

# Installing and Using Git

This document will outline the steps to download/setup Git on your PC and show you how to use some basic features of Git. For answers to frequently asked questions, please see the [Git FAQ](#).

Please **NOTE this is NOT GitHub**. GitHub and Git are two different things. [We use Git](#).

- Audience
- 1.0 - Installation / Setup
  - 1.1 - Install
  - 1.2 - Configuration
- 2.0 - Using Git
  - 2.1 - Repository Organization
  - 2.2 - Safe Pull/Push Procedures
    - 2.2.1 - Cloning and Pulling a Repository
    - 2.2.2 - Pushing
  - 2.3 - Checking Your Code In Correctly
  - 2.4 - Code Reviews and Merging Your Code
    - 2.4.1 Merging into main
- 3.0 - Resources
  - 3.1 - Team Resources

## Audience

This document is intended for all development teams, engineering leadership, QA, and RM.

## 1.0 - Installation / Setup

This guide shows you how to install Git on your system, then it shows you how to configure it for CallMiner use.

### 1.1 - Install

Follow the steps in this section to install Git properly:

1. Link: <https://git-scm.com/downloads>
  - a. Make sure to download the proper version for your operating system.  **DO NOT DOWNLOAD THE PORTABLE VERSION** .
  2. Expand the box below to learn how to run the setup wizard for Git:  
Git Installation Wizard Steps and Options
1. Click **Next** in the initial **Information** screen.

**Information**

Please read the following important information before continuing.

When you are ready to continue with Setup, click Next.

## GNU General Public License

Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc.  
59 Temple Place - Suite 330, Boston, MA 02111-1307, USA

Everyone is permitted to copy and distribute verbatim copies  
of this license document, but changing it is not allowed.

### Preamble

The licenses for most software are designed to take away your  
freedom to share and change it. By contrast, the GNU General Public  
License is intended to guarantee your freedom to share and change

<https://gitforwindows.org/>

**Next**

**Cancel**

2. In the Select Destination Location screen, leave the default location in place and click **Next**.



## Select Destination Location

Where should Git be installed?



Setup will install Git into the following folder.

To continue, click Next. If you would like to select a different folder, click Browse.

**Browse...**

At least 293.4 MB of free disk space is required.

<https://gitforwindows.org/>

**Back**

**Next**

**Cancel**

3. In the Select Components screen, leave the default options in place and click **Next**.



## Select Components

Which components should be installed?

Select the components you want to install; clear the components you do not want to install. Click Next when you are ready to continue.

- Additional icons
- On the Desktop
- Windows Explorer integration
  - Git Bash Here
  - Git GUI Here
- Git LFS (Large File Support)
- Associate .git\* configuration files with the default text editor
- Associate .sh files to be run with Bash
- Check daily for Git for Windows updates
- (NEW!) Add a Git Bash Profile to Windows Terminal

Current selection requires at least 293.4 MB of disk space.

<https://gitforwindows.org/>

Back

Next

Cancel

4. In the Select Start Menu Folder, leave the default folder name in place and click **Next**.



## Select Start Menu Folder

Where should Setup place the program's shortcuts?



Setup will create the program's shortcuts in the following Start Menu folder.

To continue, click Next. If you would like to select a different folder, click Browse.

A text input field containing the text "Git".Browse...A button labeled "Browse..." for selecting a different folder.

**Don't create a Start Menu folder**

<https://gitforwindows.org/>

BackNextCancel

5. In the Choosing the Default Editor Used by Git screen, choose your favorite editor such as Vim, Notepad++, or Visual Studio Code, then click **Next**.



## Choosing the default editor used by Git

Which editor would you like Git to use?



Use Vim (the ubiquitous text editor) as Git's default editor

**Use Vim (the ubiquitous text editor) as Git's default editor**

Use Notepad++ as Git's default editor

Use Visual Studio Code as Git's default editor

Use Visual Studio Code Insiders as Git's default editor

Use Sublime Text as Git's default editor

Use Atom as Git's default editor

Use VSCode as Git's default editor

Use Notepad as Git's default editor

may set it to some other editor of your choice.

<https://gitforwindows.org/>

Back

**Next**

Cancel

- In the Adjusting the Name of the Initial Branch in New Repositories screen, select the **Override the default branch name for new repositories** option and enter `main` as the name of the initial branch. Click **Next**.

**Adjusting the name of the initial branch in new repositories**

What would you like Git to name the initial branch after "git init"?

 **Let Git decide**

Let Git use its default branch name (currently: "master") for the initial branch in newly created repositories. The Git project [intends](#) to change this default to a more inclusive name in the near future.

 **Override the default branch name for new repositories**

**NEW!** Many teams already renamed their default branches; common choices are "main", "trunk" and "development". Specify the name "git init" should use for the initial branch:

This setting does not affect existing repositories.

<https://gitforwindows.org/>

[Back](#)[Next](#)[Cancel](#)

7. In the Adjusting Your PATH Environment screen, leave the **Git from the command line and also from 3rd-party software** default option in place, then click **Next**.



## Adjusting your PATH environment

How would you like to use Git from the command line?

**Use Git from Git Bash only**

This is the most cautious choice as your PATH will not be modified at all. You will only be able to use the Git command line tools from Git Bash.

**Git from the command line and also from 3rd-party software**

**(Recommended)** This option adds only some minimal Git wrappers to your PATH to avoid cluttering your environment with optional Unix tools.

You will be able to use Git from Git Bash, the Command Prompt and the Windows PowerShell as well as any third-party software looking for Git in PATH.

**Use Git and optional Unix tools from the Command Prompt**

Both Git and the optional Unix tools will be added to your PATH.

**Warning:** This will override Windows tools like "find" and "sort". Only use this option if you understand the implications.

<https://gitforwindows.org/> —

[Back](#)

[Next](#)

[Cancel](#)

8. In the Choosing the SSH Executable screen, leave the **Use bundled OpenSSH** default option in place, then click **Next**.



### Choosing the SSH executable

Which Secure Shell client program would you like Git to use?

**Use bundled OpenSSH**

This uses ssh.exe that comes with Git.

**Use external OpenSSH**

**NEW!** This uses an external ssh.exe. Git will not install its own OpenSSH (and related) binaries but use them as found on the PATH.

<https://gitforwindows.org/>

Back

Next

Cancel

9. In the Choosing HTTPS Transport Backend screen, select the **Use the native Windows Secure Channel Library** option, then click **Next**.

**Choosing HTTPS transport backend**

Which SSL/TLS library would you like Git to use for HTTPS connections?

 **Use the OpenSSL library**

Server certificates will be validated using the ca-bundle.crt file.

 **Use the native Windows Secure Channel library**

Server certificates will be validated using Windows Certificate Stores.

This option also allows you to use your company's internal Root CA certificates distributed e.g. via Active Directory Domain Services.

<https://gitforwindows.org/>

[Back](#)[Next](#)[Cancel](#)

10. In the Configuring the Line Ending Conversions screen, leave the **Checkout Windows-style, commit Unix- style line endings** default option in place, then click **Next**.

**Configuring the line ending conversions**

How should Git treat line endings in text files?

 **Checkout Windows-style, commit Unix-style line endings**

Git will convert LF to CRLF when checking out text files. When committing text files, CRLF will be converted to LF. For cross-platform projects, this is the recommended setting on Windows ("core.autocrlf" is set to "true").

 **Checkout as-is, commit Unix-style line endings**

Git will not perform any conversion when checking out text files. When committing text files, CRLF will be converted to LF. For cross-platform projects, this is the recommended setting on Unix ("core.autocrlf" is set to "input").

 **Checkout as-is, commit as-is**

Git will not perform any conversions when checking out or committing text files. Choosing this option is not recommended for cross-platform projects ("core.autocrlf" is set to "false").

<https://gitforwindows.org/>

[Back](#)[Next](#)[Cancel](#)

11. In the Configuring the terminal emulator use with Git Bash screen, leave the **Use MinTTY (the default terminal of MSYS2)** default option in place, then click **Next**.



## Configuring the terminal emulator to use with Git Bash

Which terminal emulator do you want to use with your Git Bash?



### Use MinTTY (the default terminal of MSYS2)

Git Bash will use MinTTY as terminal emulator, which sports a resizable window, non-rectangular selections and a Unicode font. Windows console programs (such as interactive Python) must be launched via `winpty` to work in MinTTY.

### Use Windows' default console window

Git will use the default console window of Windows ("cmd.exe"), which works very well with Win32 console programs such as interactive Python or node.js, but has a very limited default scroll-back, needs to be configured to use a Unicode font in order to display non-ASCII characters correctly, and prior to Windows 10 its window was not freely resizable and it only allowed rectangular text selections.

<https://gitforwindows.org/>

Back

Next

Cancel

12. In the Choose the Default Behavior of 'Git Pull' screen, leave the **Default (fast-forward or merge)** default option in place and click **Next**.



## Choose the default behavior of `git pull`

What should `git pull` do by default?



### Default (fast-forward or merge)

This is the standard behavior of `git pull` : fast-forward the current branch to the fetched branch when possible, otherwise create a merge commit.

### Rebase

Rebase the current branch onto the fetched branch. If there are no local commits to rebase, this is equivalent to a fast-forward.

### Only ever fast-forward

Fast-forward to the fetched branch. Fail if that is not possible.

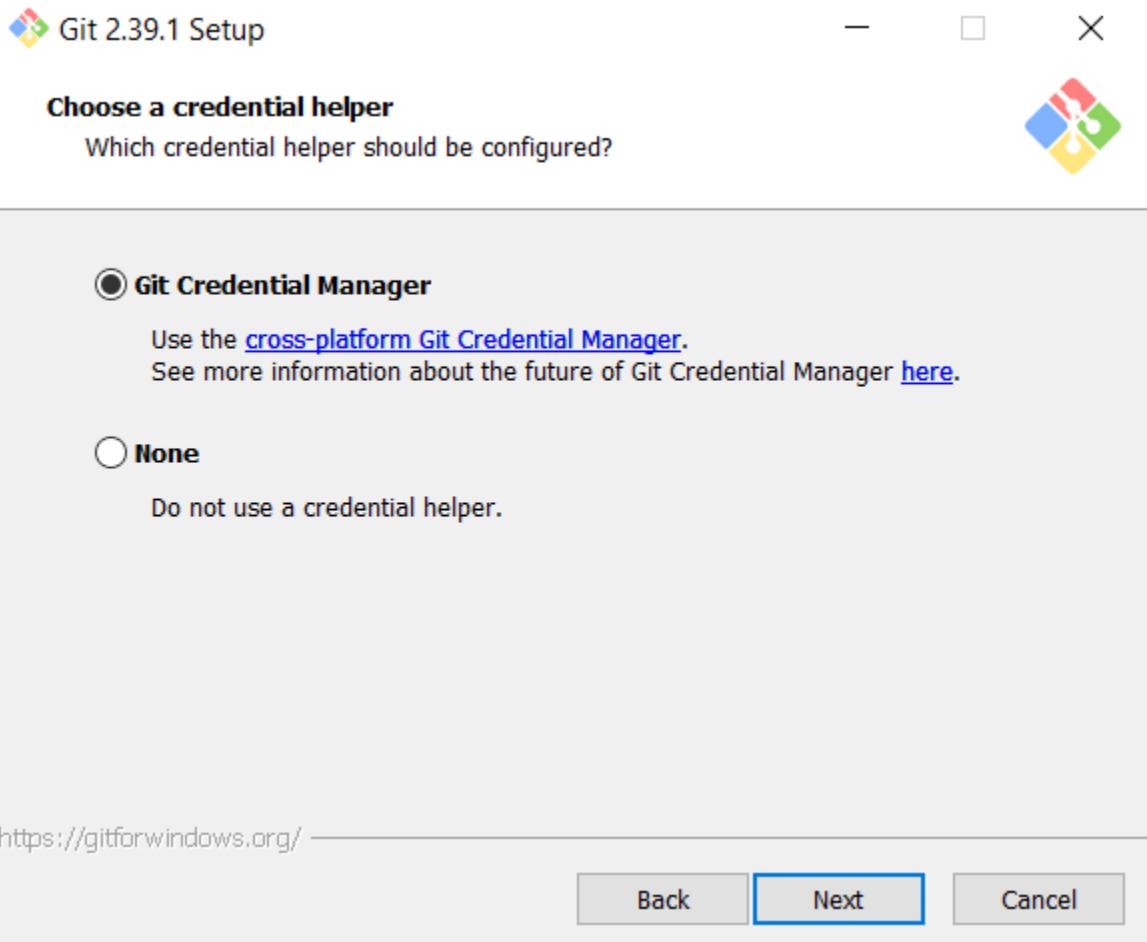
<https://gitforwindows.org/>

Back

Next

Cancel

13. In the Choose a Credential Helper screen, leave the **Git Credential Manager** default option in place and click **Next**.



14. In the Configuring Extra Options screen, leave the **Enable file system caching** option in place and click **Next**.



### Configuring extra options

Which features would you like to enable?

**Enable file system caching**

File system data will be read in bulk and cached in memory for certain operations ("core.fscache" is set to "true"). This provides a significant performance boost.

**Enable symbolic links**

Enable [symbolic links](#) (requires the SeCreateSymbolicLink permission). Please note that existing repositories are unaffected by this setting.

<https://gitforwindows.org/>

Back

Next

Cancel

15. In the Configuring Experimental Options screen, **do not select** either option and instead click **Install**.



## Configuring experimental options

These features are developed actively. Would you like to try them?



### Enable experimental support for pseudo consoles.

(NEW!) This allows running native console programs like Node or Python in a Git Bash window without using winpty, but it still has known bugs.

### Enable experimental built-in file system monitor

(NEW!) Automatically run a [built-in file system watcher](#), to speed up common operations such as `git status`, `git add`, `git commit`, etc in worktrees containing many files.

<https://gitforwindows.org/>

Back

Install

Cancel

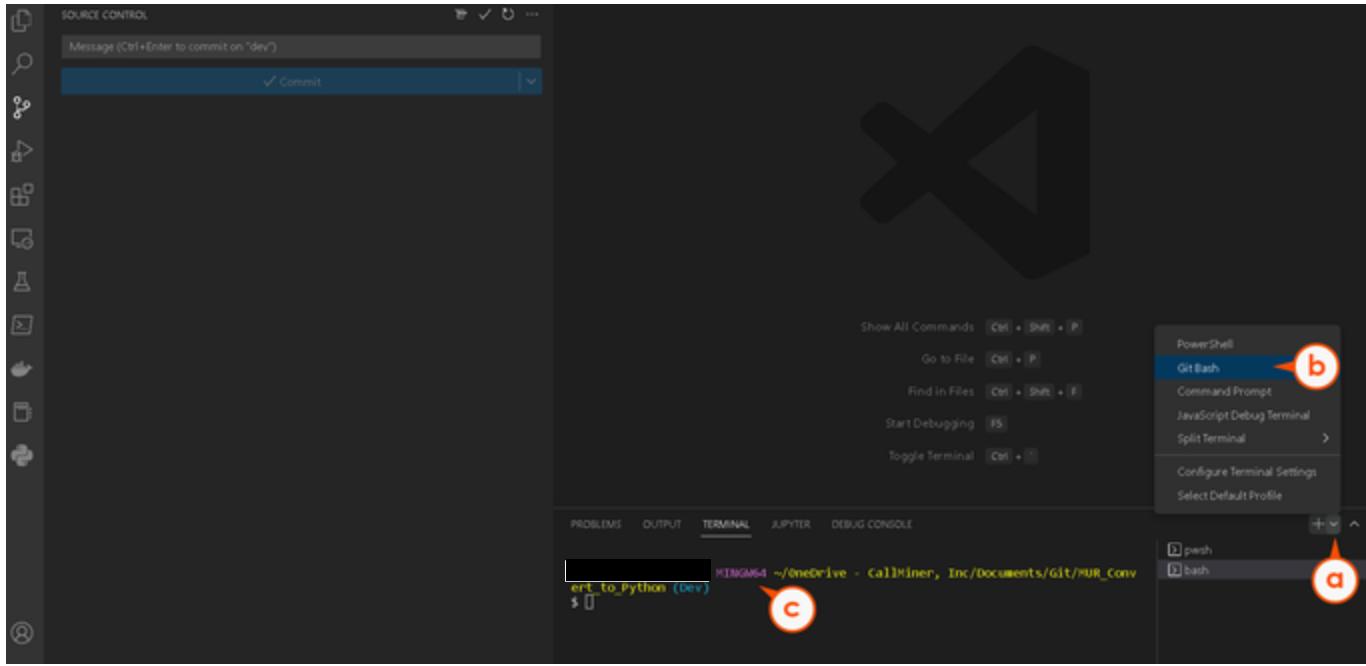
You have completed installing Git.

## 1.2 - Configuration

This section will show you how to configure Git now that you have it installed:

1. Ensure you are connected to a CallMiner VPN.
2. Run the Git application or alternatively, run the code editor you selected during Git installation.
  - a. Clone a new CallMiner repository (see [Pulling](#) for the steps to clone a repo) by [entering the link to the repo you need to use](#) (check with your team to make sure you use the proper Repo link). If you get an SSL error, then do the following. Use the below image to help guide you in the process:
    - i. In the code editor, open the terminal (a) and select **Git Bash** (b). Git Bash is essentially the Git app.
    - ii. Type the following into the terminal (c):
      1. `git config --global http.sslVerify true.`
      2. `git config --global http.sslbackend schannel.`
  - b. Configure your user.name and user.email. In the terminal (c), type the following:
    - i. `git config --global user.name "<your_name>"`  
1. Example: John Smith
    - ii. `git config --global user.email "<your_work_email>"`  
1. Example: john.smith@callminer.com

You have now completed configuring Git.



## 2.0 - Using Git

- This section will show you how to use Git safely to pull/push your code. Please use the safe practices described below when operating on the repository to prevent loss of data, work, and time.
- For more information on how Git branching works, see <https://launchdarkly.com/blog/git-branching-strategies-vs-trunk-based-development/>.

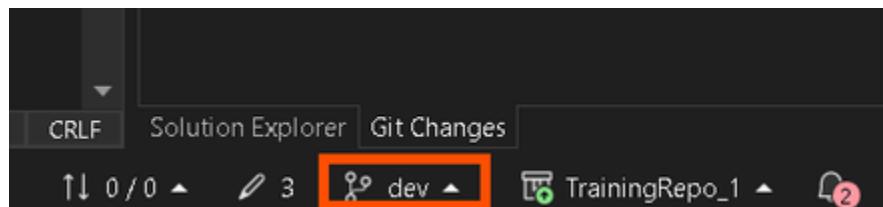
### 2.1 - Repository Organization

We use different branches to safely create our software while working together from different computers. The branches are used as follows:

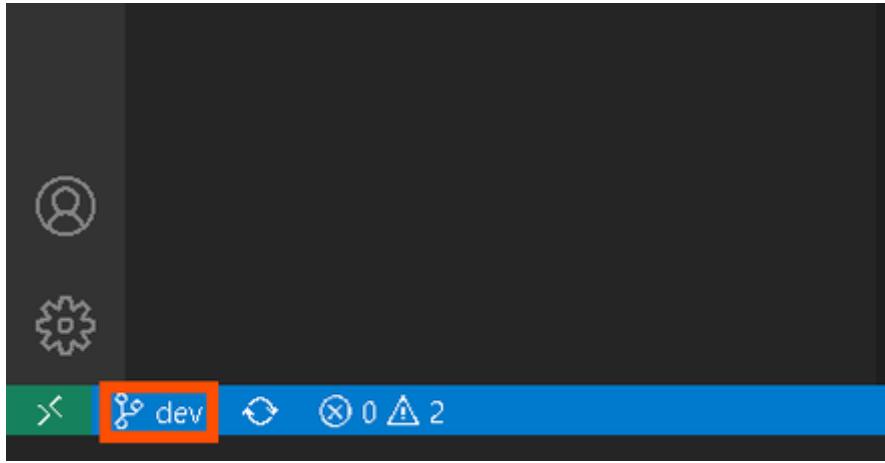
1. **dev (or your version of the dev branch)**
  - a. This is the branch where all development work is merged into. It's also used to build the dev and QA environments.
2. **main (or your version of trunk)**
  - a. This is the release branch, where code is ready for production.

```
git dev 532bb5c5
git main 95178b15
```

Remember that when you're working in **Visual Studio**, you can select the branch by clicking the bottom right of the VS window and clicking the branch name.



In **Visual Studio Code**, you can select the branch that you're working in from the bottom of the VS Code window by clicking the branch name.



## 2.2 - Safe Pull/Push Procedures

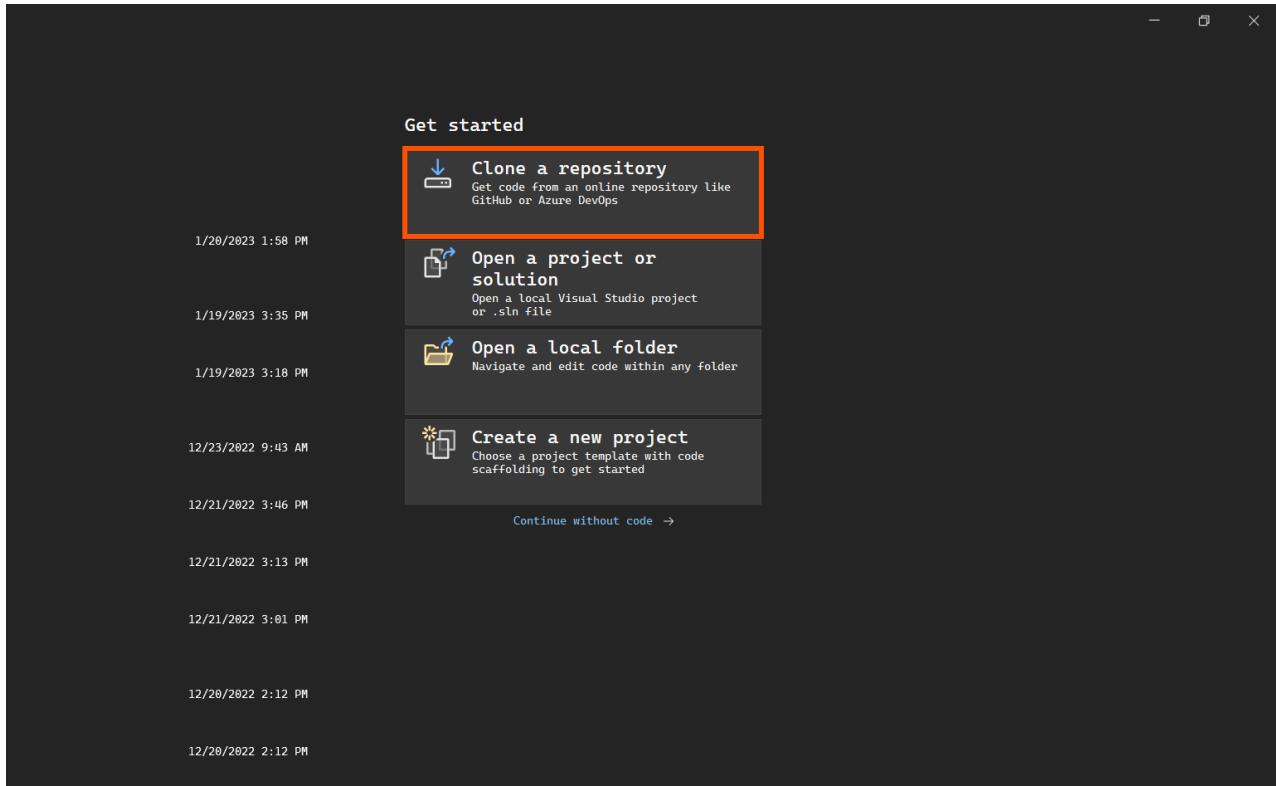
### 2.2.1 - Cloning and Pulling a Repository

It is good practice to frequently do a **Pull** while developing or prior to pushing to stay in sync with the repo. You'll need to do this in both Visual Studio and Visual Studio Code.

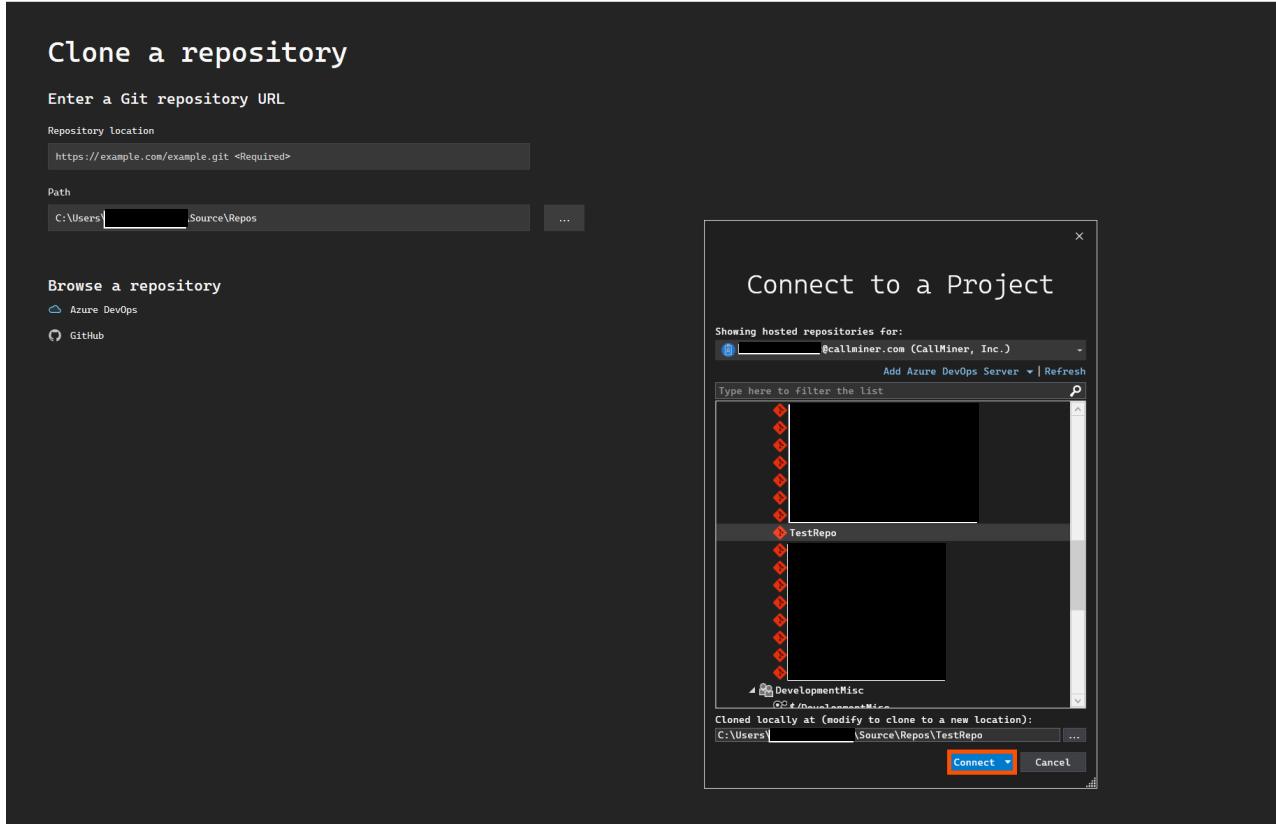
**⚠️ Important:** When pulling into your working branch, you will see a warning telling you that line ending changes do not match. To get rid of the warning, you need to commit and push changes. **There are no actual errors occurring;** the auto-line-ending configuration developed by RM will fix the line ending when you commit your change. Upon a developer first committing a change into Git, this error will go away for all developers going forward.

#### ▼ Visual Studio

1. For organizational purposes, you can create a folder on your machine that you designate as your testing folder. Create a subfolder in this folder named after a project you plan to work on. This will be the folder you'll clone a repo to.
  - a. For example, your subfolder directory could follow this template: %userprofile%\Source\Repos\Project Name>\<Project\_repo>
    - i. When written out, the directory would look like this: C:\Users\john.smith\Source\Repos\NGM\AccuracyRunMS
2. Click **Clone a repository**.

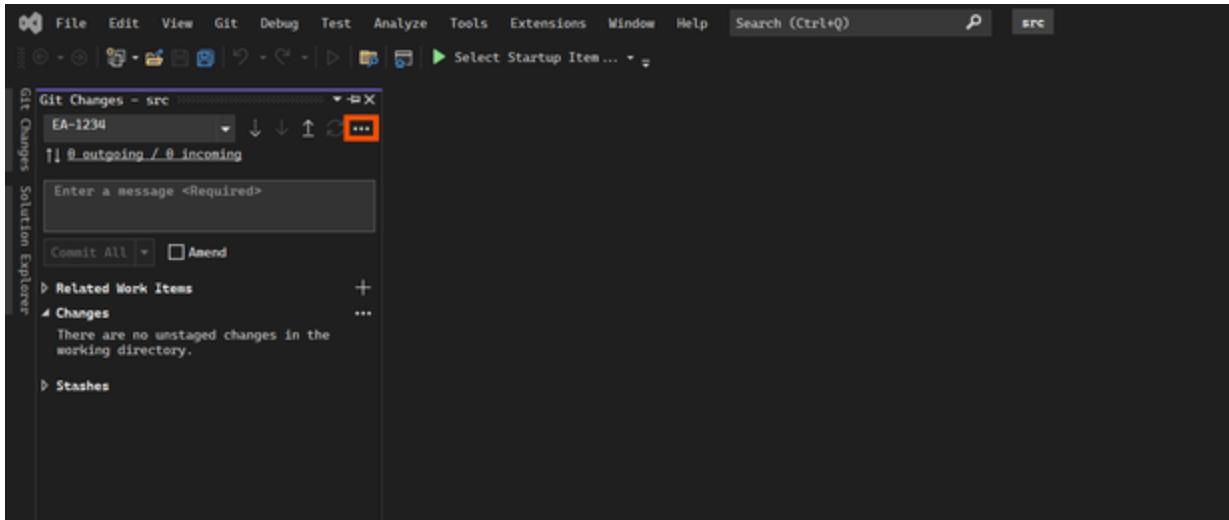


3. Choose a project to connect to. Enter the location of the subfolder you created in step 1 in the **Cloned locally at** field, then click **Connect**. If you're not connected to the Azure DevOps server, enter the URL for the server in the **Repository location** field or by clicking **Add Azure DevOps Server** in the **Connect to a Project** window.

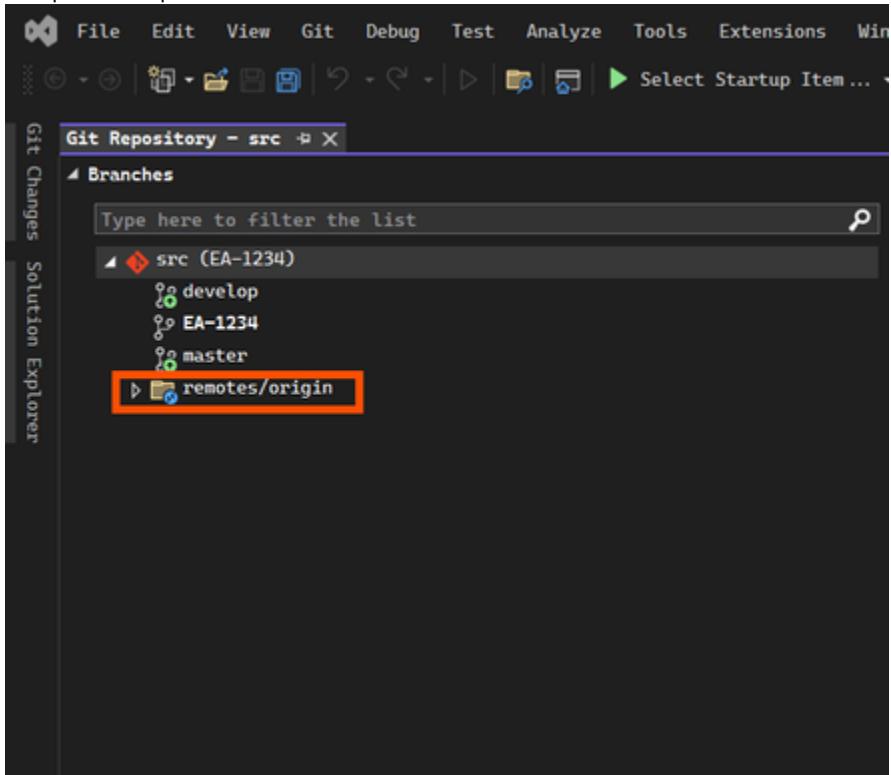


Once you clone a repo, you have the latest version of whatever exists in that repo; however, later you'll need to pull latest from dev to make sure that everything you have in your working branch is current.

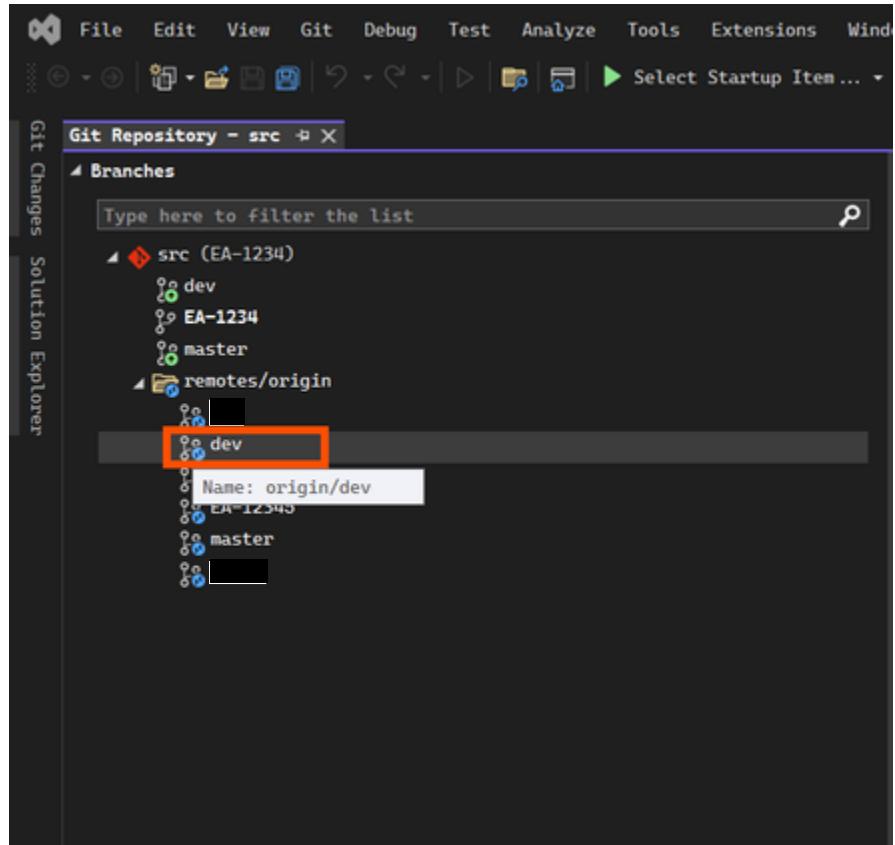
1. Click the dot menu and choose **Manage Branches**.



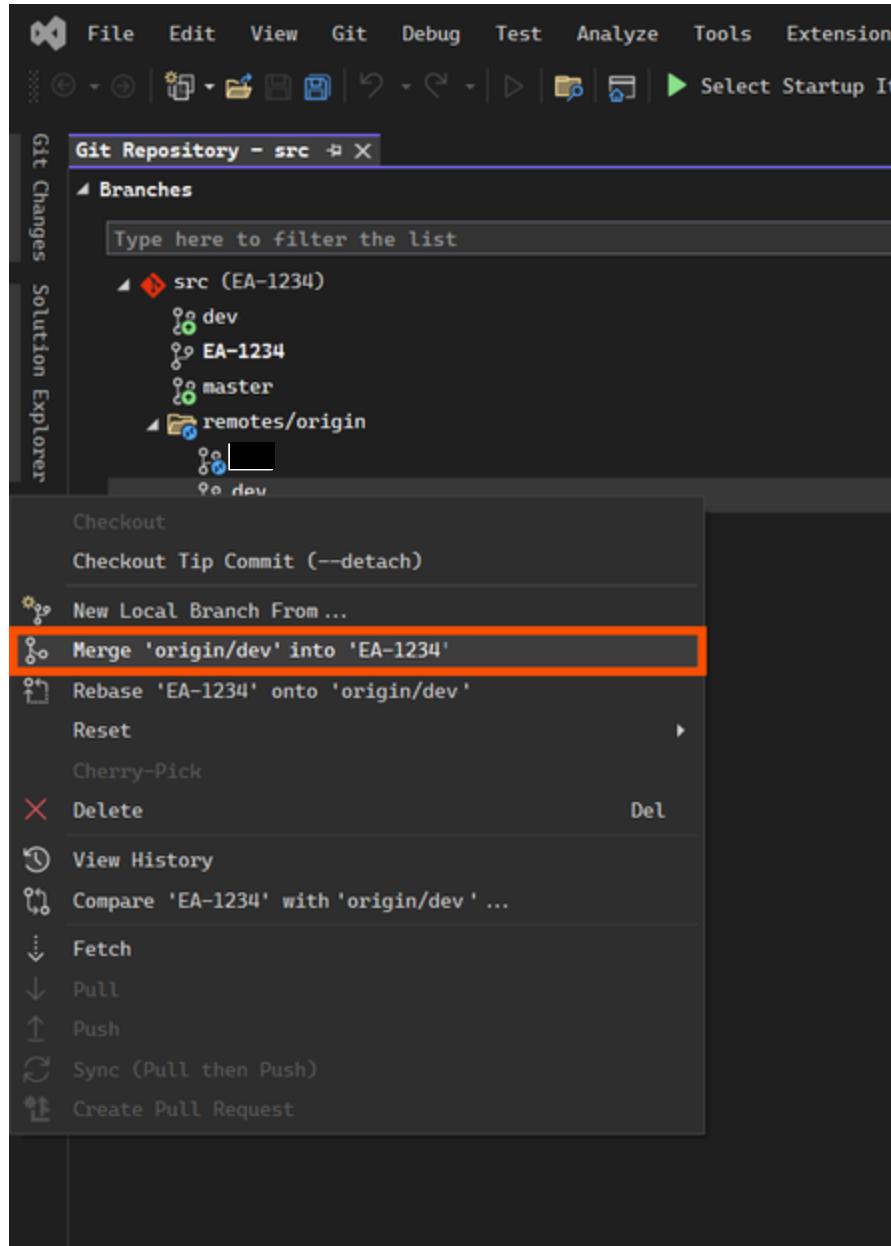
2. Click **remotes/origin** to open the drop-down menu.



3. Right-click the dev branch.



4. Click **Merge origin/dev into your working branch**. This will merge the most recent updates from dev into your working branch.

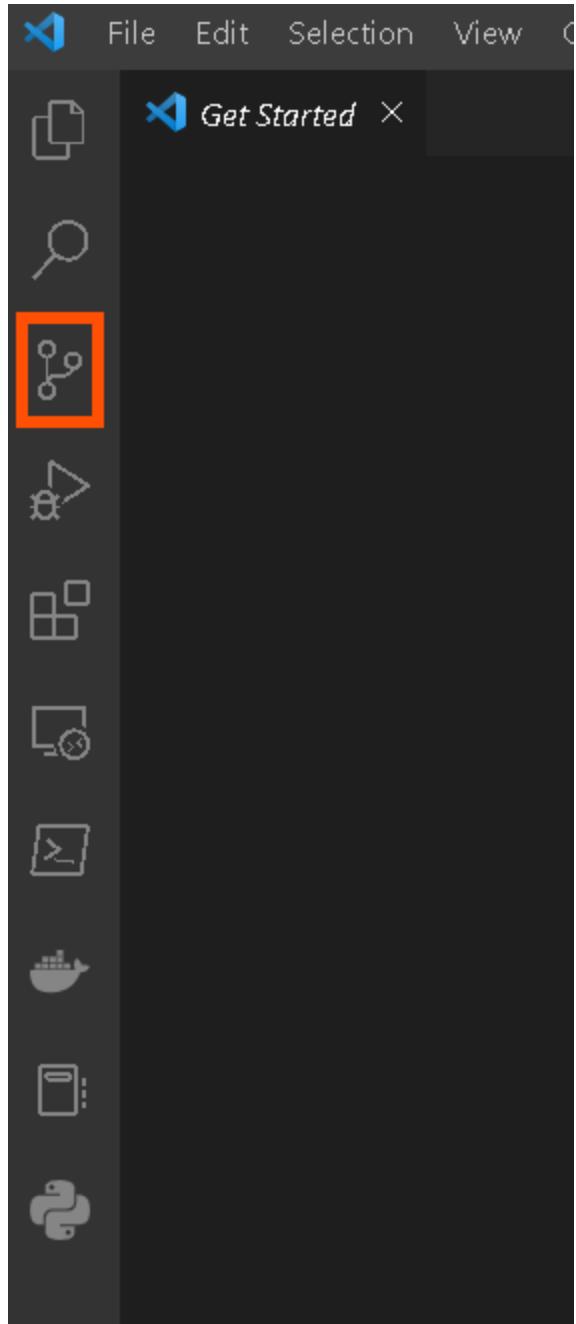


You now have the latest version of dev in your working branch.

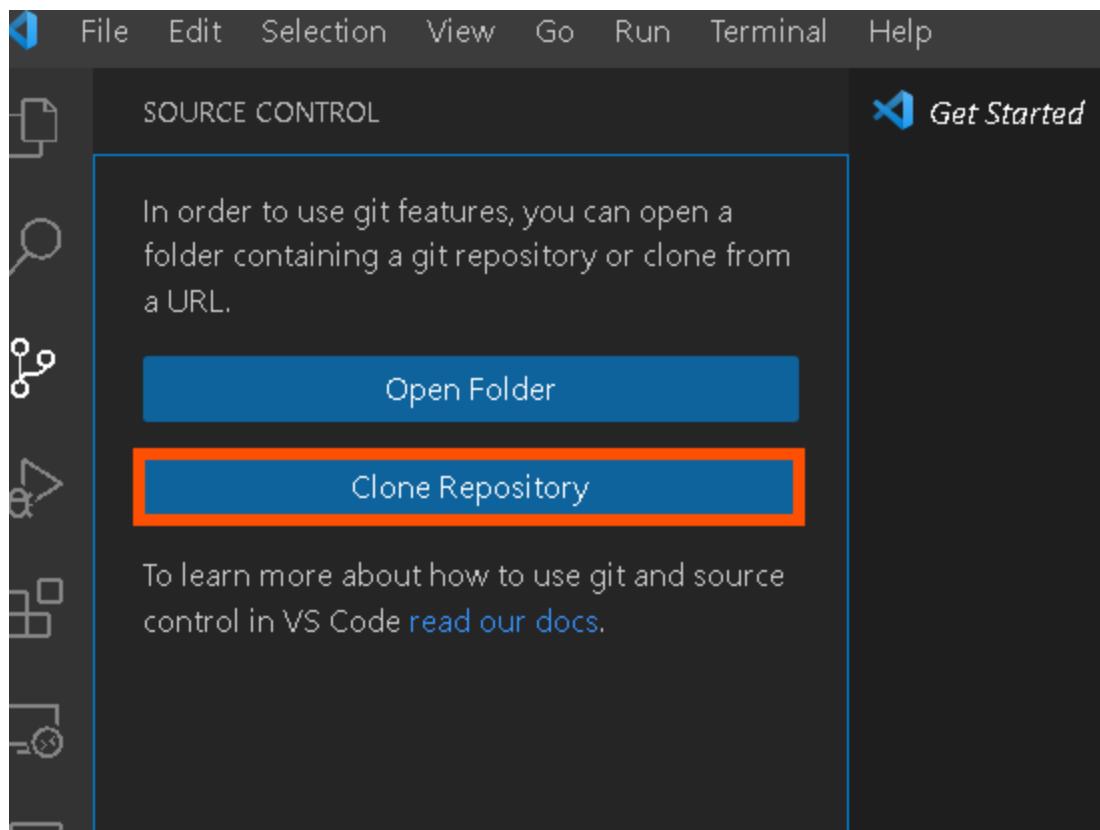
#### Visual Studio Code

Before you can pull a repo, you need to clone the repo. Follow the steps below to clone a repo:

1. For organizational purposes, you can create a folder on your machine that you designate as your testing folder. Create a subfolder in this folder named after a project you plan to work on. This will be the folder you'll clone a repo to.
  - a. For example, your subfolder directory could follow this template: %userprofile%\Source\Repos\Project Name>\<Project\_repo>
    - i. When written out, the directory would look like this: C:\Users\john.smith\Source\Repos\██████████\██████████
2. Click **Source Control**.



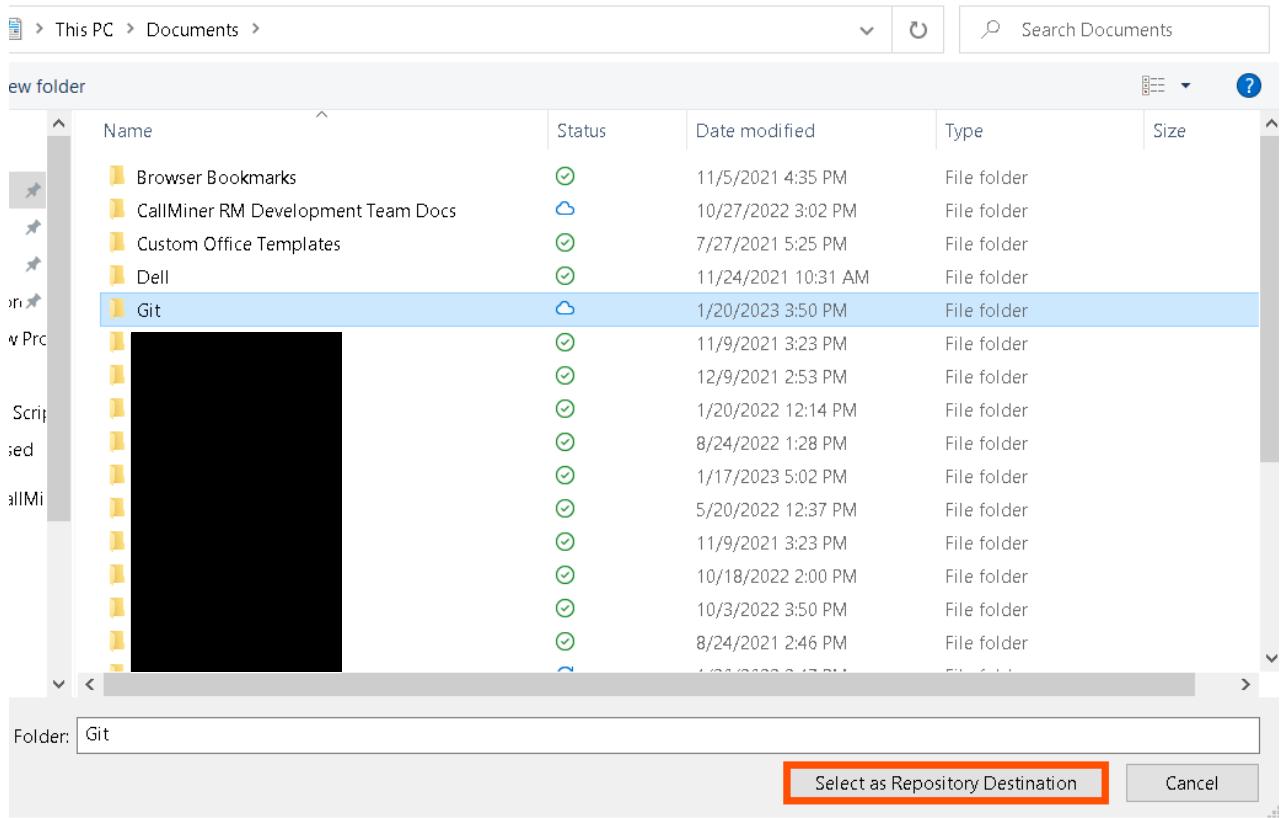
3. Click **Clone Repository**. However, if you don't see the option, click the **Refresh** link.



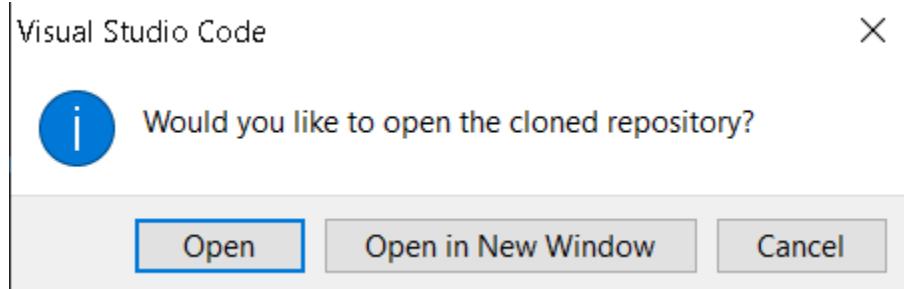
4. Paste the repository link into the field (do not select GitHub. We use Git). You can retrieve the link from the repo in Azure DevOps.

A screenshot of the "Clone Repository" dialog in VS Code. It has a text input field with placeholder text "Provide repository URL or pick a repository source." and a button "Clone from GitHub". To the right of the button is the text "remote sources".

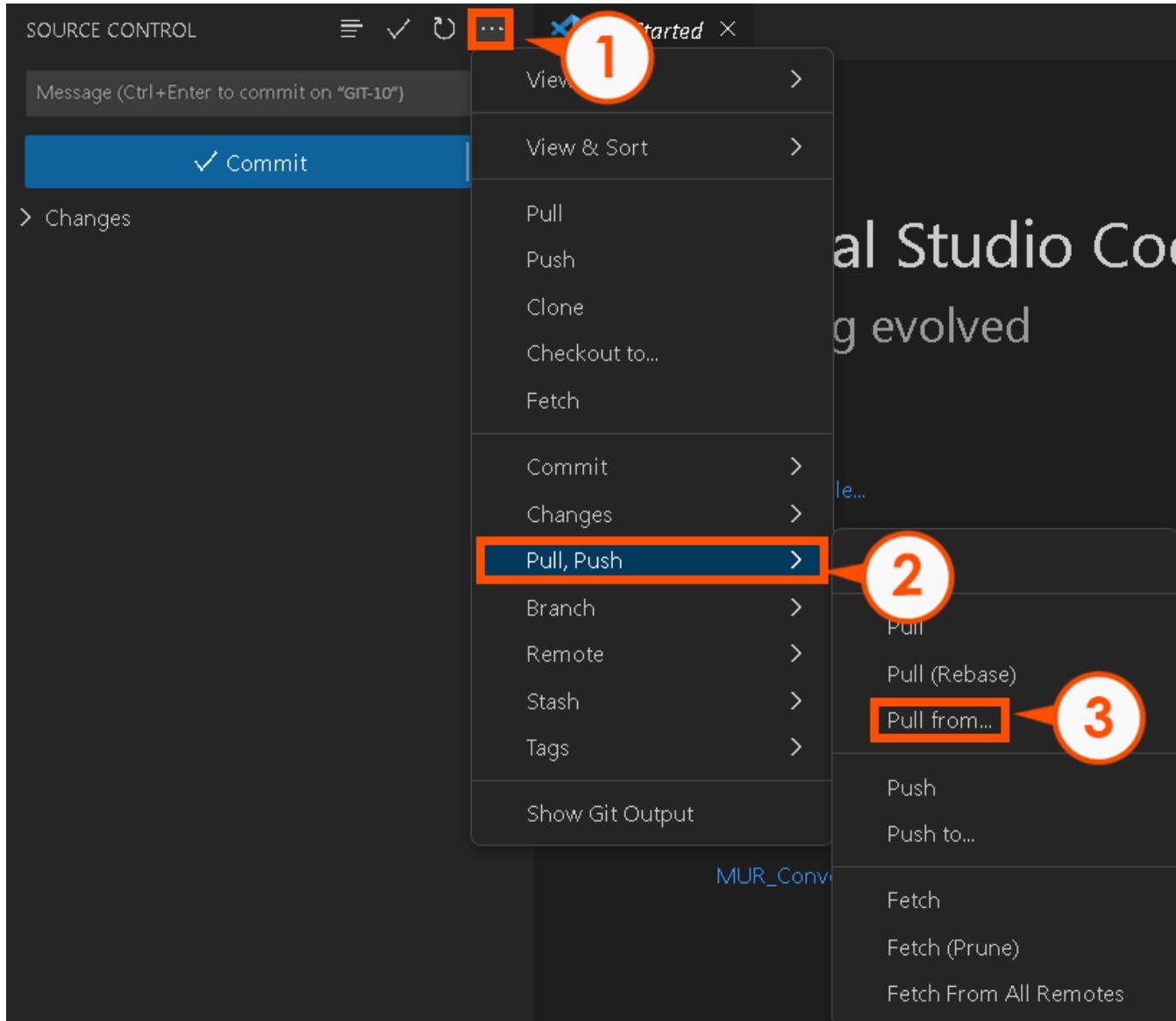
5. Find a location in your system to store the repo, or use the subfolder you created in step 1, then click **Select as Repository Destination**.



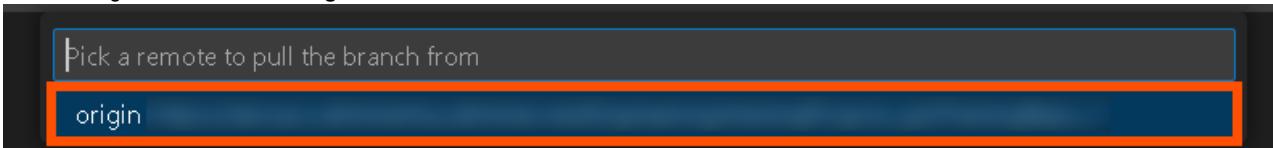
6. Click Open.



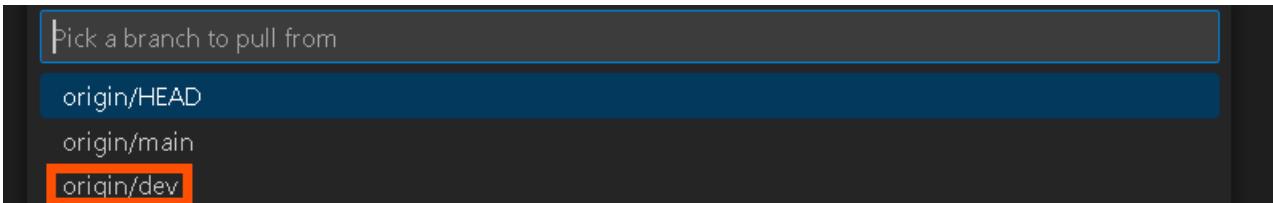
You have successfully cloned your repo. Just like in Visual Studio, once you need to get the latest updates from dev, you'll need to pull into your working branch. Follow the steps below to pull from dev:



1. Click the three dots in Source Control.
2. Click **Pull, Push**.
3. Click **Pull from...** to choose which repo to pull from.
4. After clicking **Pull from....**, click **origin**.



5. Select the **dev** branch.



You now have the latest version of **dev** in your working branch.

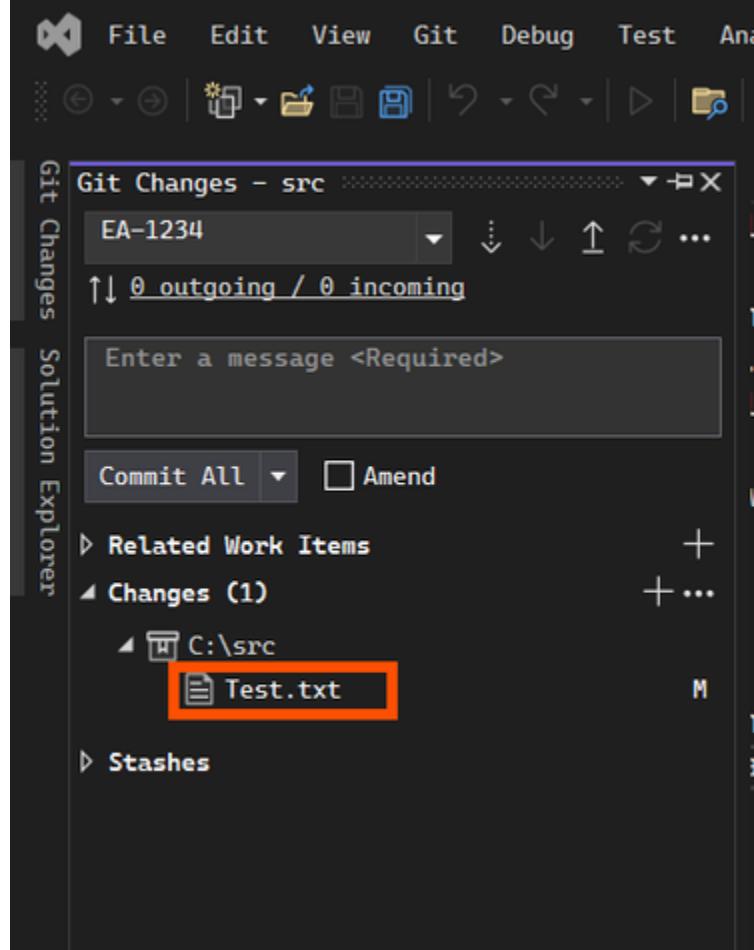
## 2.2.2 - Pushing

It is good practice to frequently do a **Pull** while developing or prior to pushing to get the latest changes. Be sure to always do a Pull to get latest in your working branch.

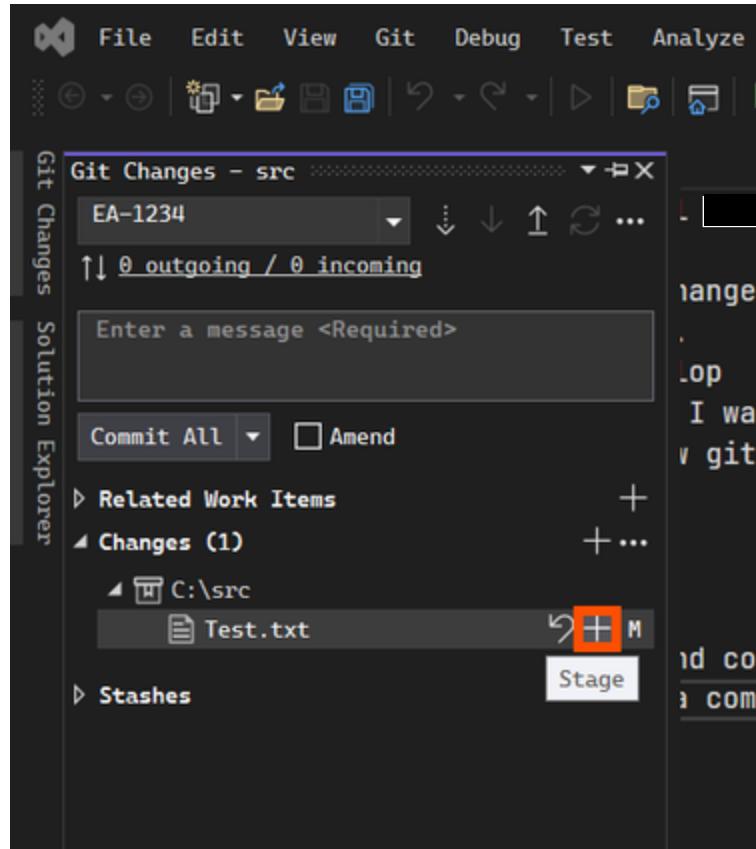
When the time comes to push the latest version of your files to a remote repo, you need to **push** your files. Follow the steps below to learn how to do this in both Visual Studio and Visual Studio Code:

▼ [Visual Studio](#)

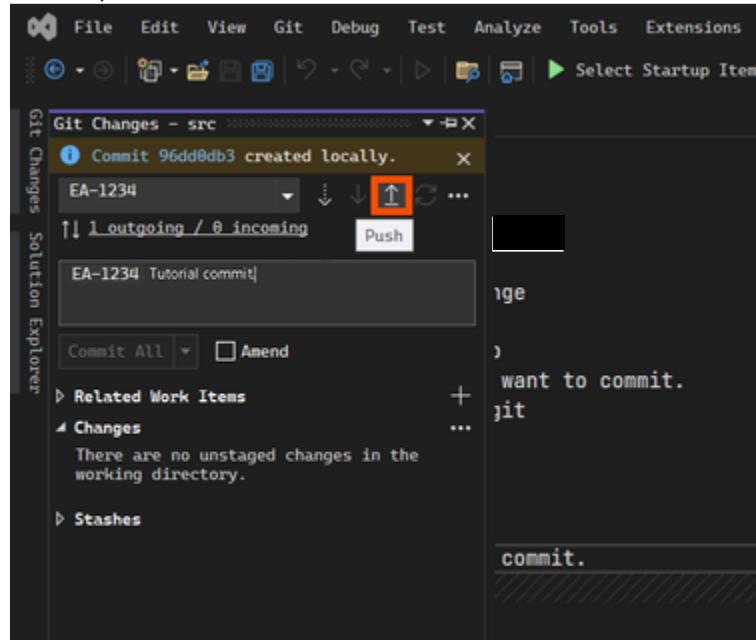
1. In the **Git Changes** tab, select the remote repo you'd like to push to, then locate your file that you have modified. Double-click the file to open a diff and verify that the changes that you have created are correct.



2. Once you've verified your changes, click the **Stage Commit (+)** button to stage the file for pushing.



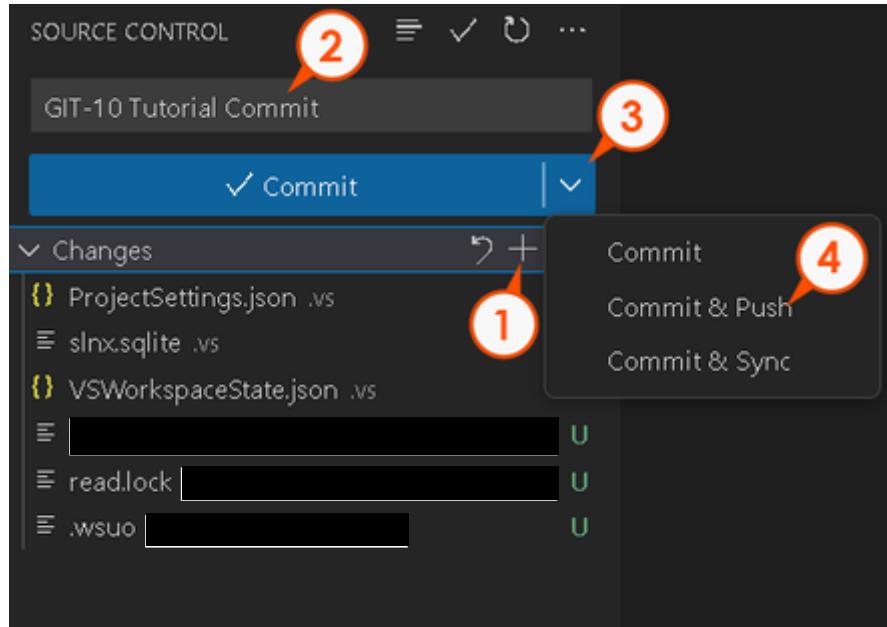
3. Enter a commit message that includes your Jira ticket number (remember that this should be the name of your branch), then click the **Push** button to push into the remote repo.



You have safely pushed into the remote repo.

#### Visual Studio Code

To safely push into the repository, complete the following procedure. It is important that you follow the steps in this section carefully:



1. Press the **Stage Commit (+)** button to stage all of your changes and resolve any conflicts before committing.
2. Add the commit. Be sure to include the Jira ticket number in the commit message, which should also be the name of your branch.
3. Click on the drop-down menu to extend further options.
4. Click **Commit & Push**.

Your code is now in the remote repo.

### 2.3 - Checking Your Code In Correctly

Whenever you're ready to check in code, it's important that you link the associated code commit to a Jira ticket. In the comment field, you'll want to describe the changes made in a short sentence; for example, you could write, "GIT-123 Change x, y, and z." If you don't check in your code with a valid Jira ticket, you'll receive an error. For more information on this behavior, see [Auditing Code Commit](#). For additional advice when writing your Git commit message, see the article <https://cbea.ms/git-commit/>.

**Your code check-ins should:**

- Have a Jira ticket referenced in the comment so that it can be linked between the commit and ticket.
- Be related to a Jira Story or a Bug.
- Relate to a Jira ticket that has been sized.

**Your code check-ins should not:**

- Be attempted without attaching a Jira ticket number.
- Be a ticket that isn't a Story or a Bug (e.g. an Epic, an Initiative, etc.).
- Be attached to a Jira ticket that hasn't been sized.

The following use cases describe scenarios for correct and incorrect check-ins:

- If the developer enters a valid Jira ticket number, and they are assigned that ticket:
  - **Step Behavior:** Step will pass, build will proceed.
  - **Step Message:** Last code check-in validated, build may proceed.
  - **Requirement:** Every commit should have a Jira ticket referenced in the comment, so that it can be linked between the commit and ticket. Note that you may use brackets when referencing your Jira ticket or not. Examples:
    - [Git-123] This is a test
    - This is a [Git-123] test
    - Git-123 This is a test

← Jobs in run #20221213.16  
test11

**Jobs**

- Job 3s
  - Initialize job <1s
  - Checkout test11@devel... 1s
  - Validate Last Commit** (highlighted with an orange arrow)
  - Post-job: Checkout te... <1s
  - Finalize Job <1s
  - Report build status <1s

**Validate Last Commit**

```

1 Starting: Validate Last Commit
2 =====
3 Task      : PowerShell
4 Description : Run a PowerShell script on Linux, macOS, or Windows
5 Version   : 2.180.1
6 Author    : Microsoft Corporation
7 Help      : https://docs.microsoft.com/azure/devops/pipelines/tasks/utility/powershell
8 =====
9 Generating script.
10 ===== Starting Command Output =====
11 /bin/sh -c curl -s https://raw.githubusercontent.com/MSOpenTech/PowerShell-Docs/main/validate-last-commit.ps1 | powershell -Command iex -EncodedCommand $(ConvertTo-SecureString -String $(Get-Content -Raw) -AsPlainText -Force)
12 Last code checkin validated, build may proceed.
13 Finishing: Validate Last Commit

```

- ✓ If the developer checks-in code, but links it to a Jira ticket they are not assigned to:
  - Step Behavior:** Step will pass, build will proceed.
  - Step Message:** The last check-in has a Jira ticket associated but is not assigned to the developer.
  - Requirement:** There is not a requirement that states that only the developer who is assigned to the Jira ticket can commit code to that Jira ticket. The build will continue, but a message will display on the build step.

← Jobs in run #20221213.12  
test11

**Jobs**

- Job 3s
  - Initialize job <1s
  - Checkout test11@devel... 1s
  - Validate Last Commit** (highlighted with an orange arrow)
  - Post-job: Checkout te... <1s
  - Finalize Job <1s
  - Report build status <1s

**Validate Last Commit**

```

1 Starting: Validate Last Commit
2 =====
3 Task      : PowerShell
4 Description : Run a PowerShell script on Linux, macOS, or Windows
5 Version   : 2.180.1
6 Author    : Microsoft Corporation
7 Help      : https://docs.microsoft.com/azure/devops/pipelines/tasks/utility/powershell
8 =====
9 Generating script.
10 ===== Starting Command Output =====
11 /bin/sh -c curl -s https://raw.githubusercontent.com/MSOpenTech/PowerShell-Docs/main/validate-last-commit.ps1 | powershell -Command iex -EncodedCommand $(ConvertTo-SecureString -String $(Get-Content -Raw) -AsPlainText -Force)
12 The last checkin has a Jira ticket associated but is not assigned to the developer.
13 Finishing: Validate Last Commit

```

- ✗ If the developer checks-in code, but does not link it to a Jira ticket:
  - Step Behavior:** Step will fail and build will not proceed.
  - Step Message:** The last check-in does not have a valid Jira ticket within the comment!
  - Requirement:** Every commit needs to be associated to a Jira ticket.
  - To Remediate:** Go through the check-in process again, but enter a valid Jira ticket in the comment.

```

1 Starting: Validate Last Commit
2 -----
3 Task      : PowerShell
4 Description : Run a PowerShell script on Linux, macOS, or Windows
5 Version   : 2.180.1
6 Author    : Microsoft Corporation
7 Help      : https://docs.microsoft.com/azure/devops/pipelines/tasks/utility/powershell
8 -----
9 Generating script.
10 -----
11 -----
12 The last checkin does not have a valid Jira ticket within the comment!
13 ##[error]PowerShell exited with code '1'.
14 Finishing: Validate Last Commit

```

- ✗ If the developer checks-in code, but enters a Jira ticket number that is not either a Story or Bug:
  - **Step Behavior:** Step will fail and build will not proceed.
  - **Step Message:** The last check-in is not associated to a valid Story or Bug.
  - **Requirement:** Cannot commit to a Jira ticket type that is not a Story or Bug (i.e. Epic, Initiative, etc.)
  - **To Remediate:** Go through the commit process again, but enter a valid Jira ticket in the comment.

```

1 Starting: Validate Last Commit
2 -----
3 Task      : PowerShell
4 Description : Run a PowerShell script on Linux, macOS, or Windows
5 Version   : 2.180.1
6 Author    : Microsoft Corporation
7 Help      : https://docs.microsoft.com/azure/devops/pipelines/tasks/utility/powershell
8 -----
9 Generating script.
10 -----
11 -----
12 The last checkin is not associated to a valid Story or Bug.
13 ##[error]PowerShell exited with code '1'.
14 Finishing: Validate Last Commit

```

- ✗ The developer tries to commit to a Story or Bug that is not sized.
  - **Step Behavior:** Step will fail and build will not proceed.
  - **Step Message:** Last check-in is associated to a ticket with no story points.
  - **Requirement:** In order for developers to commit a check-in to a Jira ticket, the Story or Bug must be sized. This requirement is currently not implemented, but the code is in place.
  - **To Remediate:** Reach out to your team lead or PO and ask them to populate the Story Points field.

The screenshot shows the Azure DevOps Pipeline interface. On the left, a sidebar titled 'Jobs in run #20221213.19' lists several steps: 'Job' (3s), 'Initialize job' (<1s), 'Checkout test11@devel...' (1s), 'Validate Last Commit' (highlighted with a red arrow), 'Post-job: Checkout te...' (<1s), 'Finalize Job' (<1s), and 'Report build status' (<1s). The main area is titled 'Validate Last Commit'. The log output is as follows:

```

1 Starting: Validate Last Commit
2 =====
3 Task      : PowerShell
4 Description : Run a PowerShell script on Linux, macOS, or Windows
5 Version   : 2.180.1
6 Author    : Microsoft Corporation
7 Help      : https://docs.microsoft.com/azure/devops/pipelines/tasks/utility/powershell
8 =====
9 Generating script.
10 ====== Starting Command Output ======
11
12 Last checkin is associated to a ticket with no story points.
13 ##[error]PowerShell exited with code '1'.
14 Finishing: Validate Last Commit

```

A red box highlights the error message 'Last checkin is associated to a ticket with no story points.' and the exit code '1'.

## 2.4 - Code Reviews and Merging Your Code

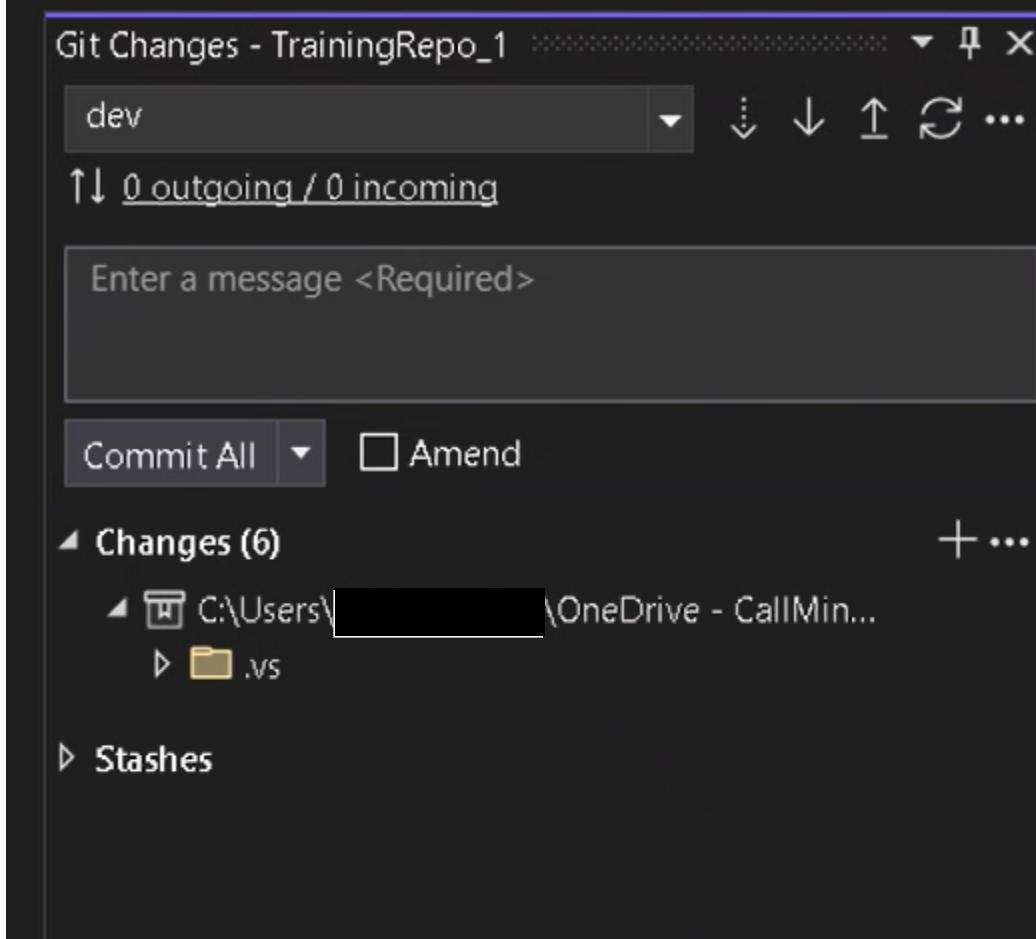
The steps to perform a code review and to merge your code into dev are located in the expandable boxes below.

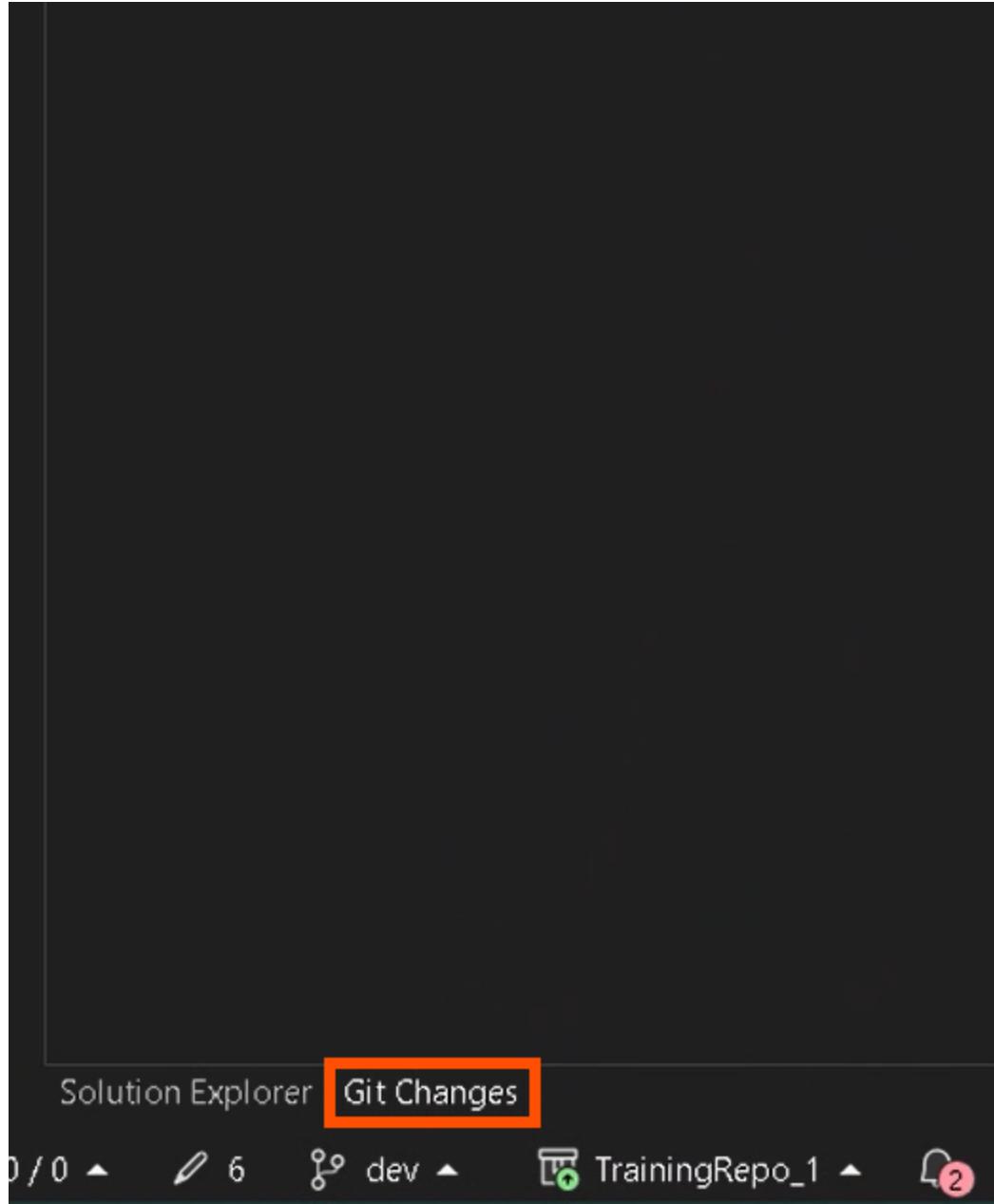
**Note:** Section 1a covers the procedure for code changes and creating a new branch using Visual Studio, while Section 1b covers the procedure using Visual Studio Code. The procedure is identical beginning with Section 2.

### Section 1a: Code Changes and Creating a New Branch in Visual Studio

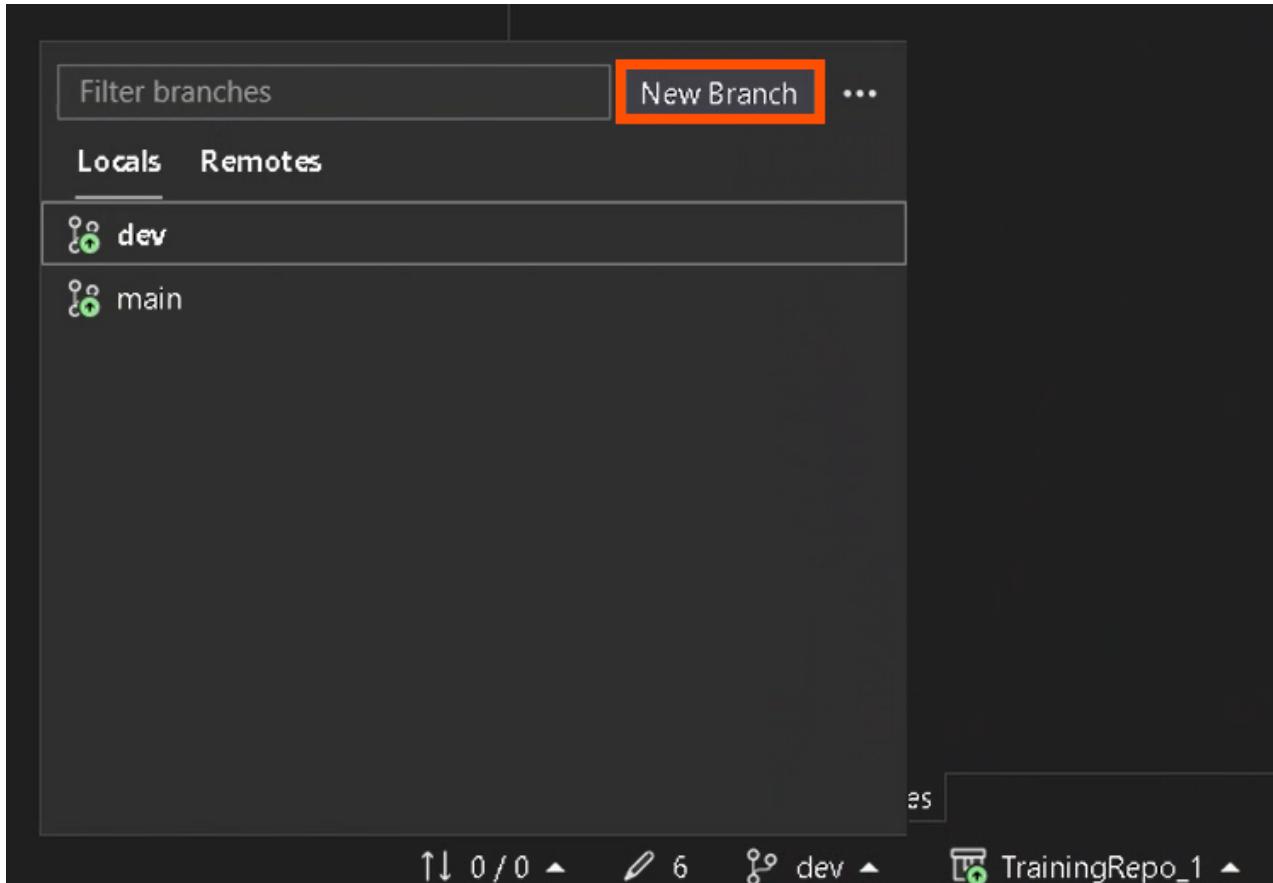
In the following steps, you'll create another branch from the dev branch so that you can store the code you'd like to have reviewed.

1. In Visual Studio, click the **Git Changes** tab in the Team Explorer window.

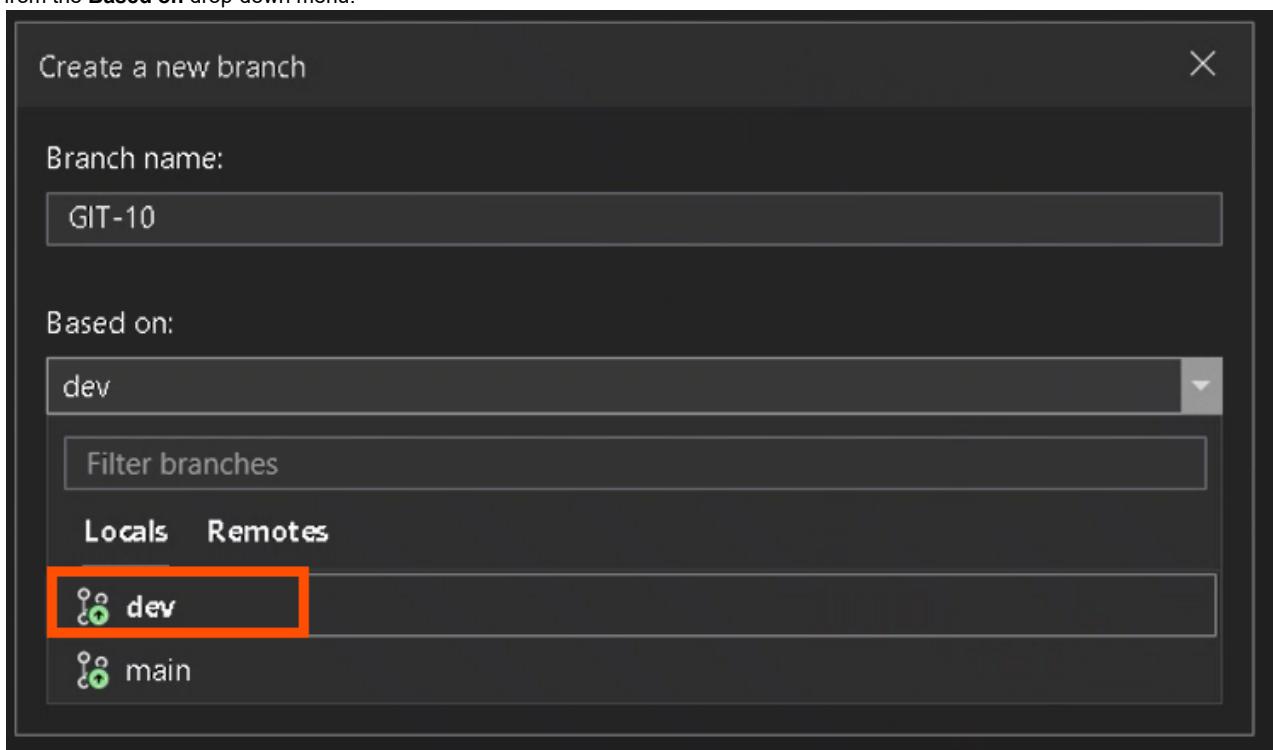




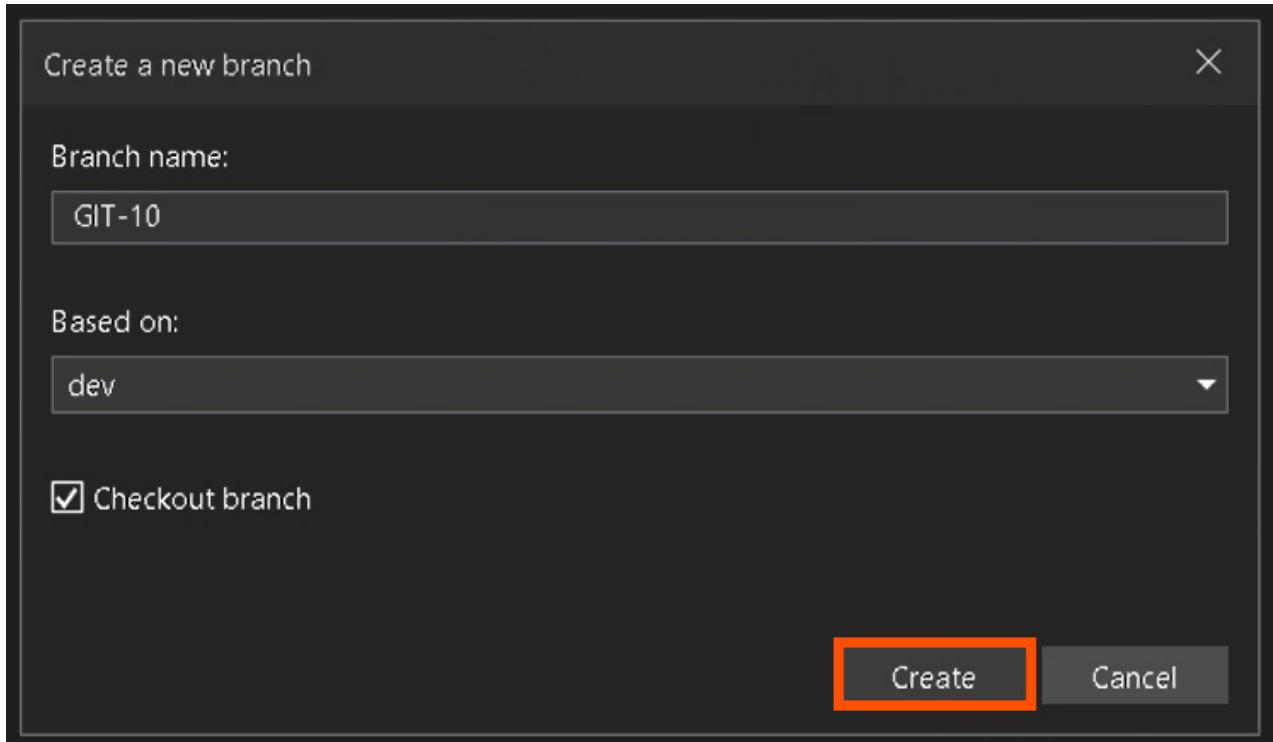
2. Click **New Branch**.



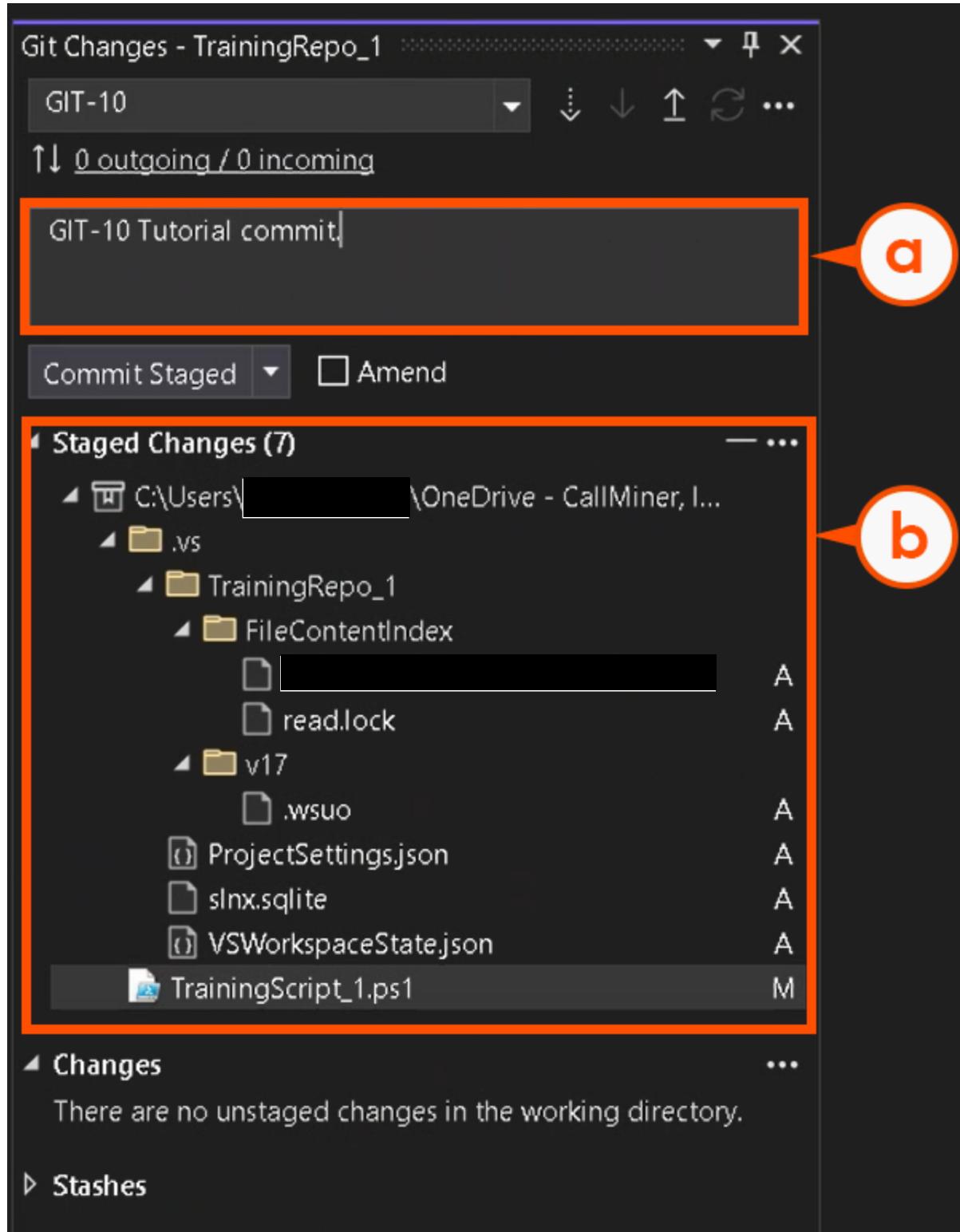
3. Enter your new branch name (be sure to follow the [Git -- Branch Naming Conventions](#)) in the **Branch name** field, then select **dev** from the **Based on** drop-down menu.



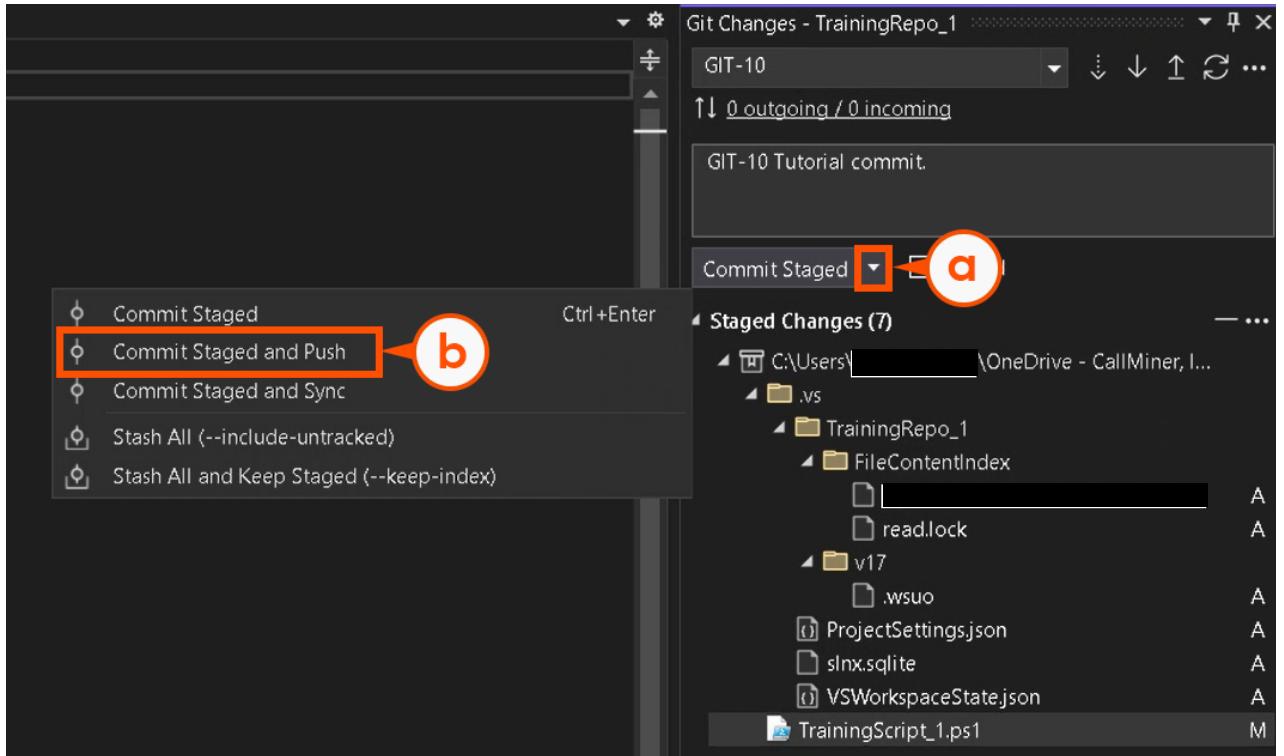
4. Make sure the **Checkout branch** box is checked, then click **Create** to finish creating your branch.



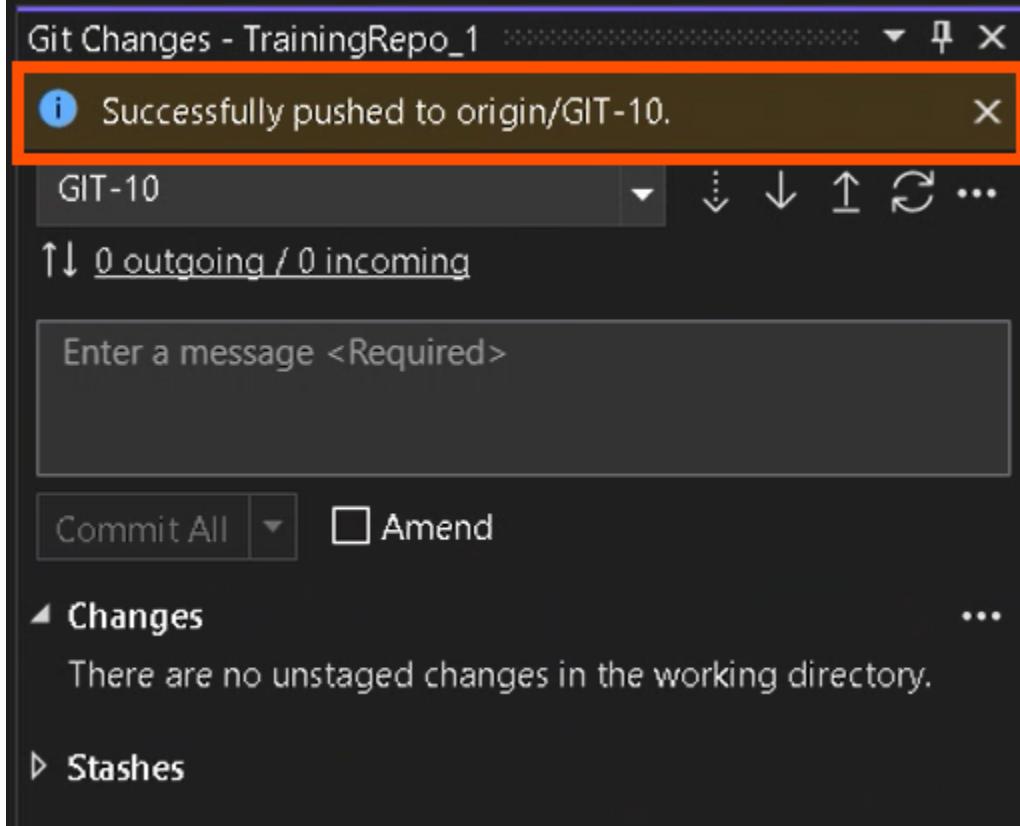
5. Modify your code as you normally would in Visual Studio. When you're ready to begin the check-in process, return to the Git Changes tab and stage your change.
6. Enter your comment in the comment field (a). **It is VERY important that you include the Jira ticket you are associating with this code commit.** You can also verify your staged changes before you begin to commit your code under **Staged Changes** (b).



7. To commit your code, press the arrow next to **Commit Staged** (a), then click **Commit Staged and Push** (b) in the drop-down menu.



8. When finished, you'll see a message indicating that you have successfully pushed your modified code to your branch.

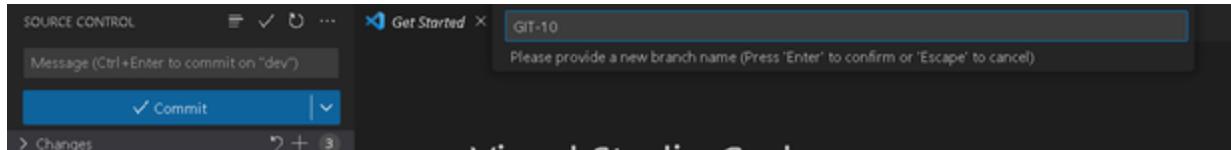


You have completed creating your own branch in Visual Studio.

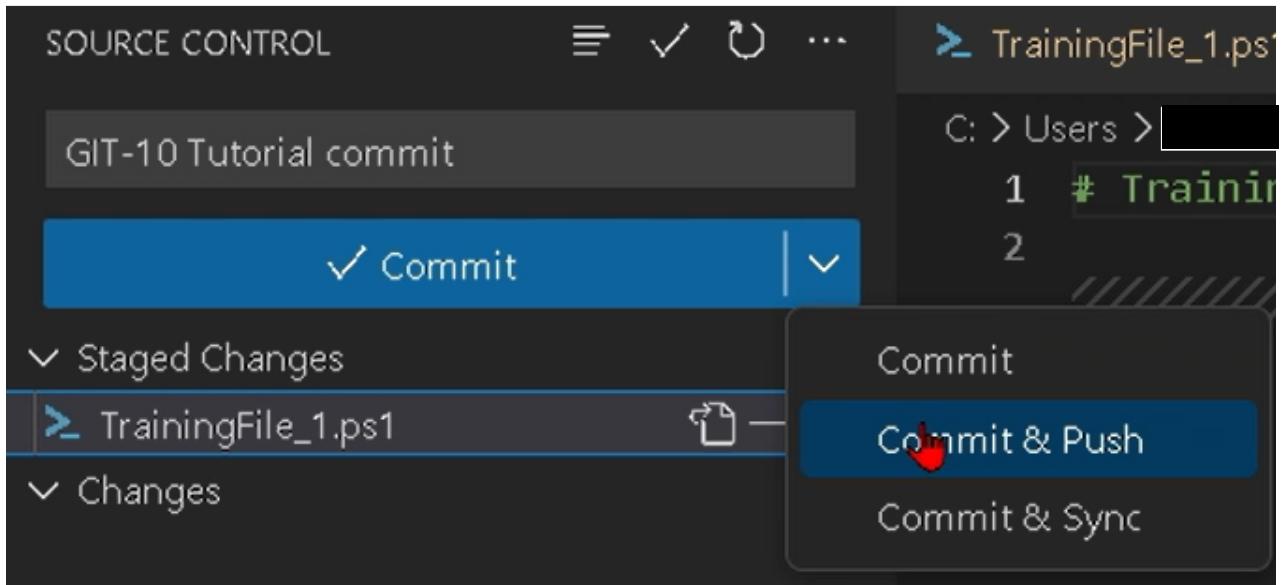
#### Section 1b: Code Changes and Creating a New Branch in Visual Studio Code

In the following steps, you'll create another branch from the dev branch so that you can store the code you'd like to have reviewed.

1. To avoid potentially corrupting another developer's code, create a new branch name from the dev branch. For the purposes of this document, our new branch is called GIT-10.



2. After you've made modifications to your code in this branch, save your changes.
3. Verify that you have the correct code by comparing it between the previous version of your code and your current, un-reviewed version.
4. Fill in the comment describing your code change. Be sure to follow the standards described in the "Checking Your Code In Correctly" section.
5. Click **Commit & Push**.



You have completed creating your own branch in Visual Studio Code. You can now move on to the next section in the process.

## Section 2: Creating a Pull Request

In this section, you'll create a pull request using the branch you created in Section 1a or Section 1b. Follow the steps below:

1. In Azure DevOps, navigate to your recently pushed code in the branch you created. Click **Create a pull request**.

A screenshot of the Azure DevOps repository page for 'TrainingScript\_1.ps1'. The page shows a commit message: 'You updated \$ GIT-10 Just now'. On the right, there's a 'Create a pull request' button, which is highlighted with a red box. The code editor shows the following script:

```
1 # Training script 001
2 #
3 #
4 # I added this code
5 #
6 #
7 # New code addition+++
```

You can also create a new pull request by clicking **Pull Requests** in the **Repos** part of your dashboard, then clicking one of the **New pull request** buttons.

2. In the **New pull request** page, configure the following settings:

Engineering / trainingProject / Repos / Pull requests / TrainingRepo\_1 ▾

## New pull request

**a.**  into 

Overview Files 7 Commits 2

### Title

**b.** 

### Description

 Add commit messages

**c.** 

① Markdown supported. Drag & drop, paste, or select files to insert.

① Link work items.



GIT-10 Training merge.

### Optional reviewers



### Required reviewers



**d.** 

- Your source should be your newly created branch, while your target should be dev.
  - Your title should include the name of the Jira ticket associated with your code commit.
  - Your description should include the name of the Jira ticket associated with your code commit.
  - Your required reviewers should be an individual or team that will review your code.
3. When you've finished configuring your pull request, click **Create**.

Optional reviewers

Required reviewers

Tyler Watson
X

Work items to link

Tags

Create

4. When you come to the **Pull Request** page, you have multiple options for interacting with the request.

The screenshot shows a pull request titled "Merge GIT-10 into DEV". The top navigation bar includes "Engineering / trainingProject / Repos / Pull requests / TrainingRepo\_1". The main area has tabs for "Overview", "Files", "Updates", and "Commits". A callout "a" points to the "Active" status indicator. Callouts "b", "c", and "d" point to the "Files", "Updates", and "Commits" tabs respectively. Callout "e" points to a circular badge indicating "Tyler Watson must approve". Below the tabs, a green checkmark says "No merge conflicts Last checked 4m ago". To the right, there are sections for "Reviewers" (with "Required" and "Optional" tabs) and "Tags". The "Reviewers" section shows "Tyler Watson" with a note "No review yet". The "Tags" section shows "No tags".

The options are as follow:

- a. **Overview:** Provides an overview of the pull request.
- b. **Files:** Contains all of the code files that have been modified and committed. By clicking on **Files**, you will be able to view all of them and review the changes made. Additionally, the review team that you have assigned can add comments on each line of code.
- c. **Updates:** Lists all updates done to the code file(s).
- d. **Commits:** Displays a list of all commits that were made as part of that pull request. It will show the details of the commits such as the author, date, and commit message. It also allows to view the changes that were made in each commit and also allows to navigate to a specific file that was changed.
- e. **The Reviewer:** Shows the name of the person or people who need to review your code before it can be merged.
- f. **The List of Reviewers:** Shows the reviewer(s) who have been requested to review your code, but includes both required and optional reviewers.

You have completed making a new pull request.

#### Section 3: Review and Approve the Code Changes

In these steps, you'll learn about receiving code review changes and how to approve the review. While the previous sections describe the steps from the perspective of a developer committing code changes, this section describes the steps of a review from the perspective of a reviewer. When working with your teams, you'll assume both roles. Follow these steps to learn how to review and approve code changes:

1. You, as the reviewer, will receive an email notifying you that you've been requested to review a pull request made by another dev. Click **View pull request** in the notification email.

Merge GIT-10 into DEV

**created a new pull request**

 [git-10 into dev](#)

GIT-10 changes merging into Dev

[View pull request](#)

## Reviewers

[Tyler Watson](#)

Required

## Files 1

 [/TrainingScript\\_1.ps1](#)

## Commits 1

GIT-10 test 100

[b7e6e75e](#) • [REDACTED] • Thursday, January 19, 2023 10:40 AM

2. To view the requested piece of code, click **Files** in the **Pull Request** page, then comment with any feedback you may have about the code.

Engineering / trainingProject / Repos / Pull requests /  TrainingRepo\_1

Merge GIT-10 into DEV

Active | 162  git-10 into dev

Approve

Overview Files Updates Commits

All Changes Filter 1 changed file

TrainingRepo\_1

TrainingScript\_1.ps1

\* Looks good.

TrainingScript\_1.ps1

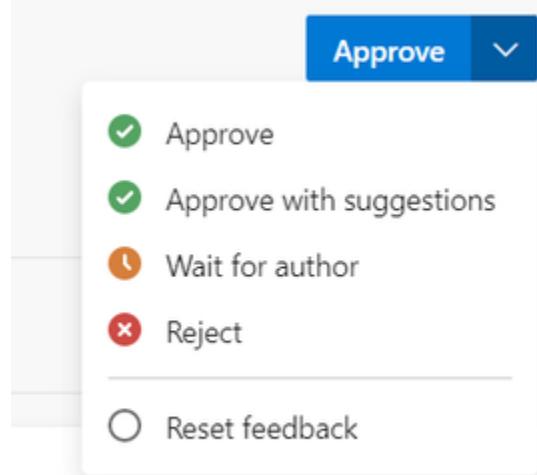
# Training script 001  
#  
- # xxxx  
# I added this code

Looks good.

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

Cancel Comment

- Click the arrow next to the **Approve** button. From the drop-down menu, you can **Approve**, **Approve with suggestions**, **Wait for author**, or **Reject** the proposed request. Should you choose anything other than **Approve**, the dev who submitted the pull request will need to make changes to their code based on your feedback and resubmit the pull request.



- Once you see no problems with the code, click **Approve** to approve the code.
  - Once the pull request has been approved, you'll see a green check-mark next to your Microsoft Office icon. The dev can then resume the merge procedure.

into main 0/1 comments resolved

Approve Complete ↻

File

Inline ↻ ↻

TrainingFile\_1.ps1 +2  
/TrainingFile\_1.ps1

View

1 # Training trial 1 - Tyler  
2 +  
3 + # sgfeveeveviewewewewewe\_cdeedcaedcadec

Tyler Watson Just now Active ↻

Looks good.

Resolve

Write a reply...

You have now completed reviewing the code and can move on to the next step in the process.

## ▼ Section 4: Completing the Merge

In these steps, you'll learn the process of completing the merge of your code in to the dev branch. Unlike Section 3, the steps below describe the process from the perspective of the developer making changes to the code, returning to the perspective from Sections 1a/1b and Section 2. Follow the steps below to complete the merge:

1. Now that your code has been approved, your **Pull Request** page will tell you that your reviewer(s) has approved your code. You can now begin the process of merging your code into dev by clicking the **Complete** button.

The screenshot shows a pull request page for 'Merge GIT-10 into DEV'. At the top, there's a navigation bar with 'Engineering / trainingProject / Repos / Pull requests / TrainingRepo\_1'. Below it, a search bar and a 'Complete' button are highlighted with a red box. The main area shows '1 reviewer approved' and 'No merge conflicts'. A 'Description' section contains the text 'GIT-10 changes merging into Dev'. On the right, there are sections for 'Reviewers' (one required reviewer named Tyler Watson), 'Optional' (no optional reviewers), 'Tags' (no tags), and 'Work items' (no work items). A comment history shows a message from 'EA' and an approval from 'Tyler Watson' just now. A file named 'TrainingScript\_1.ps1' was added 3m ago.

2. After clicking the **Complete** button, a **Complete pull request** window will appear. Here, you can choose your **Merge type** and **Post-completion options**. Select *Merge (no fast forward)* as your merge type. Choose the following post-completion options:
  - a. Complete associated work items after merging
  - b. Delete git-10 after merging. Remember that *git-10* is the name of our example branch. Whatever you named your branch will be listed in the option when you perform this procedure.
  - c. Do NOT check **Customize merge commit message**.

# Complete pull request

X

## Merge type

Merge (no fast forward)



## Post-completion options

- Complete associated work items after merging
- Delete git-10 after merging
- Customize merge commit message

Cancel

Complete merge

i. You also have the option to select *Squash Commit* as a merge type option. See the [Git FAQ](#) for more info.

3. When you've configured your merge options, click **Complete merge**. This will complete your merge and return you to the **Pull Request** page, which confirms that you've completed the pull request and your code has been merged in to dev.

Merge GIT-10 into DEV

Completed 162 git-10 into dev 0/1 comments resolved

Overview Files Updates Commits

Cherry-pick Revert

Reviewers

Required

Tyler Watson Approved

Optional

No optional reviewers

Tags

No tags

Work items

No work items

You have successfully merged your code in to dev! 😊

#### 2.4.1 Merging into main

After a successful merge of your code into the `dev` branch, the next step is to incorporate it into the `main` branch; however, this is beyond the scope of the developer's tasks, and thus should only be performed by a team lead. Nothing should ever be pushed directly into this branch. Upon accidental push into `main`, please speak to your team lead immediately.

In order to achieve this, the following prerequisites must be met:

1. Completion of regression testing to ensure the code's readiness.
2. Conclusion of all Quality Assurance (QA) testing.
3. Scheduling of the merge to `main` branch in coordination with Release Candidate (RC) testing.
4. Execution of the merge to `main` branch by a designated team lead.

## 3.0 - Resources

This section will provide resources that can be used to help you answer some of your questions or to help fix any issues that may arise during development.

Resource Name	Resource Link
CallMiner's Git FAQ	<a href="#">Git FAQ</a>
Prevent Pushing to Master on Github	<a href="https://stackoverflow.com/questions/46146491/preventpushing-to-master-on-github">https://stackoverflow.com/questions/46146491/preventpushing-to-master-on-github</a>
How to Manage Git Branches & Merge Conflicts with VS Code	<a href="https://www.youtube.com/watch?v=O7cQcdP53bA">https://www.youtube.com/watch?v=O7cQcdP53bA</a>
Git Branching Strategies vs. Trunk-Based Development	<a href="https://launchdarkly.com/blog/git-branching-strategies-vs-trunk-based-development/">https://launchdarkly.com/blog/git-branching-strategies-vs-trunk-based-development/</a>
How to Write a Git Commit Message	<a href="https://cbea.ms/git-commit/">https://cbea.ms/git-commit/</a>
How to Turn on Dark Mode in Azure DevOps	<a href="https://medium.com/@calloncampbell/azure-devops-rolls-out-a-dark-theme-1e1310a7bd30#:~:text=To%20switch%20to%20Dark%20theme,switch%20to%20the%20dark%20theme.">https://medium.com/@calloncampbell/azure-devops-rolls-out-a-dark-theme-1e1310a7bd30#:~:text=To%20switch%20to%20Dark%20theme,switch%20to%20the%20dark%20theme.</a>

### 3.1 - Team Resources

The following table lists RM DEV team members that you can contact if you have any questions.

Role	Name	Email
Lead Developer	John Doe	john.doe@example.com
Frontend Developer	Sarah Johnson	sarah.johnson@example.com
Backend Developer	Michael Chen	michael.chen@example.com
Database Administrator	David Lee	david.lee@example.com
QA Tester	Emily White	emily.white@example.com
Project Manager	Robert Green	robert.green@example.com
Customer Support	Anna Blue	anna.blue@example.com
UX Designer	Christopher Grey	christopher.grey@example.com
Machine Learning Specialist	Olivia Black	olivia.black@example.com
Cloud Architect	Matthew Red	matthew.red@example.com