Lab requirements

- This lab requires **Microsoft Edge** or an Azure DevOps supported browser.
- **Set up an Azure DevOps organization:** If you don't already have an Azure DevOps organization that you can use for this lab, create one by following the instructions available at Create an organization or project collection.
- Git for Windows download page. This will be installed as part of prerequisites for this lab.
- Visual Studio Code. This will be installed as part of prerequisites for this lab.

Lab overview

Azure DevOps supports two types of version control, Git and Team Foundation Version Control (TFVC). Here's a quick overview of the two version control systems:

- **Team Foundation Version Control (TFVC)**: 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.
- **Git**: Git is a distributed version control system. Git repositories can live locally (on a developer's machine). 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.

Git is the default version control provider for new projects. You should use Git for version control in your projects unless you need centralized version control features in TFVC.

You'll learn how to work with branches and repositories in Azure DevOps in this lab.

Objectives

After you complete this lab, you will be able to:

- Work with branches in Azure Repos.
- Work with repositories in Azure Repos.

Estimated timing: 30 minutes

Instructions

Note

If you completed **Lab 02: Version Controlling with Git in Azure Repos**, you have already finished the steps in **Exercise 0: Configure the lab prerequisites** and **Exercise 1: Clone an existing repository** and can skip ahead to **Exercise 2: Manage branches from Azure DevOps**.

Exercise 0: Configure the lab prerequisites

In this exercise, you will set up the prerequisites for the lab, which include the preconfigured Parts Unlimited team project based on an Azure DevOps Demo Generator template and a Visual Studio Code configuration.

Task 1: Configure the team project

In this task, you will use Azure DevOps Demo Generator to generate a new project based on the **Parts Unlimited** template.

1. On your lab computer, start a web browser and navigate to Azure DevOps Demo Generator. This utility site will automate the process of creating a new Azure DevOps project within your account that is prepopulated with content (work items, repos, etc.) required for the lab.
2. **Note**: For more information on the site, see https://docs.microsoft.com/en-us/azure/devops/demo-gen.
3. Click **Sign in** and sign in using the Microsoft account associated with your Azure DevOps subscription.
4. If required, on the **Azure DevOps Demo Generator** page, click **Accept** to accept the permission requests for accessing your Azure DevOps subscription.
5. On the **Create New Project** page, in the **New Project Name** textbox, type **Version Controlling with Git in Azure Repos**, in the **Select organization** dropdown list, select your Azure DevOps organization, and then click **Choose template**.
6. In the list of templates, locate the **PartsUnlimited** template and click **Select Template**.
7. Back on the **Create New Project** page, click **Create Project**
8. **Note**: Wait for the process to complete. This should take about 2 minutes. In case the process fails, navigate to your DevOps organization, delete the project, and try again.
9. On the **Create New Project** page, click **Navigate to project**.

Task 2: Install and configure Git and Visual Studio Code

In this task, you will install and configure Git and Visual Studio Code, including configuring the Git credential helper to securely store the Git credentials used to communicate with Azure DevOps. If you have already implemented these prerequisites, you can proceed directly to the next task.

10. If you don't have Git 2.29.2 or later installed yet, start a web browser, navigate to the Git for Windows download page download it, and install it.
11. If you don't have Visual Studio Code installed yet, from the web browser window, navigate to the Visual Studio Code download page, download it, and install it.
12. If you don't have Visual Studio C# extension installed yet, in the web browser window, navigate to the C# extension installation page and install it.
13. On the lab computer, open **Visual Studio Code**.
14. In the Visual Studio Code interface, from the main menu, select **Terminal | New Terminal** to open the **TERMINAL** pane.
15. Make sure that the current Terminal is running **PowerShell** by checking if the drop-down list at the top right corner of the **TERMINAL** pane shows **1: powershell**

16. **Note**: To change the current Terminal shell to **PowerShell** click the drop-down list at the top right corner of the **TERMINAL** pane and click **Select Default Shell**. At the top of the Visual Studio Code window select your preferred terminal shell **Windows PowerShell** and click the plus sign on the right-hand side of the drop-down list to open a new terminal with the selected default shell.

17. In the **TERMINAL** pane, run the following command below to configure the credential helper.

18. CodeCopy

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

20. In the **TERMINAL** pane, run the following commands to configure a user name and email for Git commits (replace the placeholders in braces with your preferred user name and email):

21. CodeCopy

```
22.     git config --global user.name "<John Doe>"
    git config --global user.email <johndoe@example.com>
```

Exercise 1: Clone an existing repository

In this exercise, you use Visual Studio Code to clone the Git repository you provisioned as part of the previous exercise.

Task 1: Clone an existing repository

In this task, you will step through the process of cloning a Git repository by using Visual Studio Code.

23. If needed, start a web browser, navigate to your Azure DevOps organization, and open the **Version Controlling with Git in Azure Repos** project you generated in the previous exercise.

24. **Note**: Alternatively, you can access the project page directly by navigating to the https://dev.azure.com/<your-Azure-DevOps-account-name>/Version%20Controlling%20with%20Git%20in%20Azure%20Repos URL, where the <your-Azure-DevOps-account-name> placeholder, represents your account name.

25. In the vertical navigational pane of the of the DevOps interface, select the **Repos** icon.
26. In the upper right corner of the **PartsUnlimited** pane, click **Clone**.

27. **Note**: 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.
28. On the **Clone Repository** panel, with the **HTTPS** Command line option selected, click the **Copy to clipboard** button next to the repo clone URL.
29. **Note**: You can use this URL with any Git-compatible tool to get a copy of the codebase.
30. Close the **Clone Repository** panel.
31. Switch to **Visual Studio Code** running on your lab computer.
32. Click the **View** menu header and, in the drop-down menu, click **Command Palette**.
33. **Note**: The Command Palette provides an easy and convenient way to access a wide variety of tasks, including those implemented as 3rd party extensions. You can use the keyboard shortcut **Ctrl+Shift+P** to open it.
34. At the Command Palette prompt, run the **Git: Clone** command.
35. **Note**: To see all relevant commands, you can start by typing **Git**.
36. In the **Provide repository URL or pick a repository source** text box, paste the repo clone URL you copied earlier in this task and press the **Enter** key.
37. Within the **Select Folder** dialog box, navigate to the C: drive, create a new folder named **Git**, select it, and then click **Select Repository Location**.
38. When prompted, log in to your Azure DevOps account.
39. After the cloning process completes, once prompted ,in the Visual Studio Code, click **Open** to open the cloned repository.
40. **Note**: You can ignore warnings you might receive regarding problems with loading of the project. The solution may not be in the state suitable for a build, but we're going to focus on working with Git, so building the project is not required.

Exercise 2: Manage branches from Azure DevOps

In this exercise, you will work with branches by using Azure DevOps. You can manage your repo branches directly from the Azure DevOps portal, in addition to the functionality available in Visual Studio Code.

Task 1: Create a new branch

In this task, you will create a branch by using the Azure DevOps portal and use fetch it by using Visual Studio Code.

41. Switch to the the web browser displaying your Azure DevOps organization with the **Version Controlling with Git in Azure Repos** project you generated in the previous exercise.

42. **Note**: Alternatively, you can access the project page directly by navigating to the [https://dev.azure.com/<your-Azure-DevOps-account-name>/Version%20Controlling%20with%20Git%20in%20Azure%20Repos) URL, where the <your-Azure-DevOps-account-name> placeholder, represents your account name.

43. In the web browser window, navigate to the **Repos** pane of the project and select **Branches**.
44. On the **Branches** pane, click **New branch**.
45. In the **Create a branch** panel, in the **Name** textbox, type **release**, ensure that **master** appears in the **Based on** dropdown list, in the **Work items to link** drop-down list, select one or more available work items, and click **Create**.
46. Switch to the **Visual Studio Code** window.
47. Press **Ctrl+Shift+P** to open the **Command Palette**.
48. At the **Command Palette** prompt, start typing **Git: Fetch** and select **Git: Fetch** when it becomes visible. This command will update the origin branches in the local snapshot.
49. In the lower left corner of the Visual Studio Code window, click the **master** entry again.
50. In the list of branches, select **origin/release**. This will create a new local branch called **release** and check it out.

Task 2: Delete and restore a branch

In this task, you will use the Azure DevOps portal to delete and restore the branch you created in the previous task.

51. Switch to the web browser displaying the **Mine** tab of the **Branches** pane in the Azure DevOps portal.
52. On the **Mine** tab of the **Branches** pane, hover the mouse pointer over the **release** branch entry to reveal the ellipsis symbol on the right side.
53. Click the ellipsis, in the pop-up menu, select **Delete branch**, and, when prompted for confirmation, click **Delete**.
54. On the **Mine** tab of the **Branches** pane, select the **All** tab.
55. On the **All** tab of the **Branches** pane, in the **Search branch name** text box, type **release**.
56. Review the **Deleted branches** section containing the entry representing the newly deleted branch.
57. In the **Deleted branches** section, hover the mouse pointer over the **release** branch entry to reveal the ellipsis symbol on the right side.
58. Click the ellipsis, in the pop-up menu and select **Restore branch**.

59. **Note**: You can use this functionality to restore a deleted branch as long as you know its exact name.

Task 3: Lock and unlock a branch

In this task, you will use the Azure DevOps portal to lock and unlock the master branch.

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 for locking it and make sure they know what to do to work with the branch after it is unlocked.

60. Switch to the web browser displaying the **Mine** tab of the **Branches** pane in the Azure DevOps portal.
61. On the **Mine** tab of the **Branches** pane, hover the mouse pointer over the **master** branch entry to reveal the ellipsis symbol on the right side.
62. Click the ellipsis and, in the pop-up menu, select **Lock**.
63. On the **Mine** tab of the **Branches** pane, hover the mouse pointer over the **master** branch entry to reveal the ellipsis symbol on the right side.
64. Click the ellipsis and, in the pop-up menu, select **Unlock**.

Task 4: Tag a release

In this task, you will use the Azure DevOps portal to tag a release in the Azure DevOps Repos.

The product team has decided that the current version of the site should be released as v1.1.0-beta.

65. In the vertical navigational pane of the of the Azure DevOps portal, in the **Repos** section, select **Tags**.
66. In the **Tags** pane, click **New tag**.
67. In the **Create a tag** panel, in the **Name** text box, type **v1.1.0-beta**, in the **Based on** drop-down list leave the **master** entry selected, in the **Description** text box, type **Beta release v1.1.0** and click **Create**.

68. **Note**: 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.

Exercise 3: Manage repositories

In this exercise, you will use the Azure DevOps portal to create and delete a Git repository in Azure DevOps Repos.

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.

Task 1: Create a new repo from Azure DevOps

In this task, you will use the Azure DevOps portal to create a Git repository in Azure DevOps Repos.

69. In the web browser displaying the Azure DevOps portal, in the vertical navigational pane, click the plus sign in the upper left corner, directly to the right of the project name and, in the cascading menu, click **New repository**.
70. In the **Create a repository** pane, in the **Repository type**, leave the default **Git** entry, in the **Repository name** text box, type **New Repo**, leave other settings with their default values, and click **Create**.

71. **Note**: You have the option to create a file named **README.md**. This would be the default markdown file that is rendered when someone navigates to the repo root with a web browser. Additionally, you can preconfigure the repo with a **.gitignore** file. This file specifies which files, based on naming pattern and/or path, to ignore from source control. There are multiple templates available that include the common patterns and paths to ignore based on the project type you are creating.

72. **Note**: At this point, your repo is available. You can now clone it with Visual Studio Code or any other git-compatible tool.

Task 2: Delete and rename Git repos

In this task, you will use the Azure DevOps portal to delete a Git repository in Azure DevOps Repos.

Sometimes you'll have a need to rename or delete a repo, which is just as easy.

73. In the web browser displaying the Azure DevOps portal, at the bottom of the vertical navigational pane, click **Project settings**.
74. In the **Project Settings** vertical navigational pane, scroll down to the **Repos** section and click **Repositories**.
75. On the **Repositories** tab of the **All Repositories** pane, hover the mouse pointer over the **New Repo** branch entry to reveal the ellipsis symbol on the right side.
76. Click the ellipsis, in the pop-up menu, select **Delete**, in the **Delete New Repo repository** panel, type **New Repo**, and click **Delete**.

Review

In this lab, you used the Azure DevOps portal to manage branches and repositories.