlimited-aspnet45\src\...
PARTSUNLIMITED
  PartsUnlimited-aspnet45
    docs
    env
    PartsUnlimited.DepValidation
    src
      PartsUnlimitedWebsite
        api
        App_Start
        Areas
        Content
        Controllers
        fonts
        Hubs
        Images
        Migrations
        Models
          AccountViewModels.cs
          ApplicationUser.cs
          CartItem.cs
```

2. Add a comment to the file. It doesn't really matter what the commend is since the goal is just to make a change. Press **Ctrl + S** to save the file.



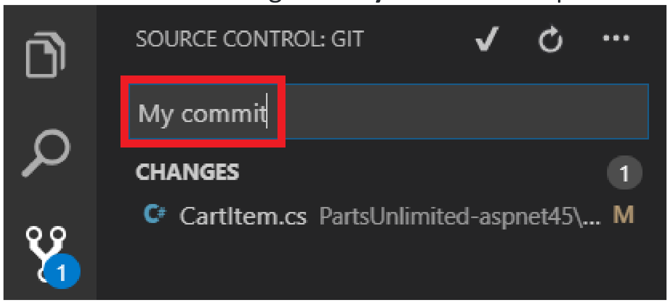
```
using System;
using System.ComponentModel.DataAnnotations;

namespace PartsUnlimited.Models
{
    public class CartItem : ILineItem
    {
        //My first change
        [Key]
        public int CartItemId { get; set; }

        [Required]
        public string CartId { get; set; }
        public int ProductId { get; set; }
        public int Count { get; set; }

        [DataType(DataType.DateTime)]
        public DateTime DateCreated { get; set; }

        public virtual Product Product { get; set; }
    }
}
```

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

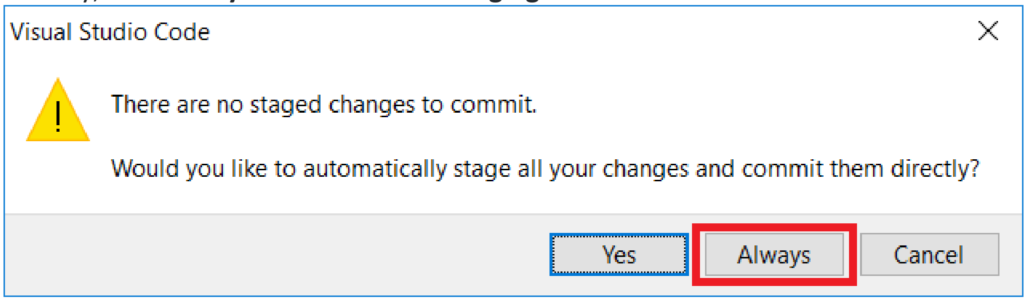


SOURCE CONTROL: GIT
Message (press Ctrl+Enter to commit)
CHANGES 1
CartItem.cs PartsUnlimited-aspnet45\... M
1

4. Enter a commit message of "**My commit**" and press **Ctrl + Enter** to commit it locally.



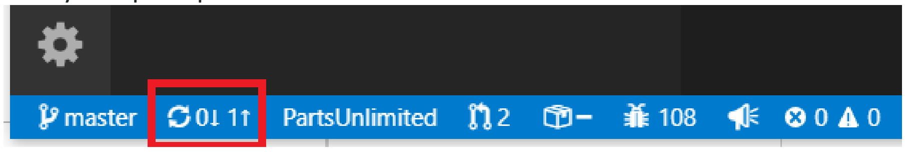
5. If asked whether you would like to automatically stage your changes and commit them directly, click **Always**. We will discuss **staging** later in the lab.



There are no staged changes to commit.

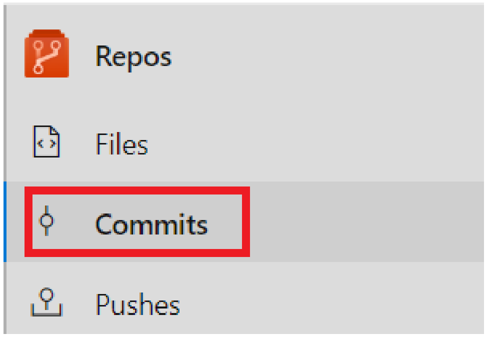
Would you like to automatically stage all your changes and commit them directly?

6. Click the **Synchronize Changes** button to synchronize your changes with the server. Confirm the sync if prompted.

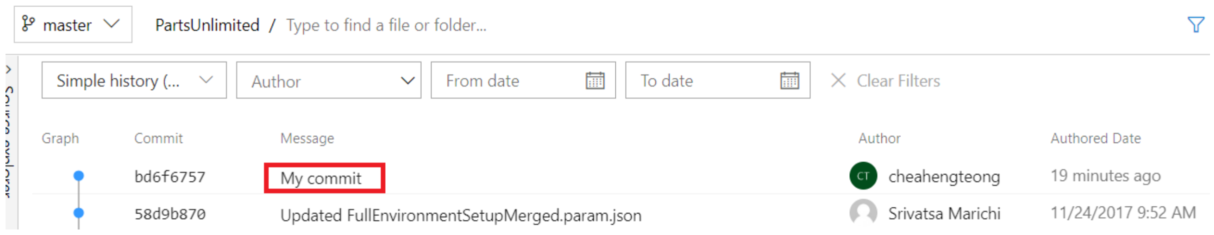


Task 2: Reviewing commits

1. Switch to the Azure DevOps browser tab. You can review the latest commits on Azure DevOps under the **Commits** tab of the **Repos** hub.



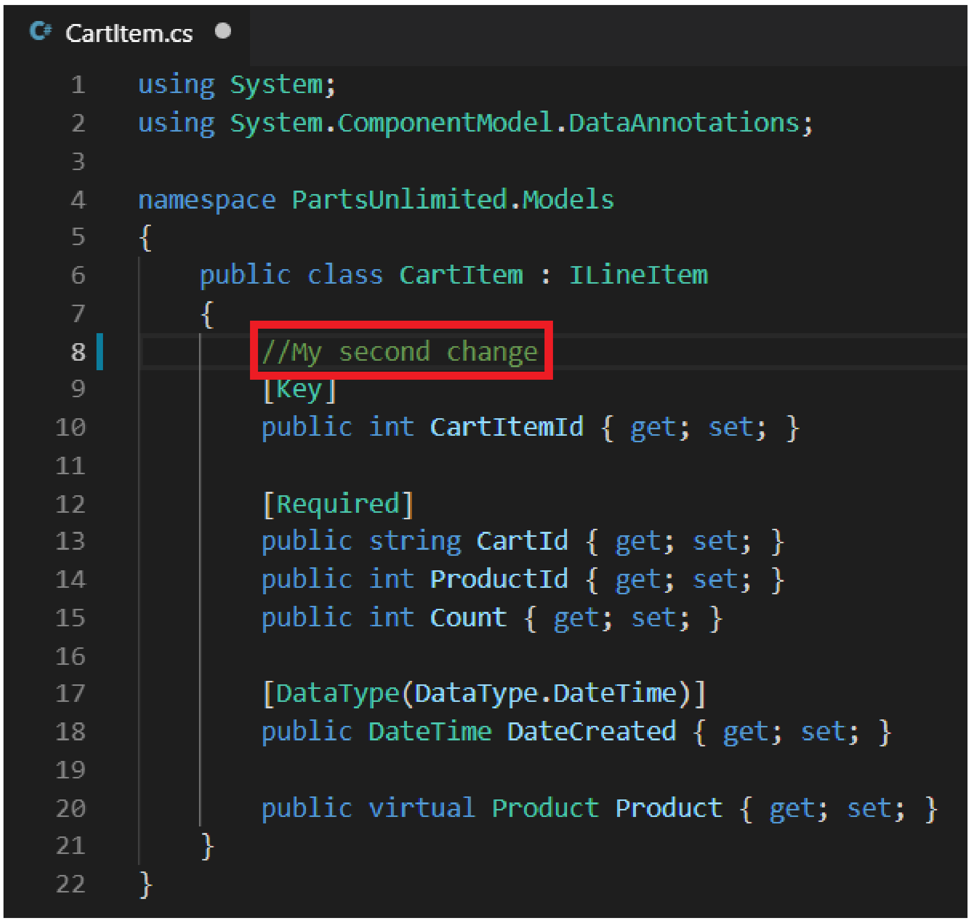
2. The recent commit should be right at the top.



Graph	Commit	Message	Author	Authored Date
	bd6f6757	My commit	cheahengteong	19 minutes ago
	58d9b870	Updated FullEnvironmentSetupMerged.param.json	Srivatsa Marichi	11/24/2017 9:52 AM

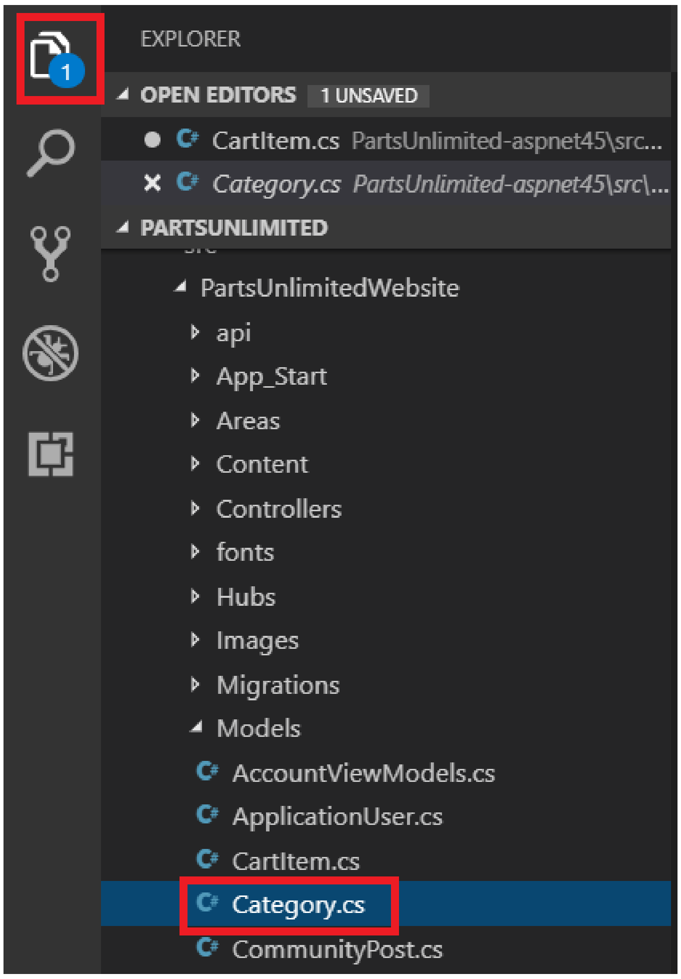
Task 3: Staging changes

1. Return to **Visual Studio Code**.
2. Update the open **CartItem.cs** class by editing the comment you made earlier and **saving the file**.

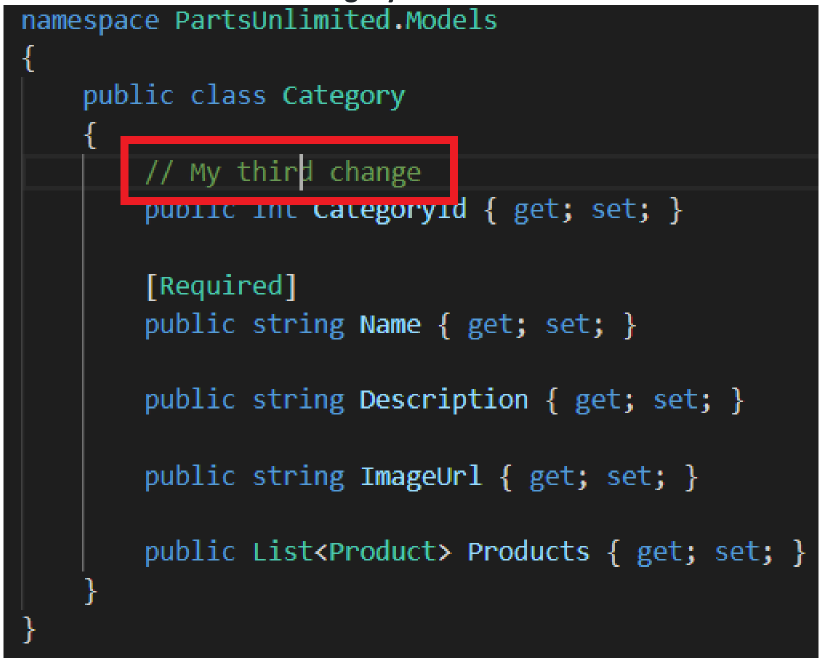


```
using System;
using System.ComponentModel.DataAnnotations;

namespace PartsUnlimited.Models
{
    public class CartItem : ILineItem
    {
        //My second change
        [Key]
        public int CartItemId { get; set; }

        [Required]
        public string CartId { get; set; }
        public int ProductId { get; set; }
        public int Count { get; set; }

        [DataType(DataType.DateTime)]
        public DateTime DateCreated { get; set; }

        public virtual Product Product { get; set; }
    }
}
```

3. Open **Category.cs** as well.

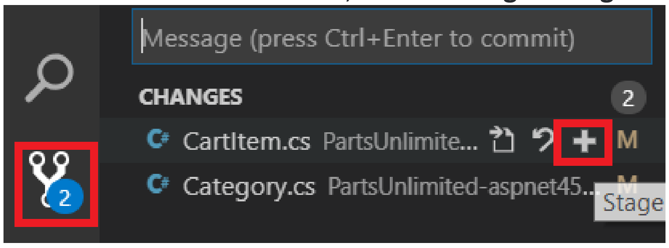


4. Add a new comment to **Category.cs** so there will be two files with changes. **Save the file**.

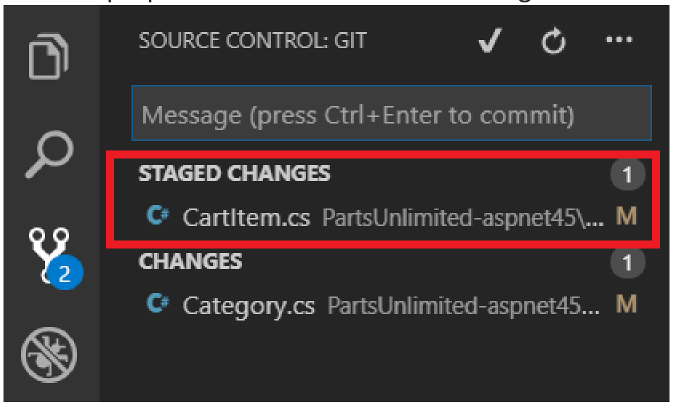


```
namespace PartsUnlimited.Models
{
    public class Category
    {
        // My third change
        public int CategoryId { get; set; }

        [Required]
        public string Name { get; set; }

        public string Description { get; set; }

        public string ImageUrl { get; set; }

        public List<Product> Products { get; set; }
    }
}
```

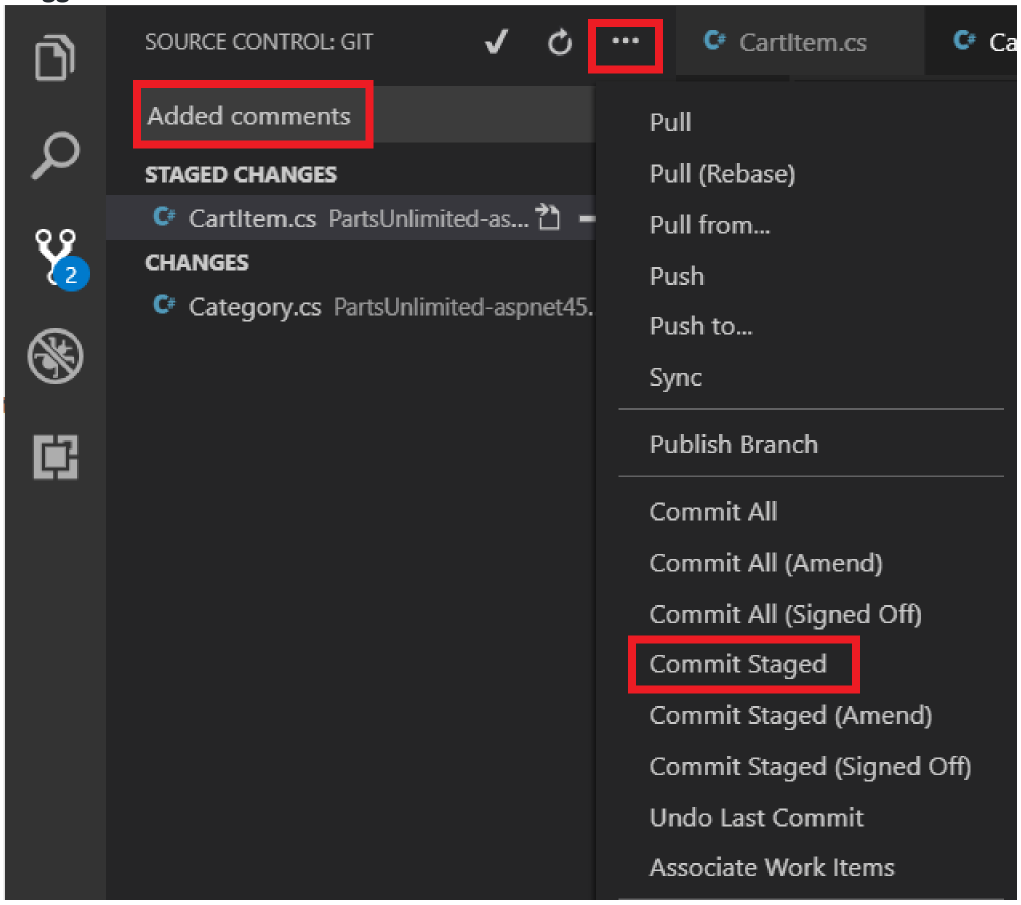
5. From the **Source Control** tab, click the **Stage Changes** button for **CartItem.cs**.


Message (press Ctrl+Enter to commit)
CHANGES 2
CartItem.cs PartsUnlimite... M
Category.cs PartsUnlimited-aspnet45... M
Stage

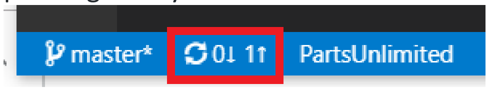
6. This will prepare **CartItem.cs** for committing without **Category.cs**.


SOURCE CONTROL: GIT
Message (press Ctrl+Enter to commit)
STAGED CHANGES 1
CartItem.cs PartsUnlimited-aspnet45\... M
CHANGES 1
Category.cs PartsUnlimited-aspnet45... M

7. Enter a comment of "**Added comments**". From the **More Actions** dropdown, select **Commit Stagged**.

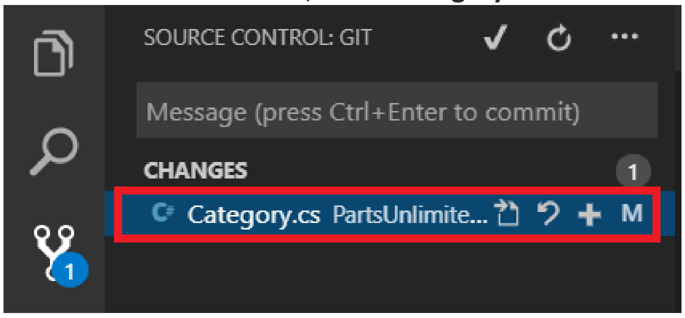


8. Click the **Synchronize Changes** button to synchronize the committed changes with the server. Note that since only the staged changes were committed, the other changes are still pending locally.

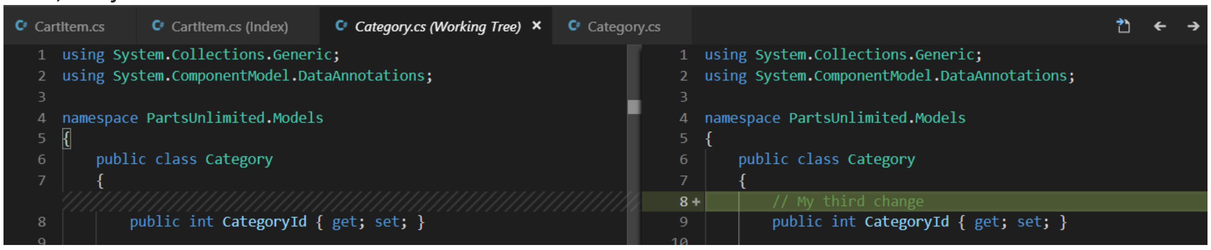


Exercise 3: Reviewing history
Task 1: Comparing files

1. In the **Source Control** tab, select **Category.cs**.


SOURCE CONTROL: GIT
Message (press Ctrl+Enter to commit)
CHANGES
1
Category.cs PartsUnlimite...
M

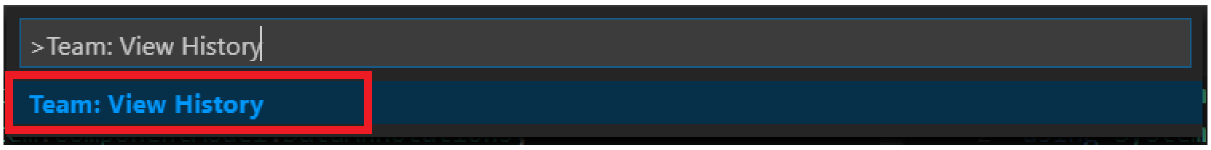
2. A comparison view is opened to enable you to easily locate the changes you've made. In this case, it's just the one comment.



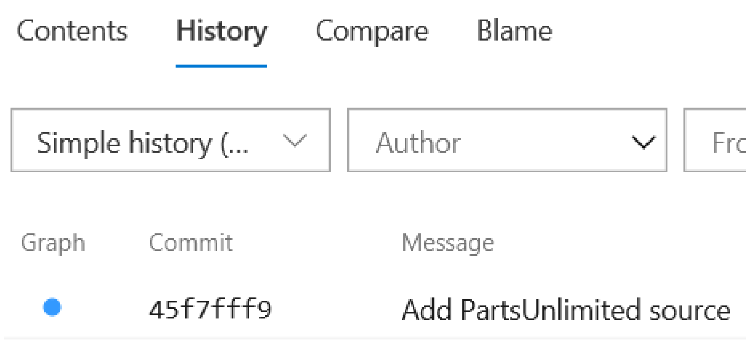
```
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;

namespace PartsUnlimited.Models
{
    public class Category
    {
        public int CategoryId { get; set; }
```

```
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;

namespace PartsUnlimited.Models
{
    public class Category
    {
        // My third change
        public int CategoryId { get; set; }
```

3. Press **Ctrl+Shift+P** to open the **Command Palette**.
4. Start typing "**Team**" and select **Team: View History** when it becomes available. This is a feature of the Azure Repos extension that makes it easy to jump right to the history of this file in a new browser tab.

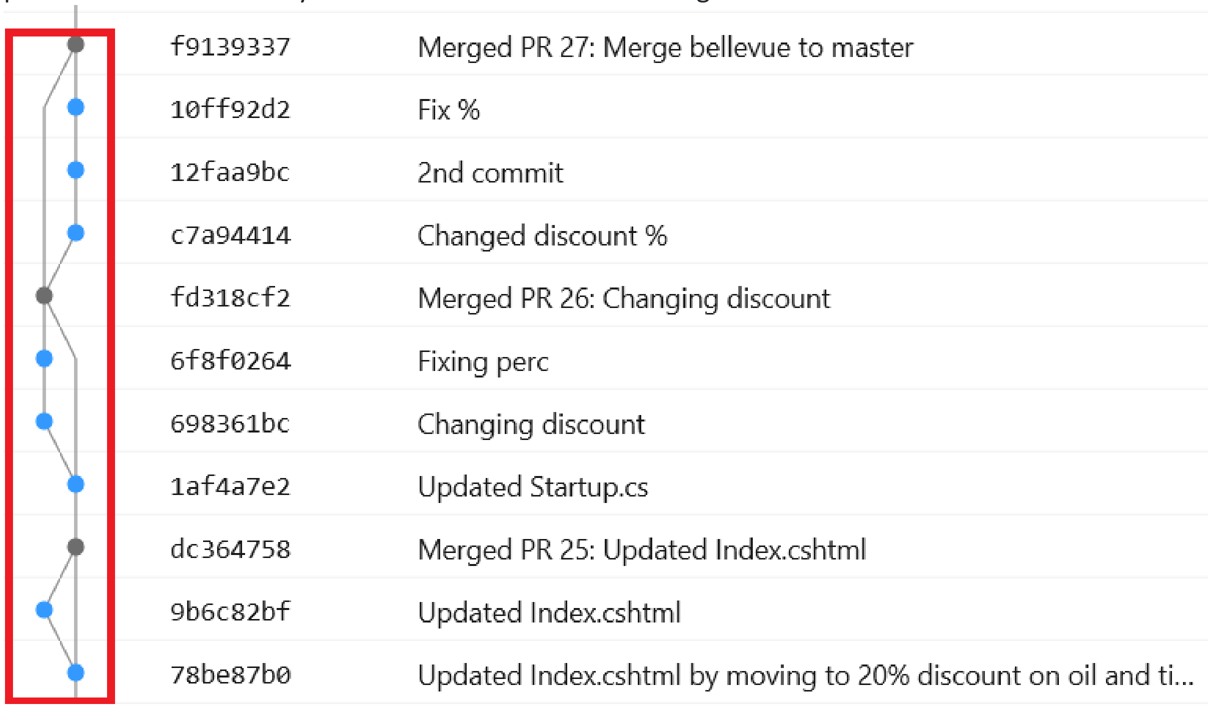


5. Note the history of **Category.cs**. Close the newly created tab, which will should return you to the **Commits** tab from earlier.



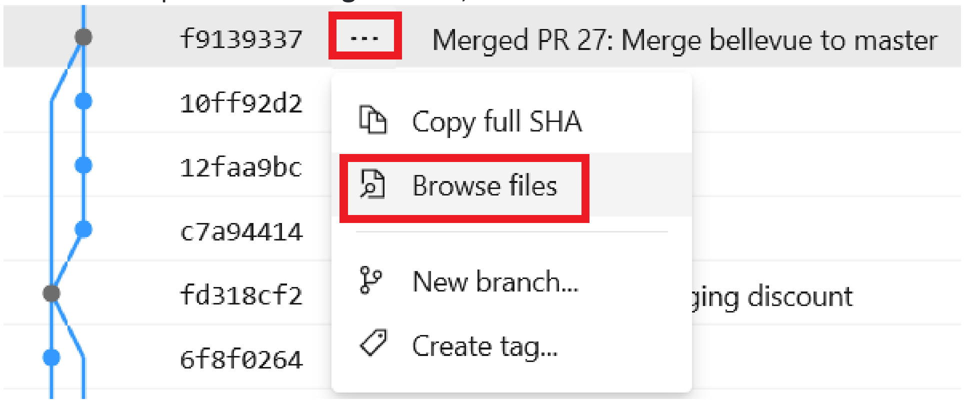
Graph	Commit	Message
●	45f7fff9	Add PartsUnlimited source

6. Scroll down in the **Commits** view to locate some of the source branches and merges. These provide a convenient way to visualize when and how changes were made to the source.



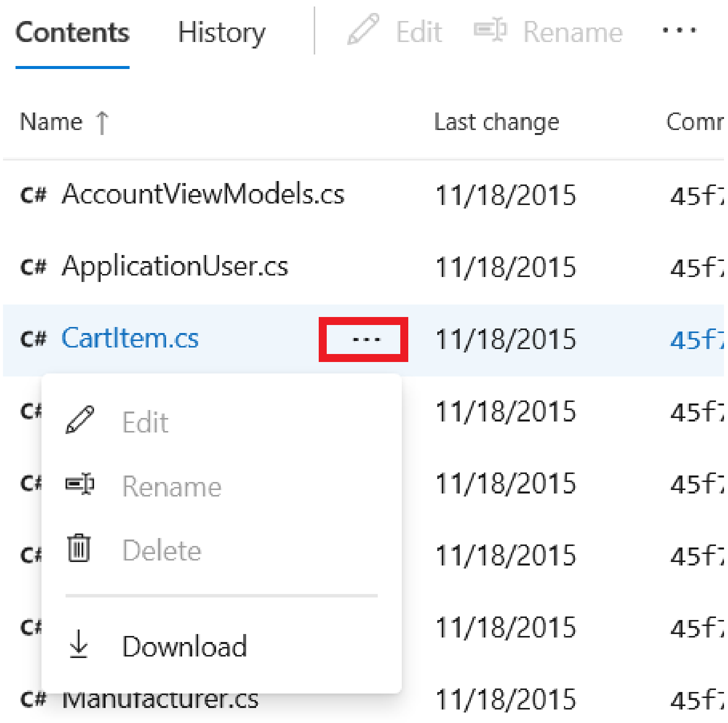
	f9139337	Merged PR 27: Merge bellevue to master
	10ff92d2	Fix %
	12faa9bc	2nd commit
	c7a94414	Changed discount %
	fd318cf2	Merged PR 26: Changing discount
	6f8f0264	Fixing perc
	698361bc	Changing discount
	1af4a7e2	Updated Startup.cs
	dc364758	Merged PR 25: Updated Index.cshtml
	9b6c82bf	Updated Index.cshtml
	78be87b0	Updated Index.cshtml by moving to 20% discount on oil and ti...

7. From the dropdown for **Merged PR 27**, select **Broowse Files**.



f9139337 ··· Merged PR 27: Merge bellevue to master
10ff92d2
12faa9bc
c7a94414
fd318cf2 ging discount
6f8f0264

8. This view offers the ability to navigate around the state of the source at that commit so you can review and download those files.

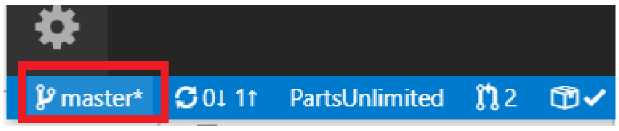


Name ↑	Last change	Comr
AccountViewModels.cs	11/18/2015	45f7
ApplicationUser.cs	11/18/2015	45f7
CartItem.cs	11/18/2015	45f7
	11/18/2015	45f7
	11/18/2015	45f7
	11/18/2015	45f7
	11/18/2015	45f7
Manufacturer.cs	11/18/2015	45f7

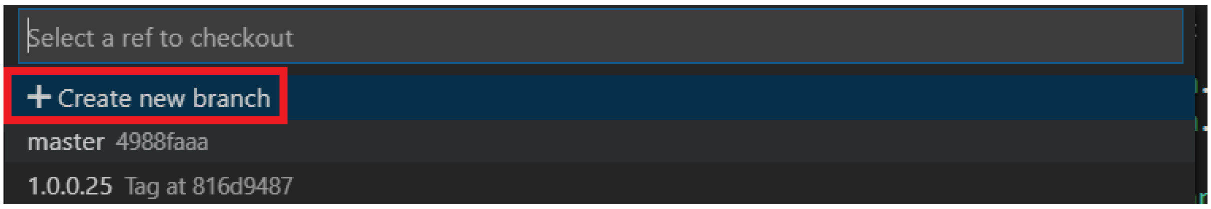
Exercise 4: Working with branches

Task 1: Creating a new branch in your local repository

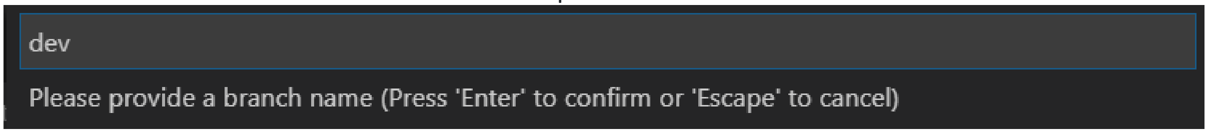
1. Return to **Visual Studio Code**.
2. Click the **master** branch from the bottom left.



3. Select **Create new branch**.

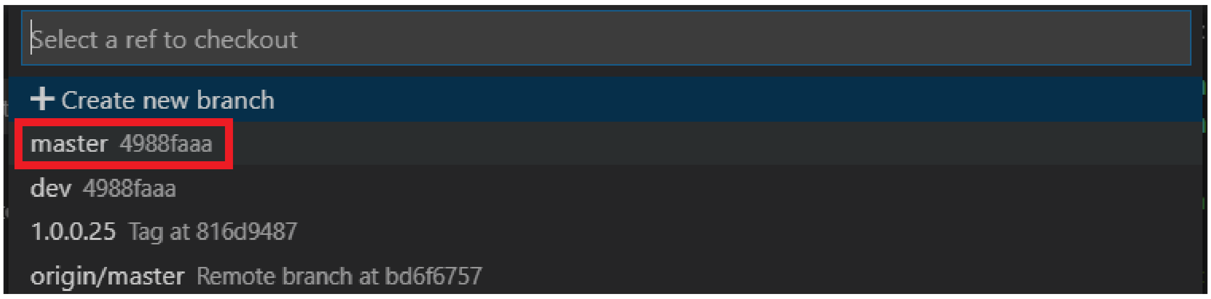


4. Enter the name "**dev**" for the new branch and press **Enter**.

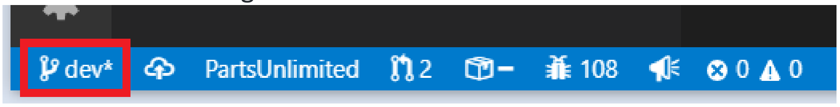


Please provide a branch name (Press 'Enter' to confirm or 'Escape' to cancel)

5. Select the **master** as the reference branch.

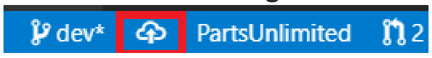


6. You are now working on that branch.

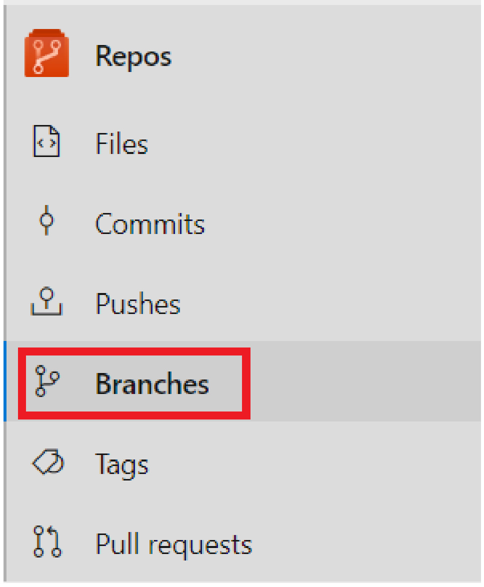


Task 2: Working with branches

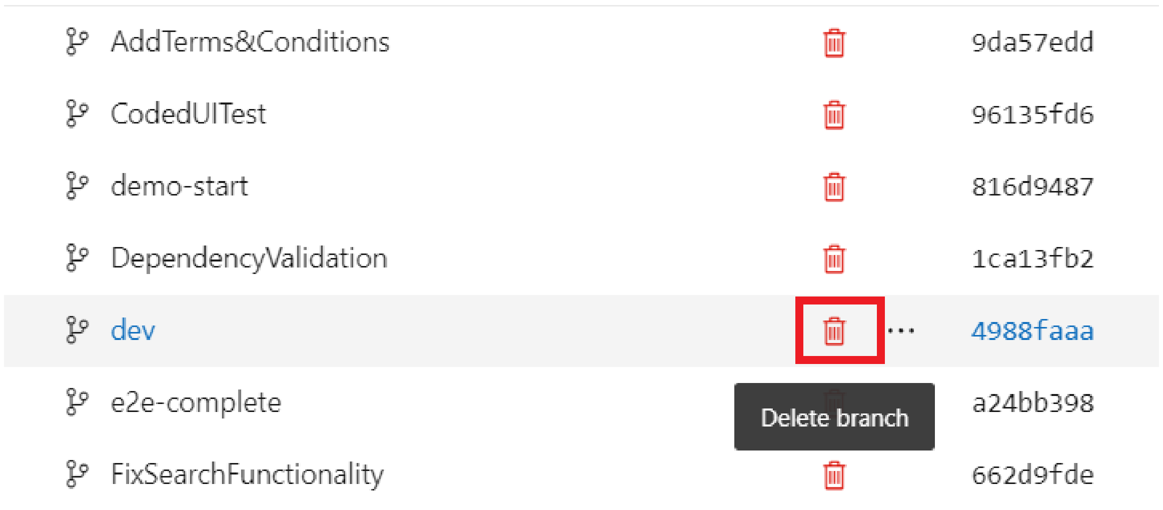
1. Click the **Publish changes** button next to the branch.



2. From the Azure DevOps browser tab, select **Branches**.

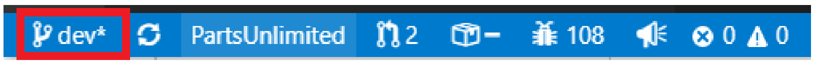


3. You should see the newly pushed **dev** branch. Click the **Delete branch** button to delete it. Confirm the delete.

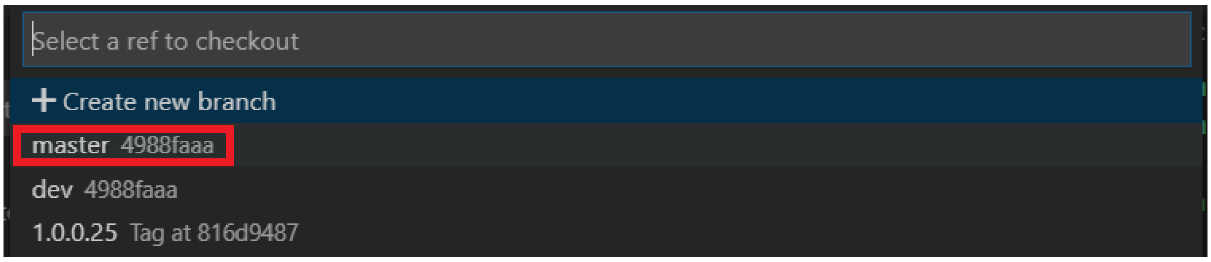


AddTerms&Conditions	9da57edd
CodedUITest	96135fd6
demo-start	816d9487
DependencyValidation	1ca13fb2
dev	4988faaa
e2e-complete	a24bb398
FixSearchFunctionality	662d9fde

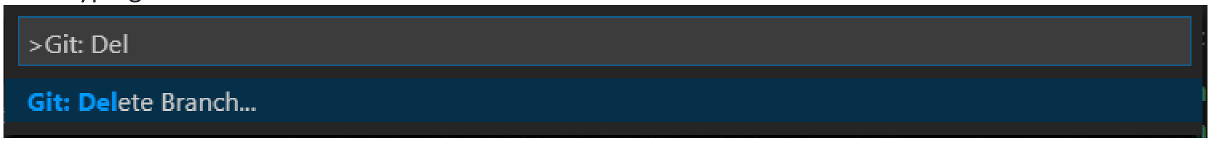
4. Return to **Visual Studio Code**.
5. Click **dev** branch.



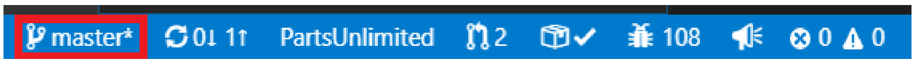
6. Note that there are two **dev** branches listed. The local (dev) branch is there because it's not deleted when the server branch is delete. The server(**origin/dev**) is there because it hasn't been pruned. Select the **master** branch to check it out.



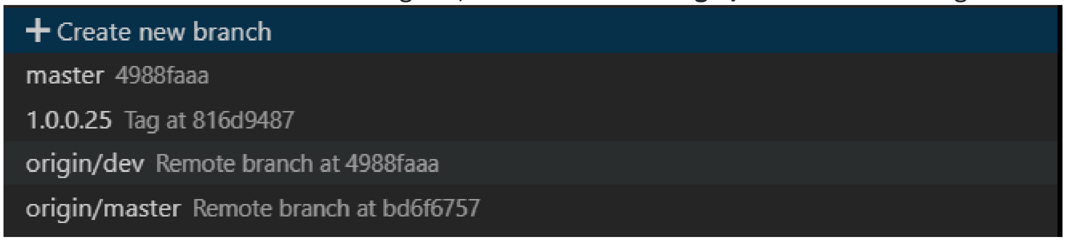
7. Press **Ctrl+Shift+P** to open the **Command Palette**.
8. Start typing "**Git: Delete**" and select **Git: Delete Branch** when it becomes visible.



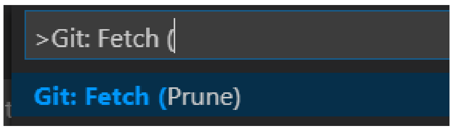
9. There is only one local branch to delete, so select it.
10. Click the **master** branch.



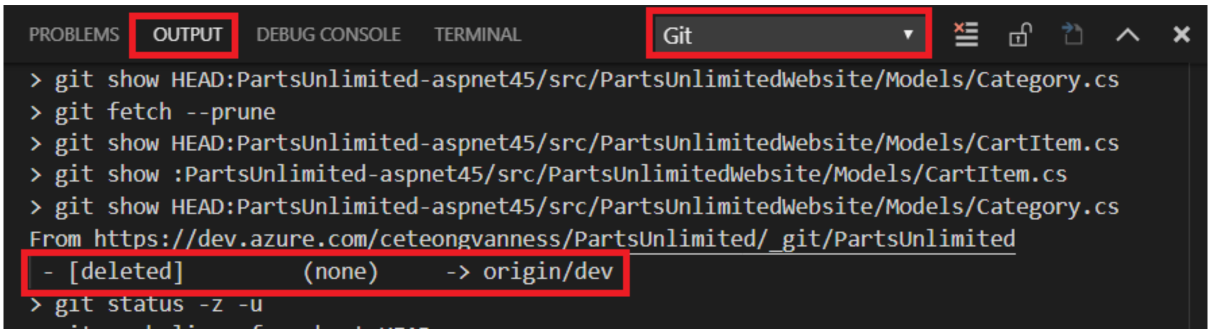
11. Note that the local **dev** branch is gone, but the remote **origin/dev** is still showing.


+ Create new branch
master 4988faaa
1.0.0.25 Tag at 816d9487
origin/dev Remote branch at 4988faaa
origin/master Remote branch at bd6f6757

12. Press **Ctrl+Shift+P** to open the **Command Palette**.
13. Start trying "**Git: Fetch**" and select **Git: Fetch (Prune)** when it become visible. This command will update the origin branches in the local snapshot and delete those that are no longer there.

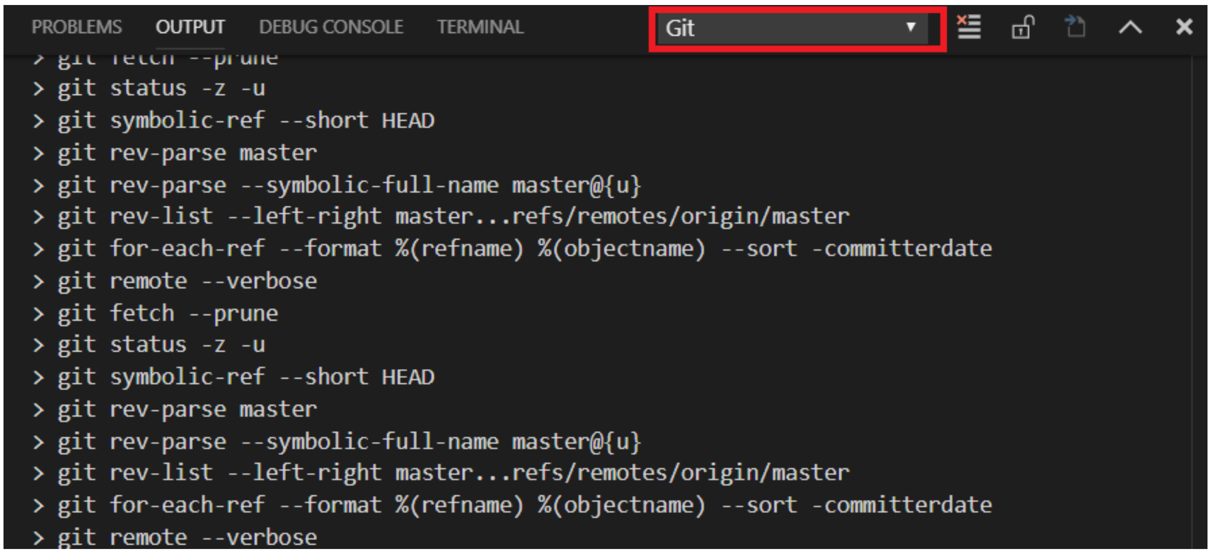


14. You can check in on exactly what these tasks are doing by selecting the **Output** window at the bottom of the screen.



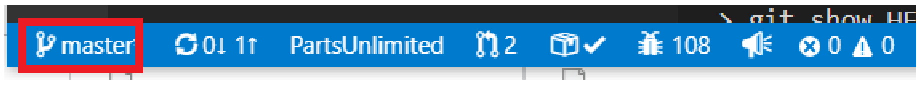
```
> git show HEAD:PartsUnlimited-aspnet45/src/PartsUnlimitedWebsite/Models/Category.cs
> git fetch --prune
> git show HEAD:PartsUnlimited-aspnet45/src/PartsUnlimitedWebsite/Models/CartItem.cs
> git show :PartsUnlimited-aspnet45/src/PartsUnlimitedWebsite/Models/CartItem.cs
> git show HEAD:PartsUnlimited-aspnet45/src/PartsUnlimitedWebsite/Models/Category.cs
From https://dev.azure.com/ceteongvanness/PartsUnlimited/_git/PartsUnlimited
 - [deleted]            (none)      -> origin/dev
> git status -z -u
```

15. Note that if you don't see the Git logs in the output console, you may need to select **Git** as the source.

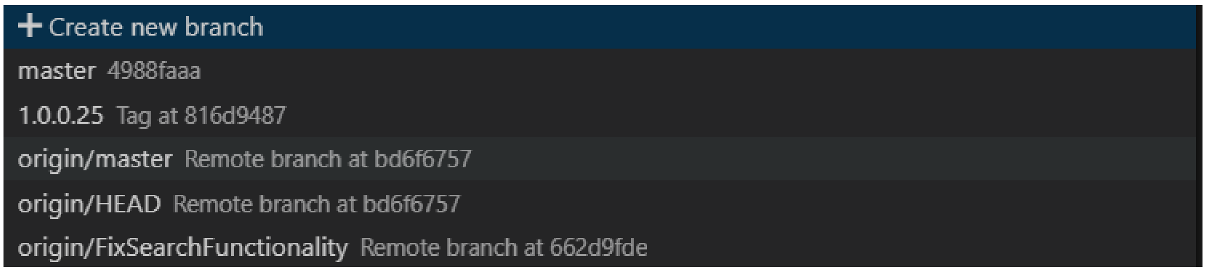


```
> git fetch --prune
> git status -z -u
> git symbolic-ref --short HEAD
> git rev-parse master
> git rev-parse --symbolic-full-name master@{u}
> git rev-list --left-right master...refs/remotes/origin/master
> git for-each-ref --format %(refname) %(objectname) --sort -committerdate
> git remote --verbose
> git fetch --prune
> git status -z -u
> git symbolic-ref --short HEAD
> git rev-parse master
> git rev-parse --symbolic-full-name master@{u}
> git rev-list --left-right master...refs/remotes/origin/master
> git for-each-ref --format %(refname) %(objectname) --sort -committerdate
> git remote --verbose
```

16. Click the **master** branch.



17. The **origin/dev** branch should no longer be in the list.

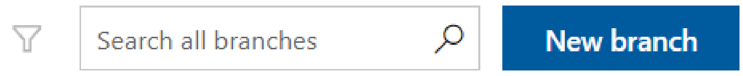


+ Create new branch
master 4988faaa
1.0.0.25 Tag at 816d9487
origin/master Remote branch at bd6f6757
origin/HEAD Remote branch at bd6f6757
origin/FixSearchFunctionality Remote branch at 662d9fde

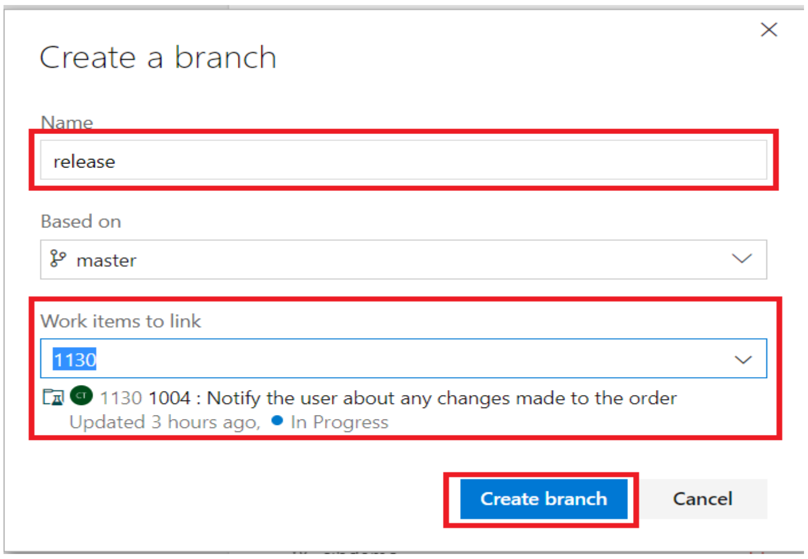
Exercise 5: Managing branches from Azure DevOps

Task 1: Creating a new branch

1. Switch to the Azure DevOps browser tab.
2. Click **New branch**.



3. Enter a name of "**release**" for the new branch. Use the **Work items to link** dropdown to select one or more work items to link to this new branch. Click **Create branch** to create it.


Create a branch

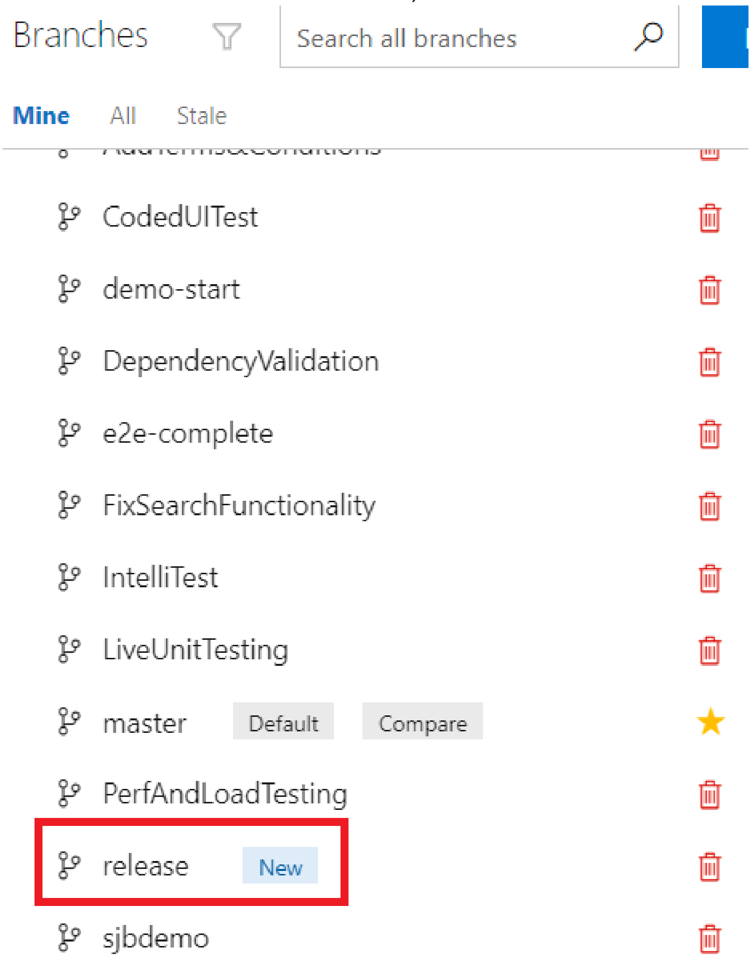
Name
release

Based on
master

Work items to link
1130

1130 1004 : Notify the user about any changes made to the order
Updated 3 hours ago, ● In Progress

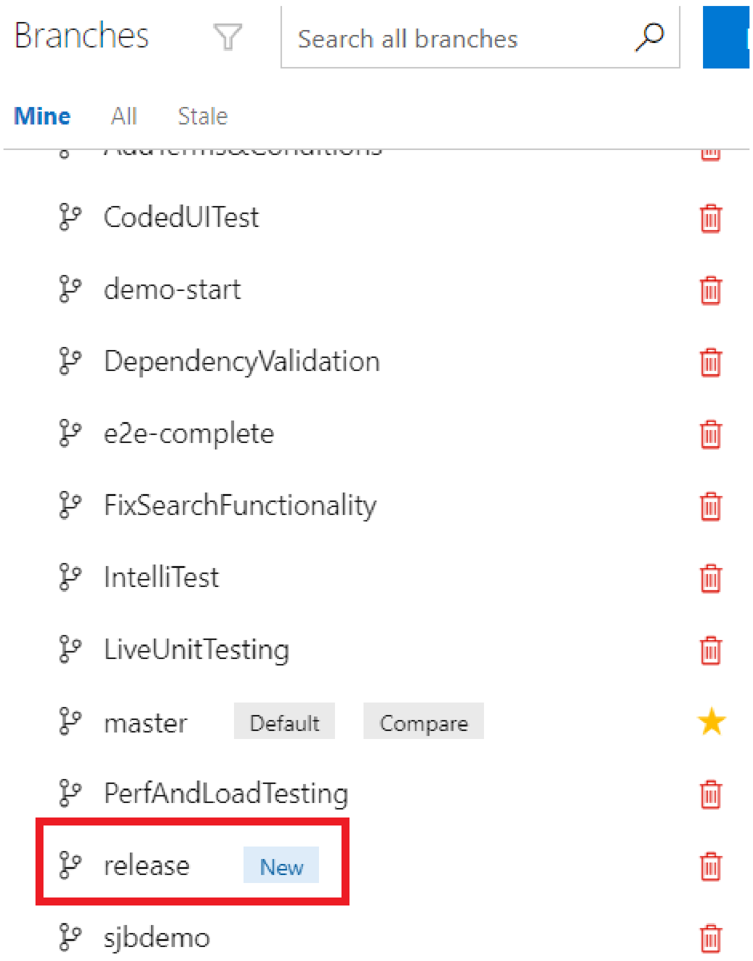
4. After the branch has been created, it will be available in the list.


Branches

Search all branches

CodedUITest
demo-start
DependencyValidation
e2e-complete
FixSearchFunctionality
IntelliTest
LiveUnitTesting
master Default Compare
PerfAndLoadTesting
release New
sjbdemo

5. Return to **Visual Studio Code**.
6. Press **Ctrl+Shift+P** to open the **Command Palette**.
7. Start typing "**Git: Fetch**" and select **Git: Fetch** when it becomes visible. This command will update the origin branches in the local snapshot.


Branches

Search all branches

Mine All Stale

CodedUITest

demo-start

DependencyValidation

e2e-complete

FixSearchFunctionality

IntelliTest

LiveUnitTesting

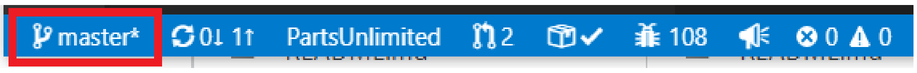
master Default Compare

PerfAndLoadTesting

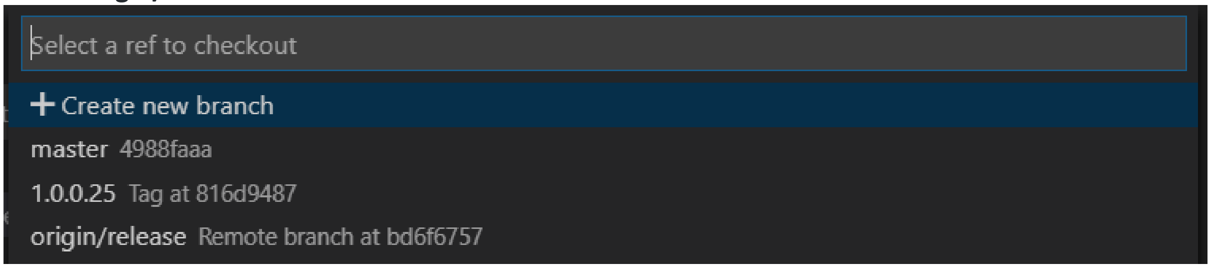
release New

sjbdemo

8. Click the **master** branch.



9. Select **origin/release**. This will create a new local branch called "**release**" and check it out.


Select a ref to checkout

+ Create new branch

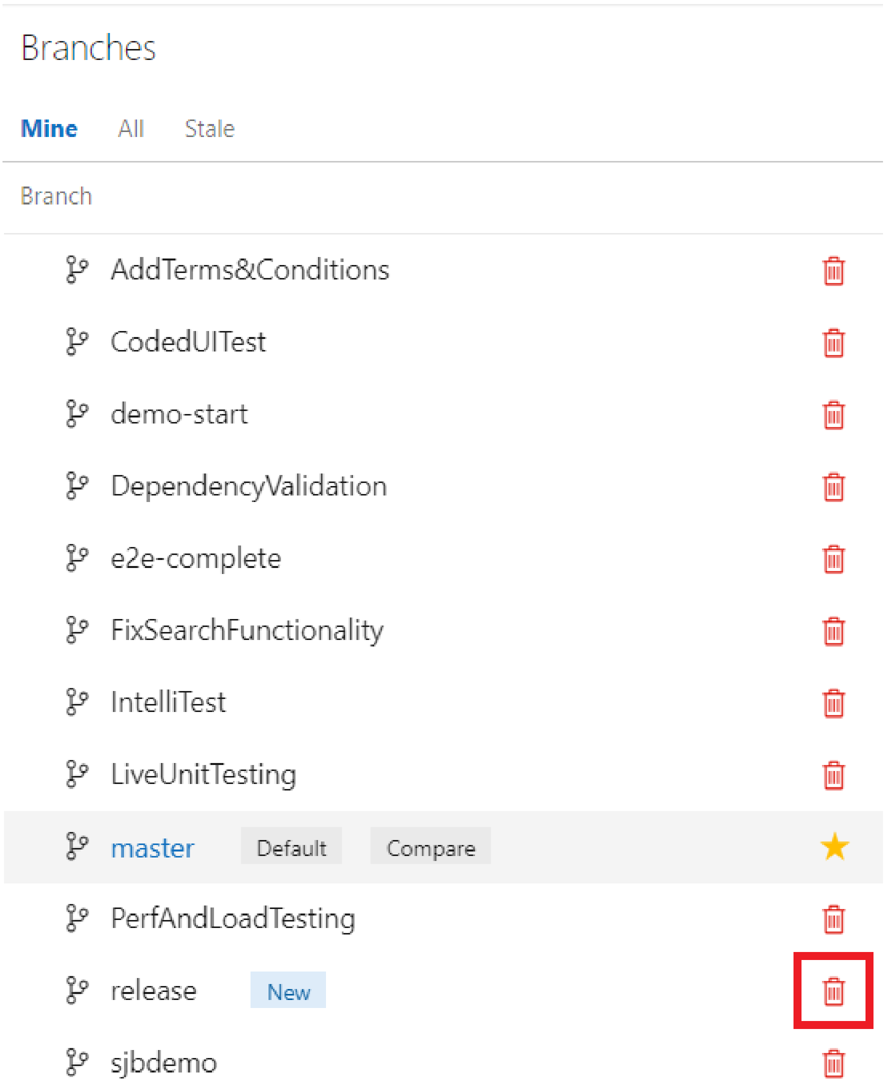
master 4988faaa

1.0.0.25 Tag at 816d9487

origin/release Remote branch at bd6f6757

Task 2: Deleting a branch

1. Return to Azure DevOps and click the **Delete** button that appears when you hover over the **release** branch to delete it.

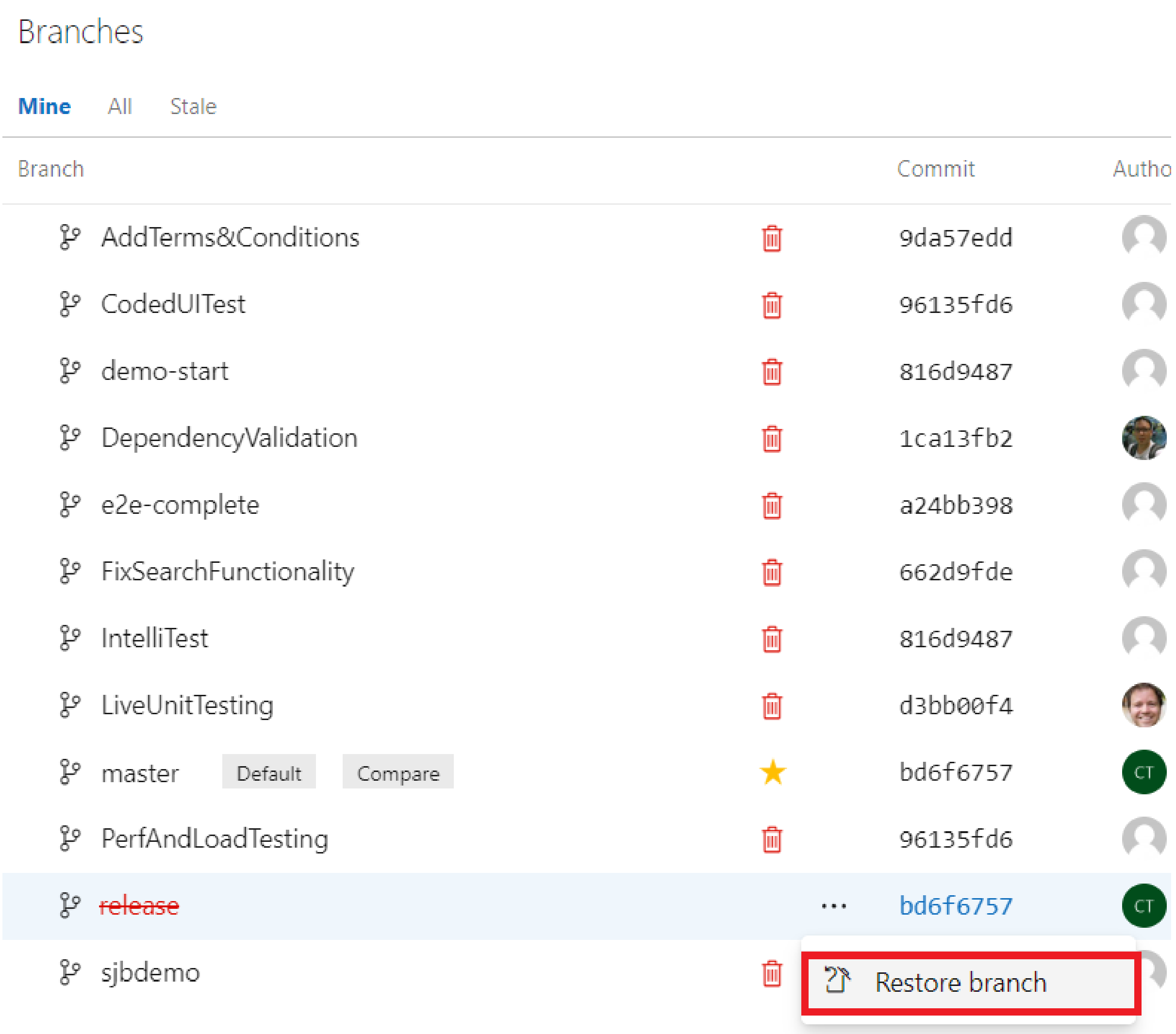


Branches

Mine All Stale

Branch	
AddTerms&Conditions	🗑
CodedUITest	🗑
demo-start	🗑
DependencyValidation	🗑
e2e-complete	🗑
FixSearchFunctionality	🗑
IntelliTest	🗑
LiveUnitTesting	🗑
master Default Compare	★
PerfAndLoadTesting	🗑
release New	🗑
sjbdemo	🗑

2. However, maybe we should keep it around for a little longer. From its context menu, select **Restore branch**.



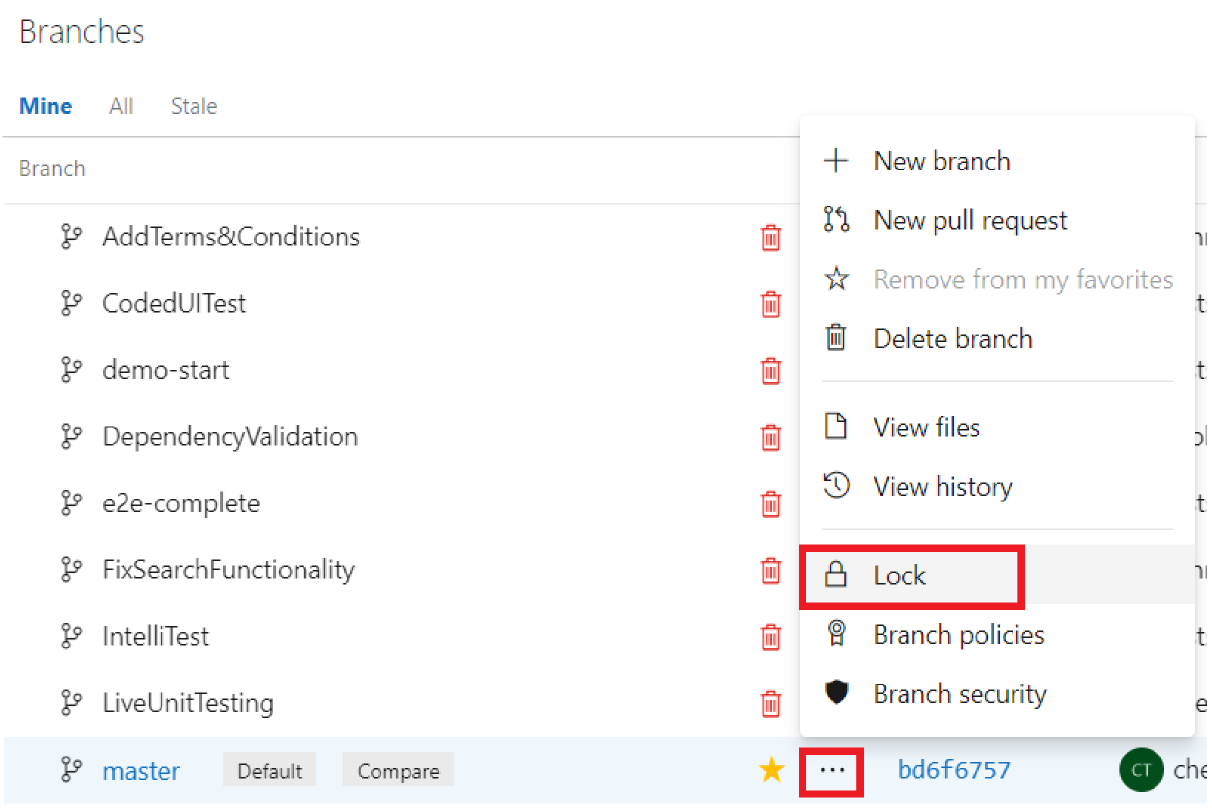
Branches

Mine All Stale

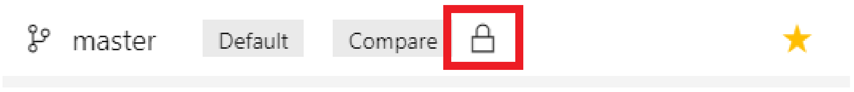
Branch		Commit	Autho
AddTerms&Conditions		9da57edd	
CodedUITest		96135fd6	
demo-start		816d9487	
DependencyValidation		1ca13fb2	
e2e-complete		a24bb398	
FixSearchFunctionality		662d9fde	
IntelliTest		816d9487	
LiveUnitTesting		d3bb00f4	
master Default Compare	★	bd6f6757	CT
PerfAndLoadTesting		96135fd6	
~~release~~	···	bd6f6757	CT
sjbdemo			

Task 3: Locking a branch

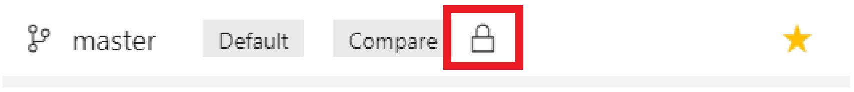
1. From the **master** context menu, select **Lock**.



2. The branch is now locked.

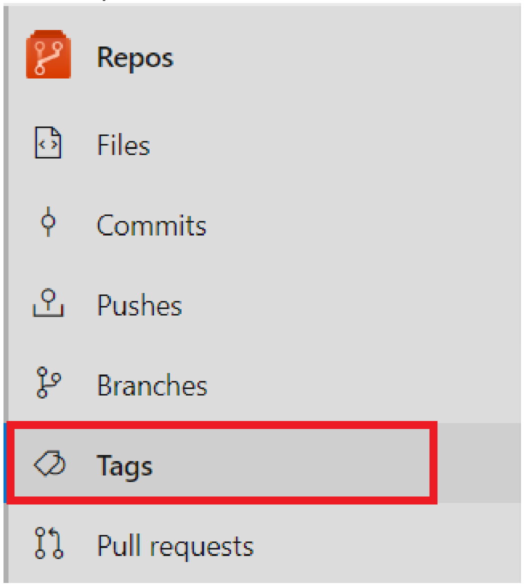


3. Now **Unlock** the branch using the same process.

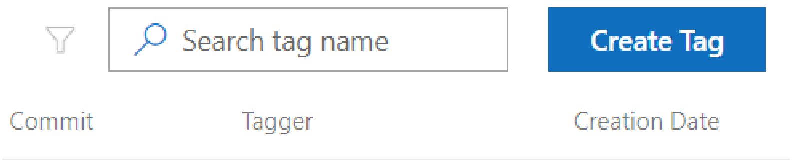


Task 4: Tagging a release

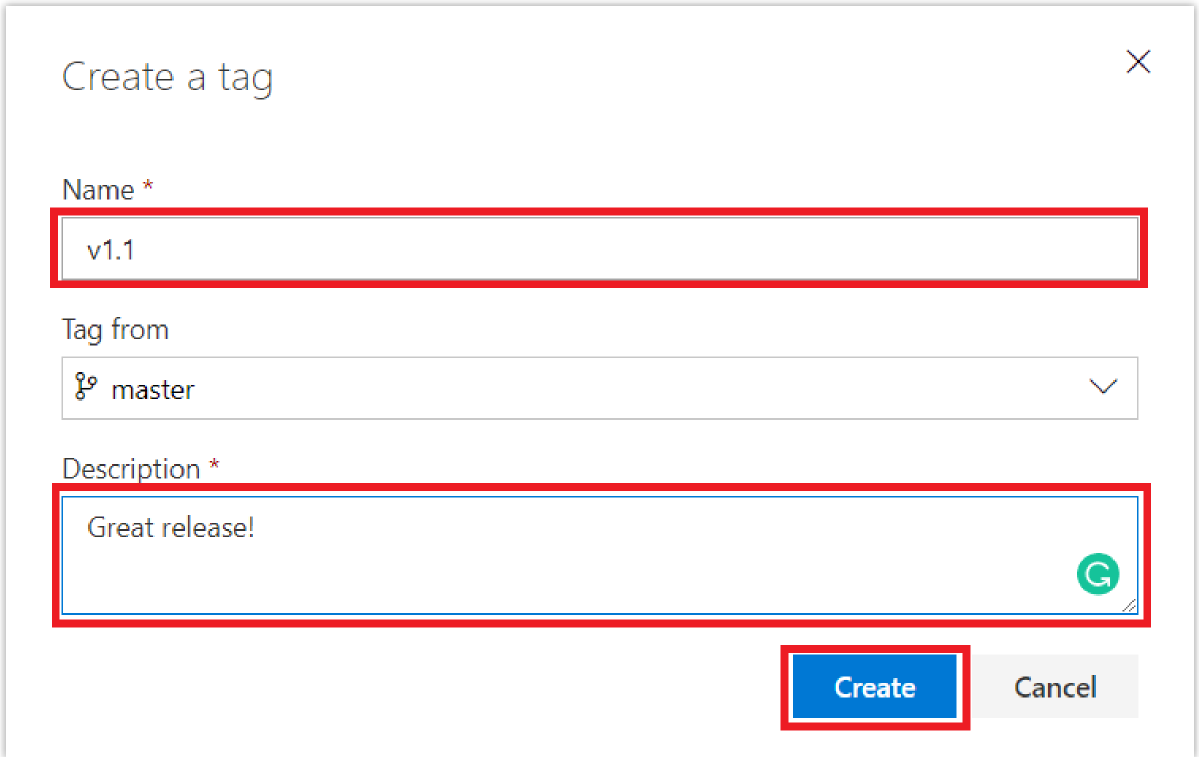
1. While it may not seem like much, the product team has decided that this version of the site is exactly what's needed for v1.1. In order to mark it as such, navigate to the **Tags** tab.



2. Click **Create Tag**.



3. Enter a **name** of "**v1.1**" and a **Description** of "**Great release!**". Click **Create**.

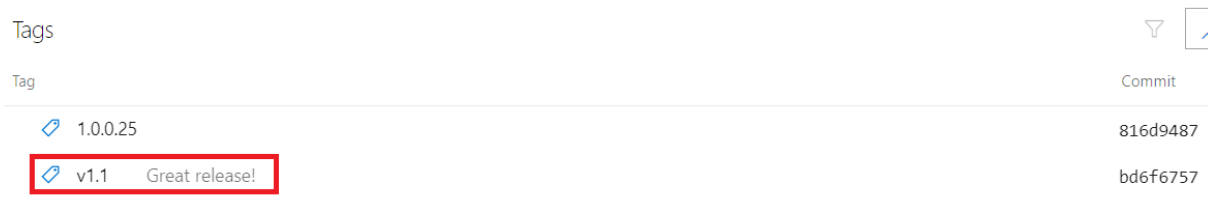

Create a tag

Name *
v1.1

Tag from
master

Description *
Great release!

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


Tags

Tag		Commit
1.0.0.25		816d9487
v1.1	Great release!	bd6f6757