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

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.

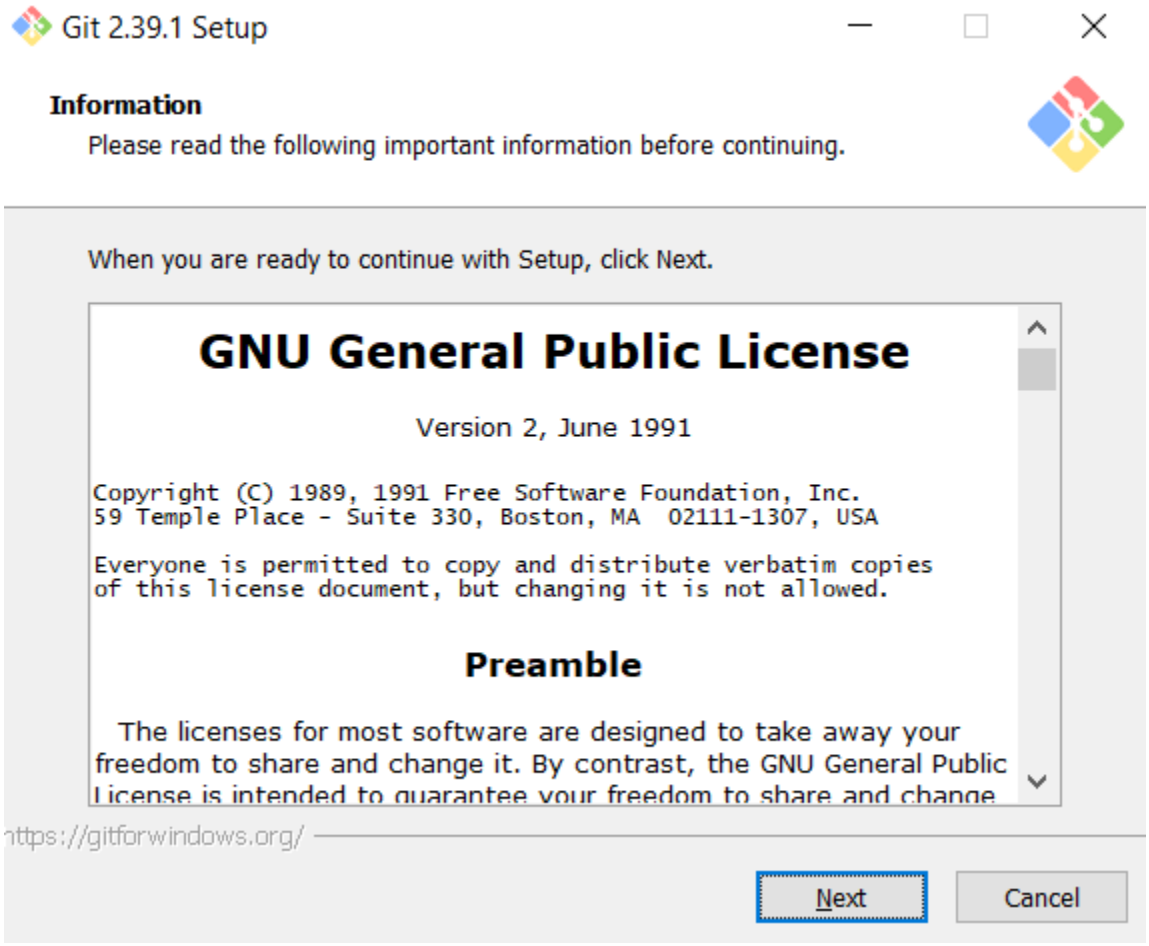
1.1 - Install

Follow the steps in this section to install Git properly:

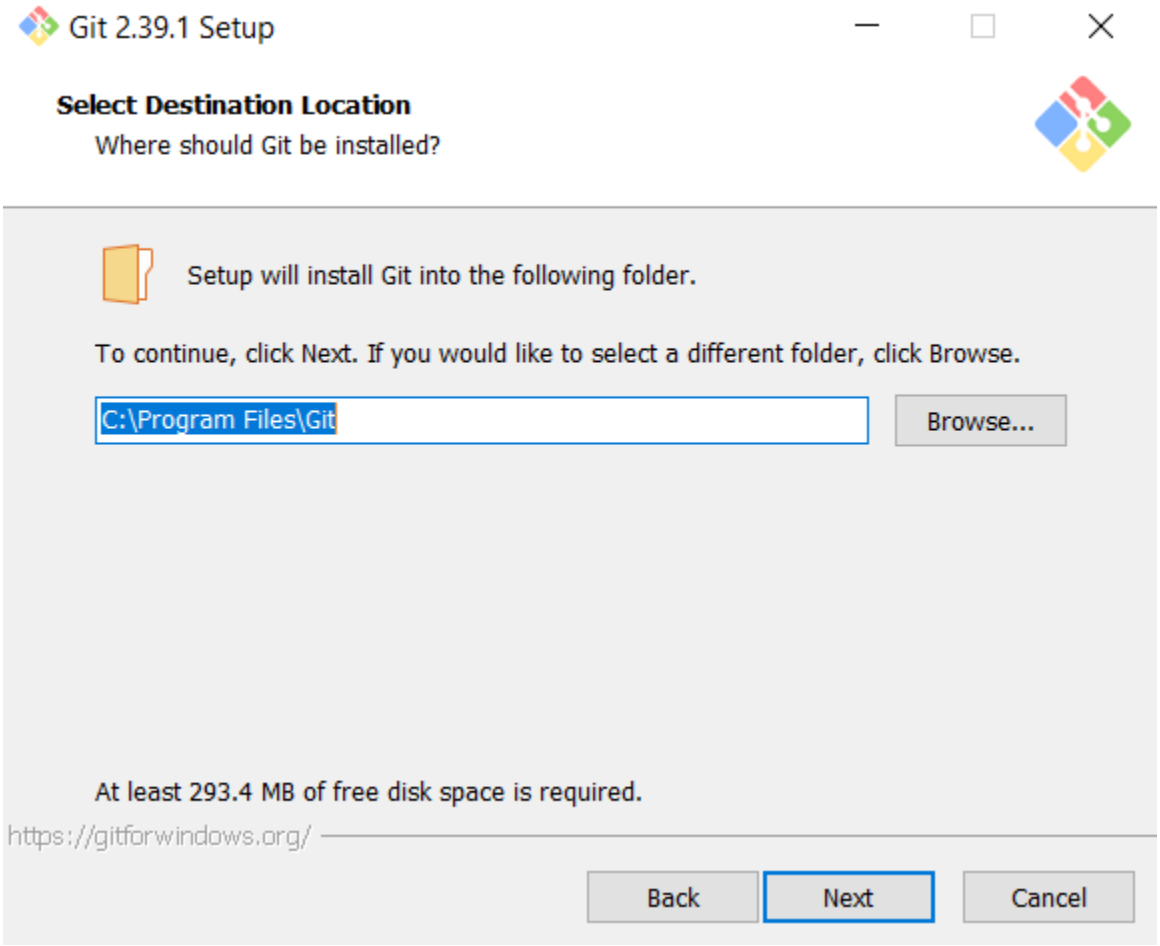
1. Link: https://git-scm.com/downloads
   a. Make sure to download the proper version for your operating system. 🚨 **DO NOT DOWNLOAD THE PORTABLE VERSION** 🚨 .
2. Expand the box below to learn how to run the setup wizard for Git:

<details>
<summary>Git Installation Wizard Steps and Options</summary>

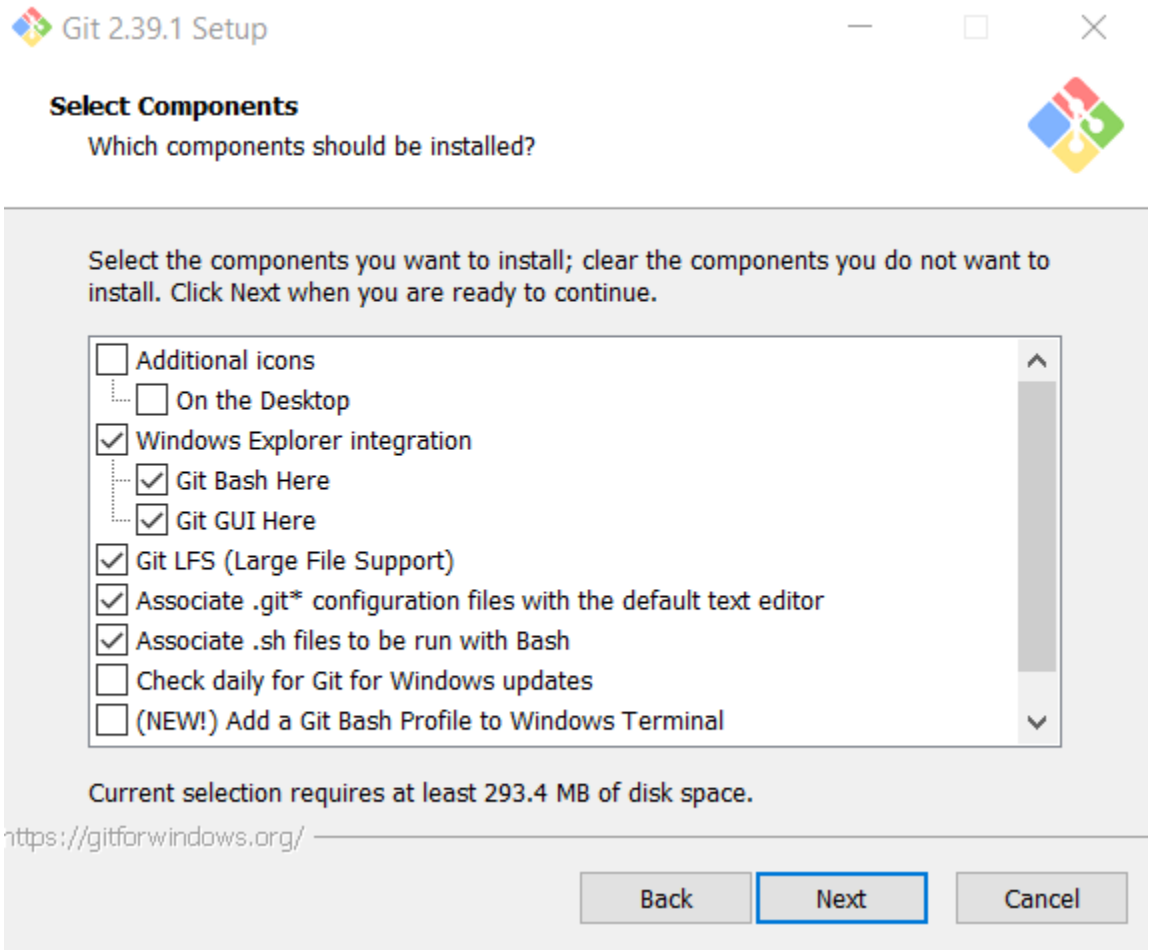
1. Click **Next** in the initial **Information** screen.
</details>



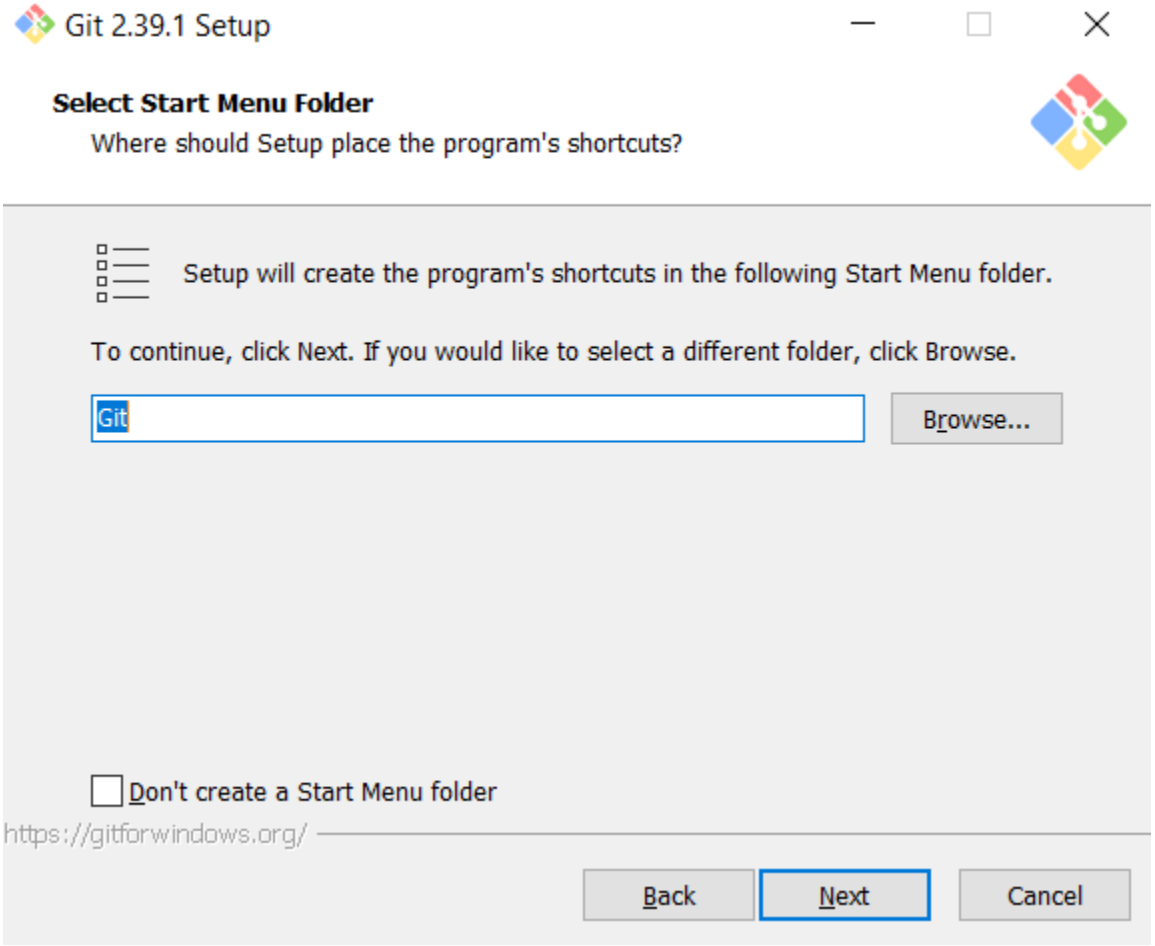
2. In the Select Destination Location screen, leave the default location in place and click **Next**.



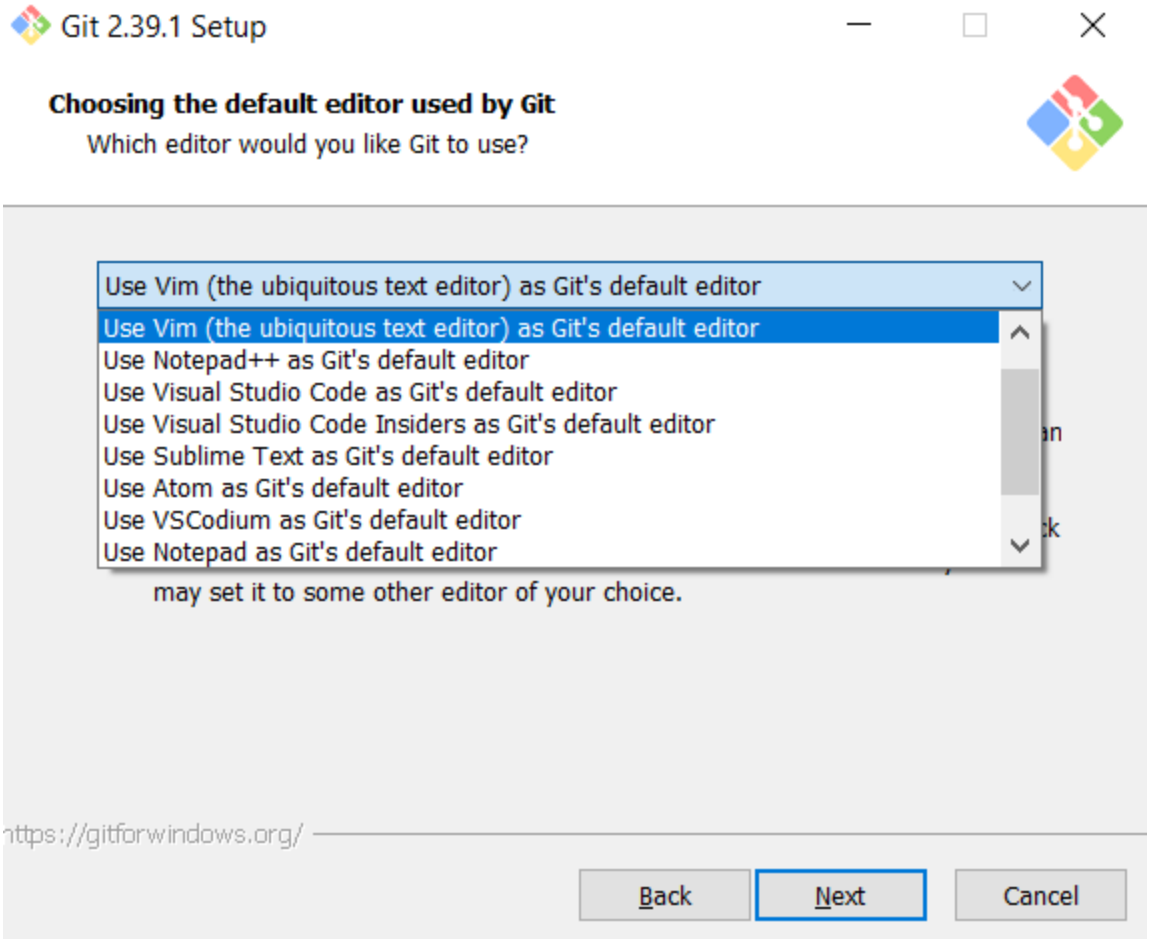
3. In the Select Components screen, leave the default options in place and click **Next**.



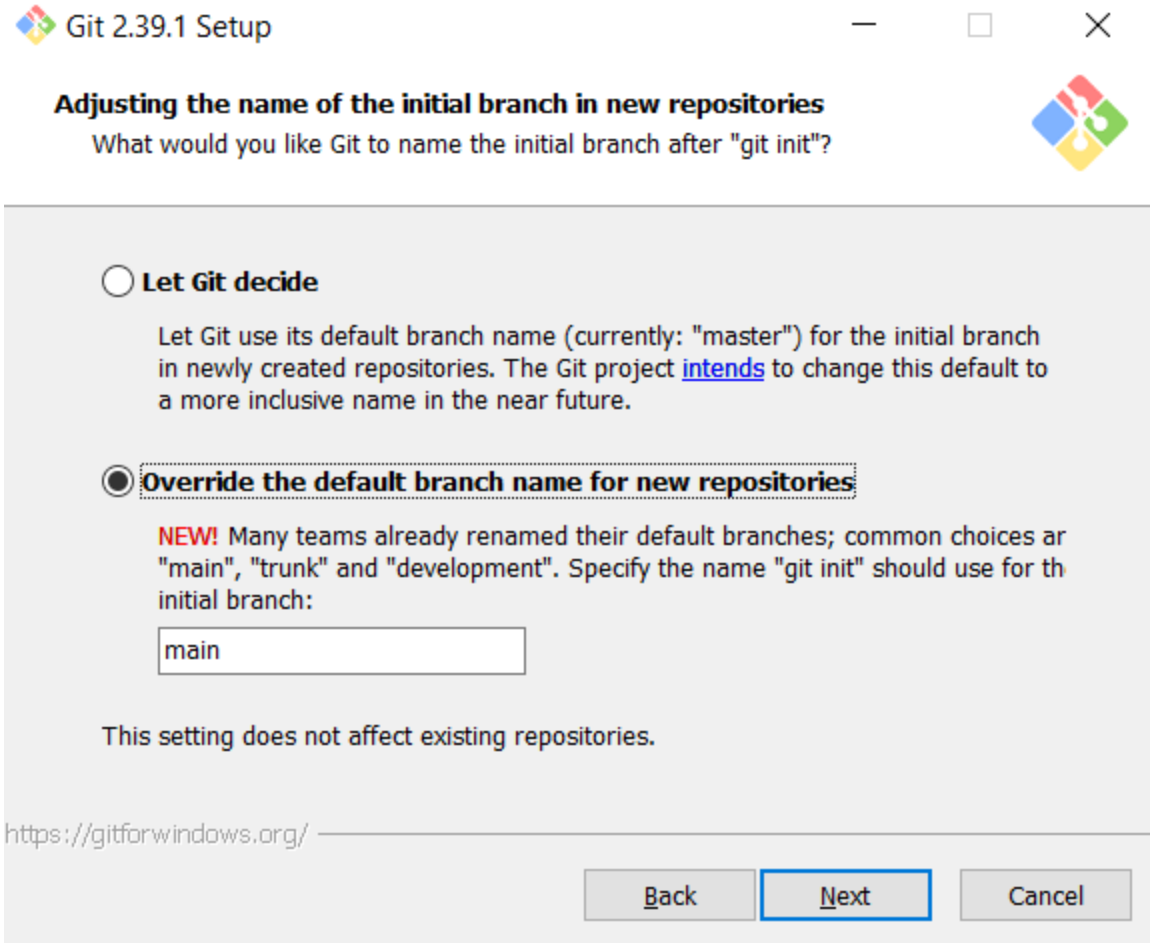
4. In the Select Start Menu Folder, leave the default folder name in place and click **Next**.



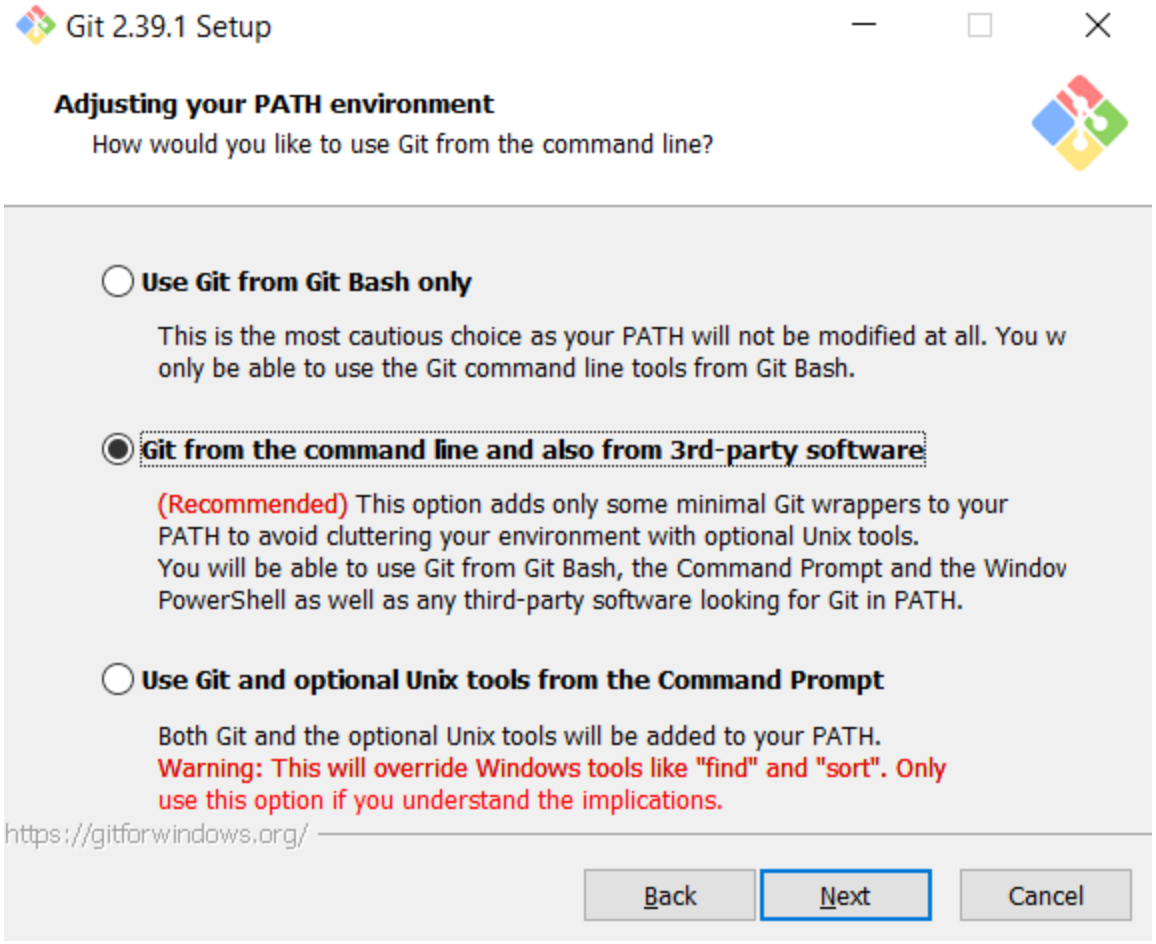
5. In the Choosing the Default Editor Used by Git screen, choose your favorite editor such as Vim, Notepad++, or Visual Studio Code, then click **Next**.



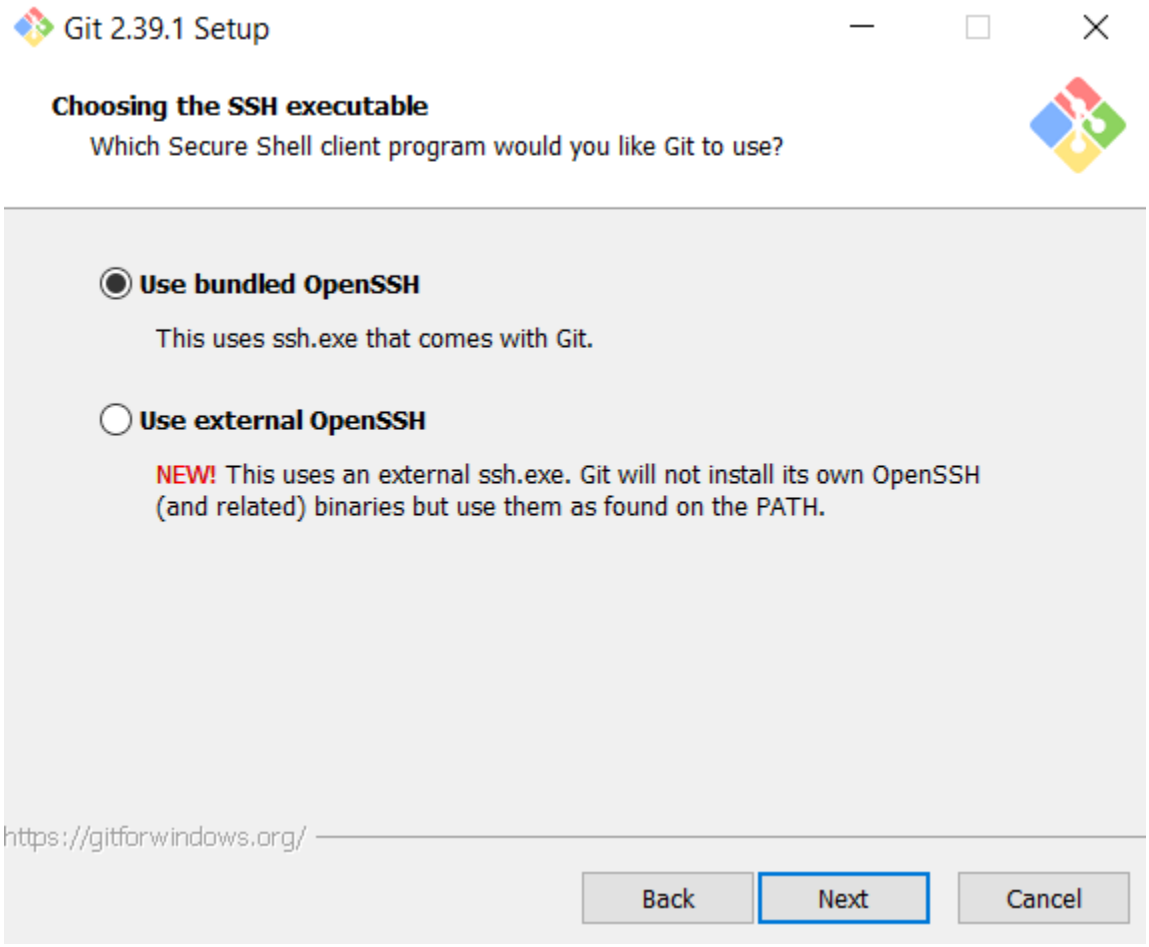
6. In the Adjusting the Name of the Initial Branch in New Repositories screen, select the **Override the default branch name for new repositories** option and enter `main` as the name of the initial branch. Click **Next**.



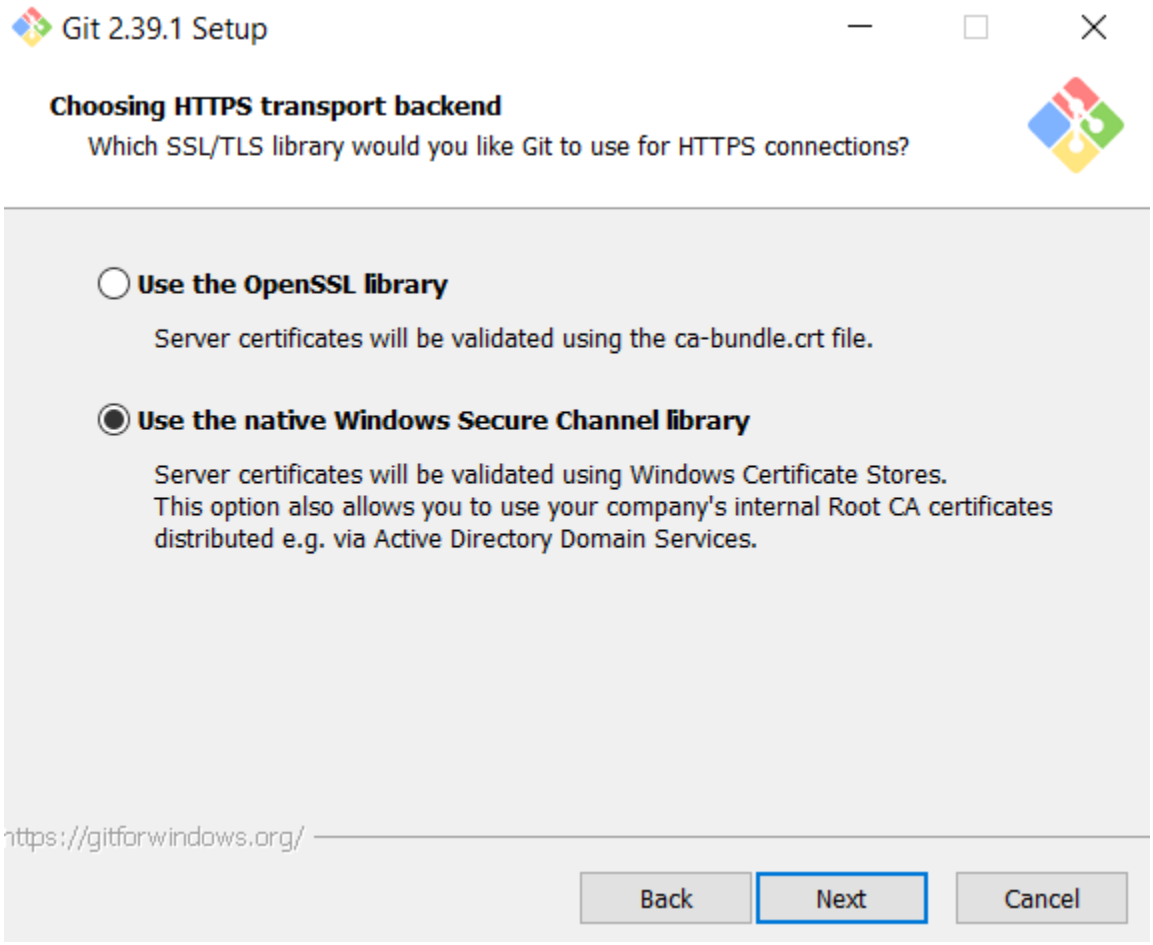
7. In the Adjusting Your PATH Environment screen, leave the **Git from the command line and also from 3rd-party software** default option in place, then click **Next.**



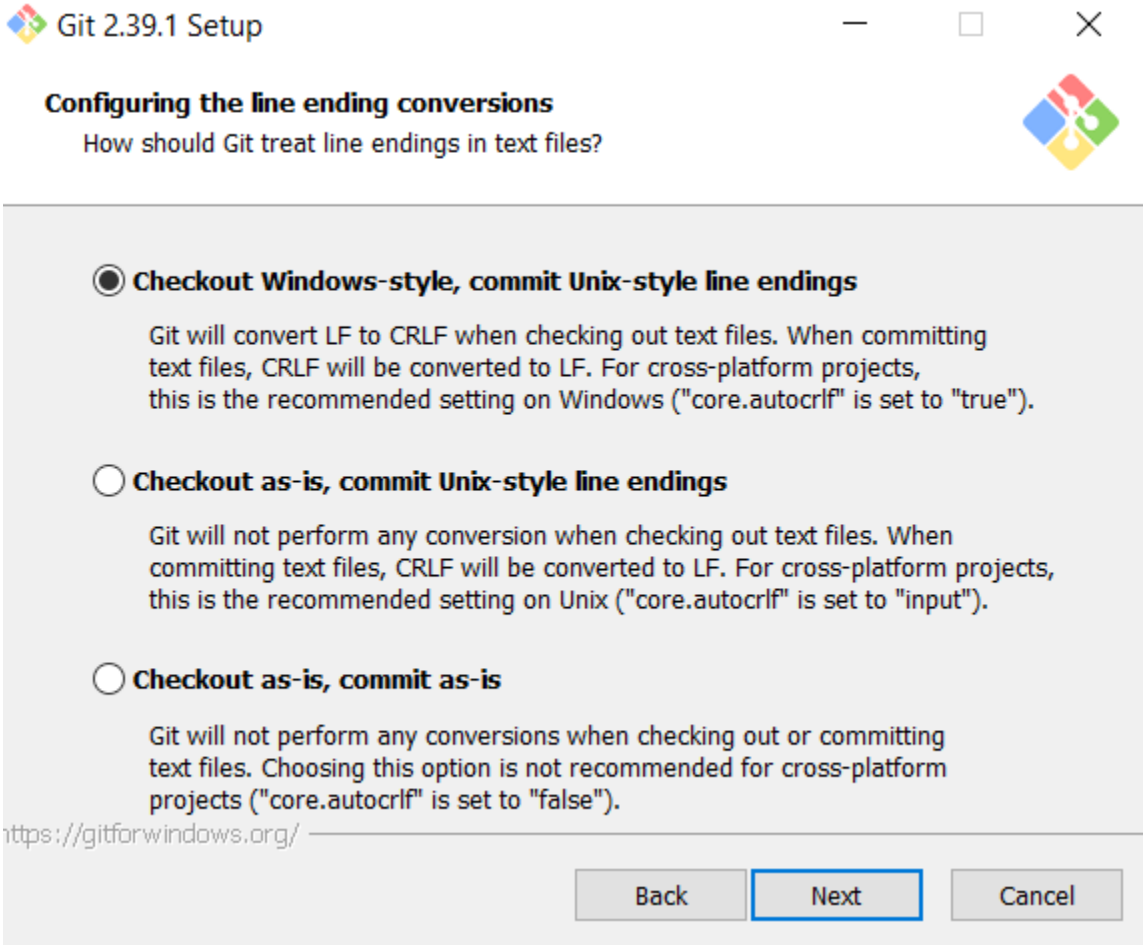
8. In the Choosing the SSH Executable screen, leave the **Use bundled OpenSSH** default option in place, then click **Next**.



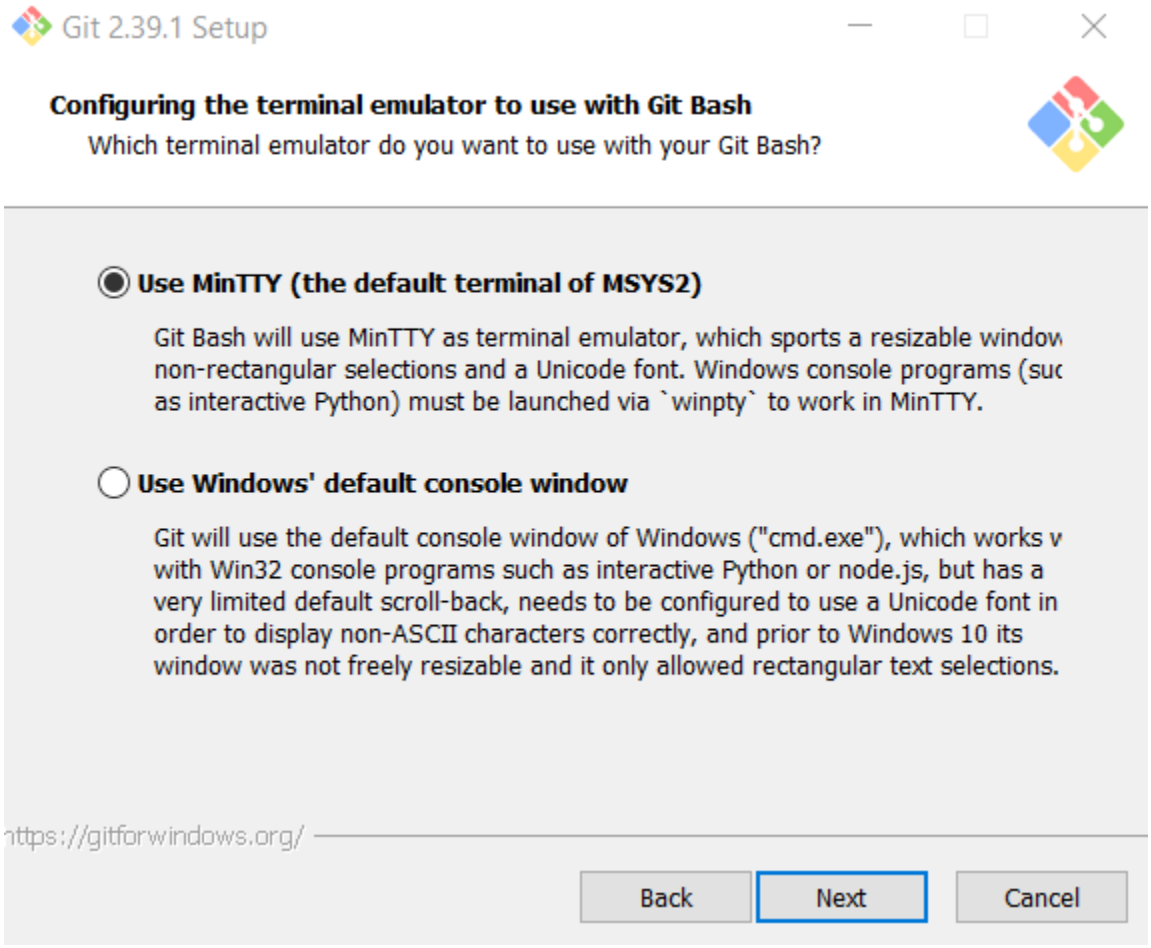
9. In the Choosing HTTPS Transport Backend screen, select the **Use the native Windows Secure Channel Library** option, then click **N ext**.



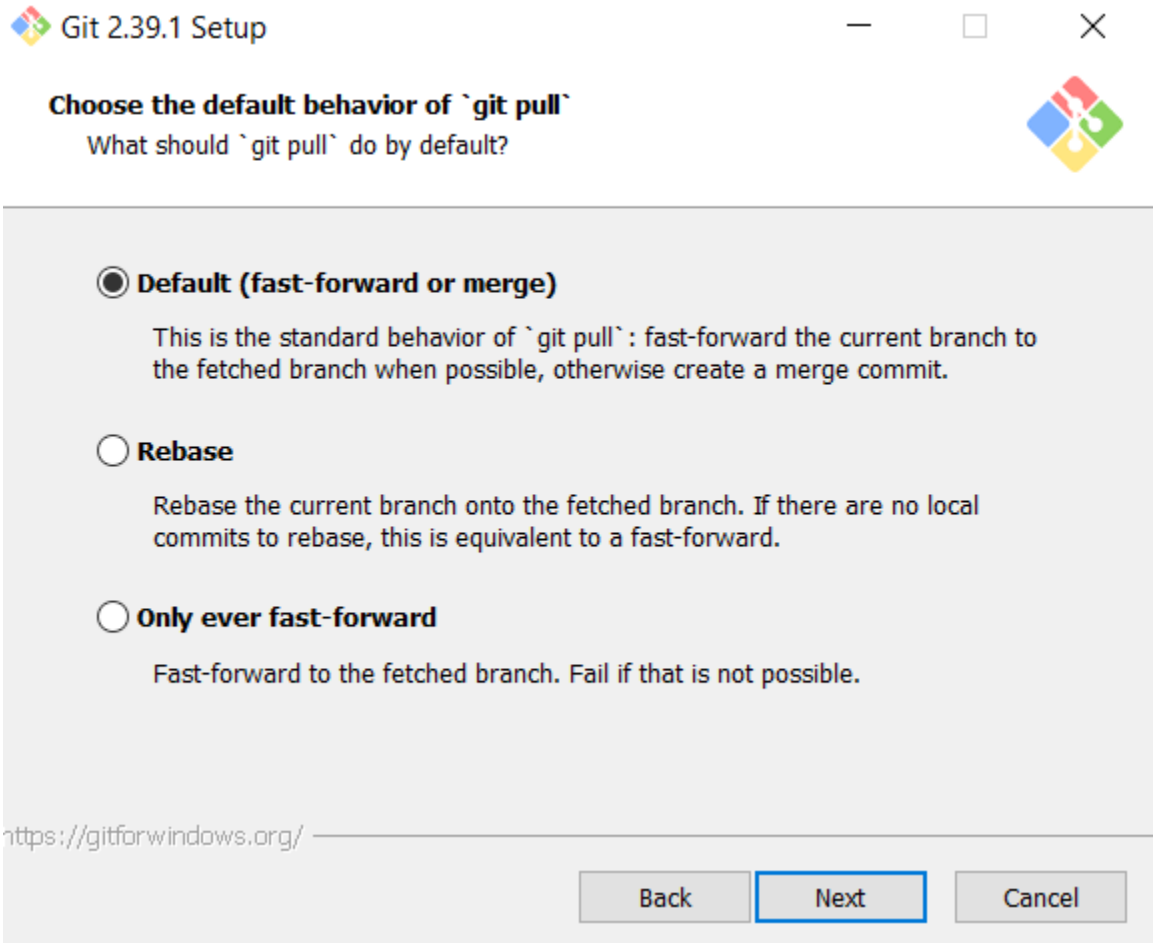
10. In the Configuring the Line Ending Conversions screen, leave the **Checkout Windows-style, commit Unix- style line endings** default option in place, then click **Next**.



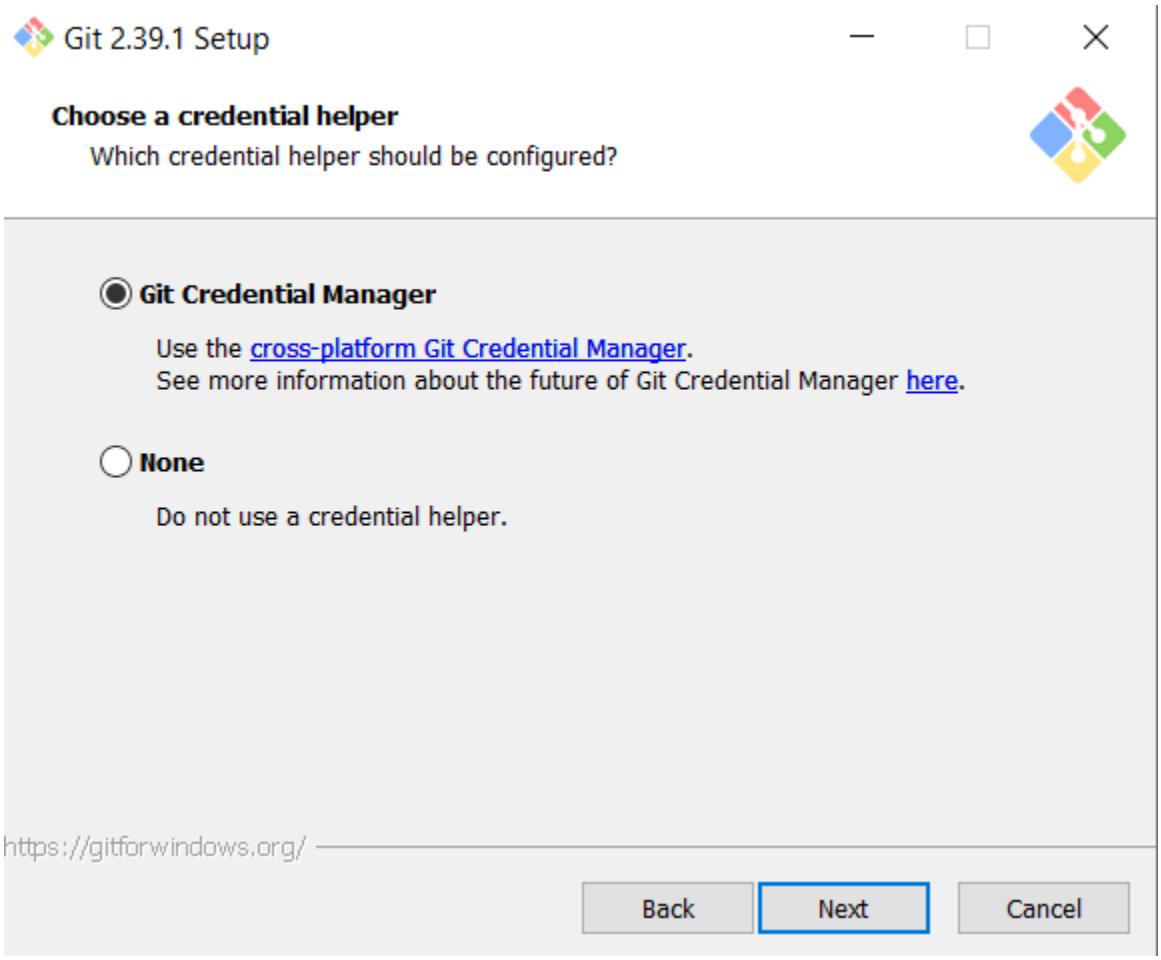
11. In the Configuring the terminal emulator use with Git Bash screen, leave the **Use MinTTY (the default terminal of MSYS2)** default option in place, then click **Next**.



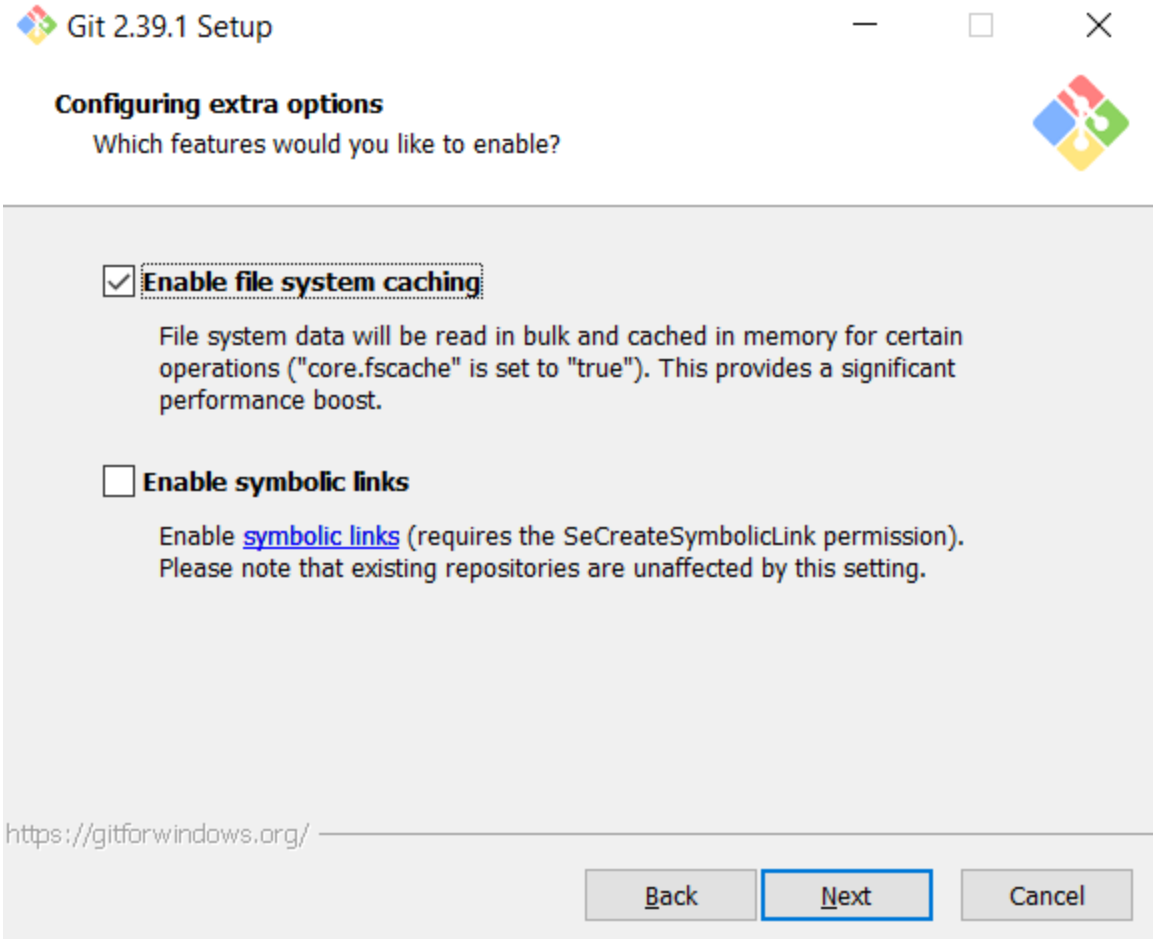
12. In the Choose the Default Behavior of 'Git Pull' screen, leave the **Default (fast-forward or merge)** default option in place and click **Ne xt**.



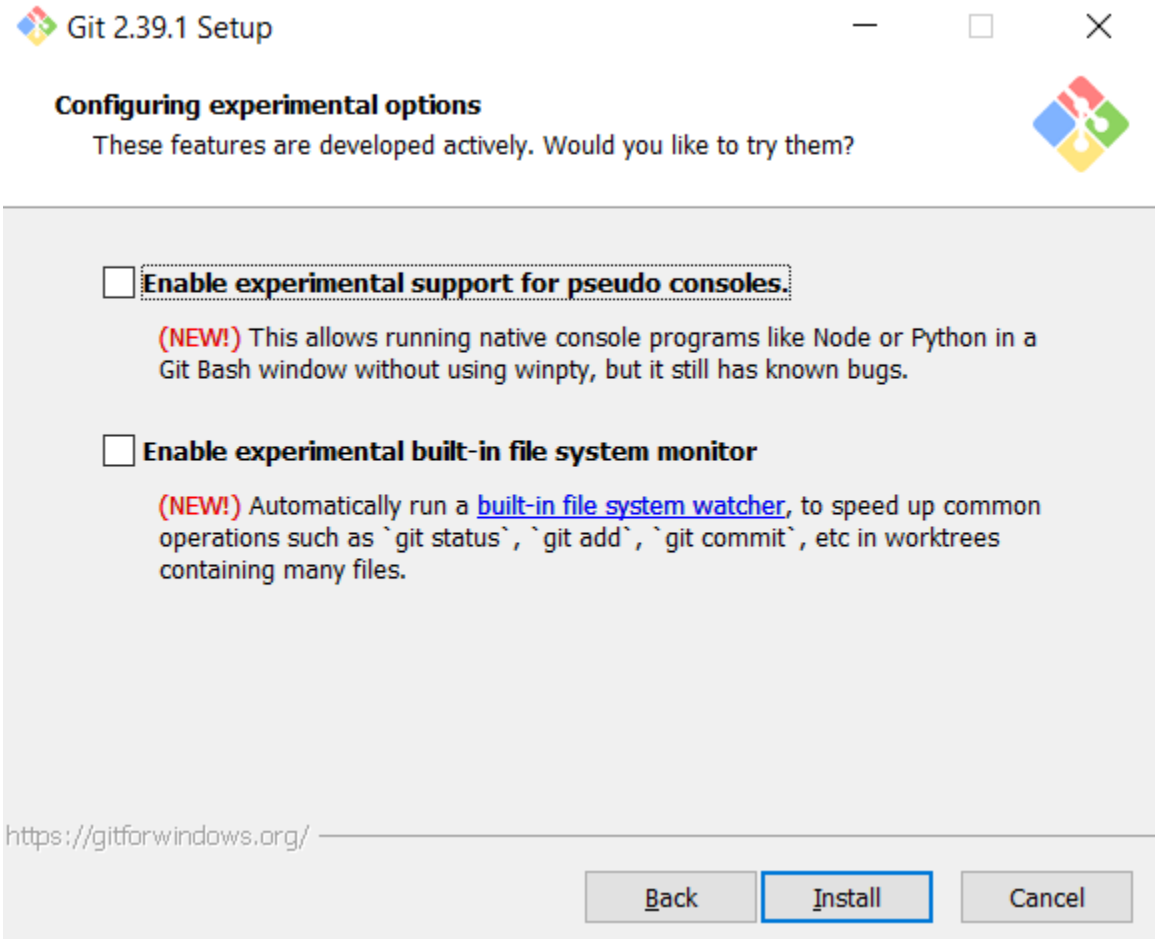
13. In the Choose a Credential Helper screen, leave the **Git Credential Manager** default option in place and click **Next**.



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



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



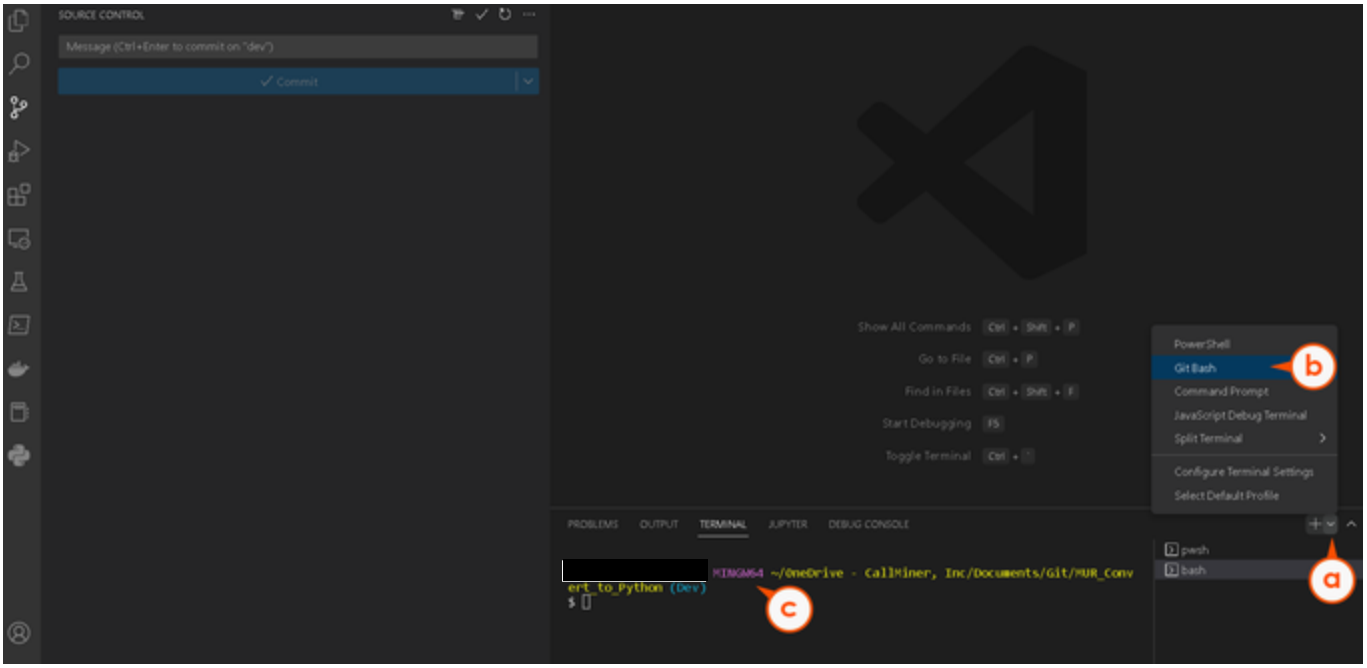
You have completed installing Git.

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.



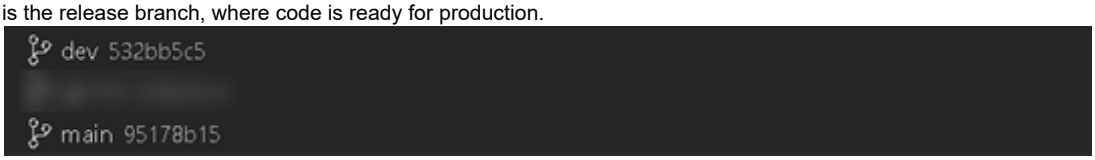
2.0 - Using Git

- 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/ .

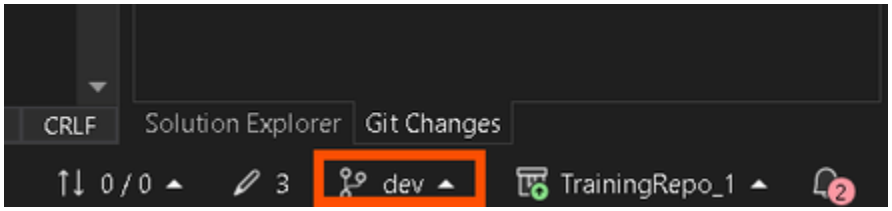
2.1 - Repository Organization

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

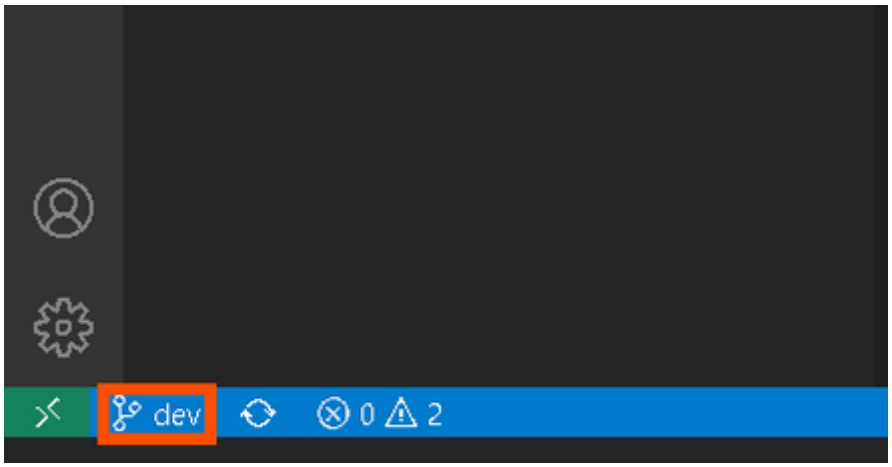
1. `dev` **(or your version of the** `dev` **branch)**
   a. This is the branch where all development work is merged into. It's also used to build the dev and QA environments.
2. `main` **(or your version of trunk)**
   a. This is the release branch, where code is ready for production.



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

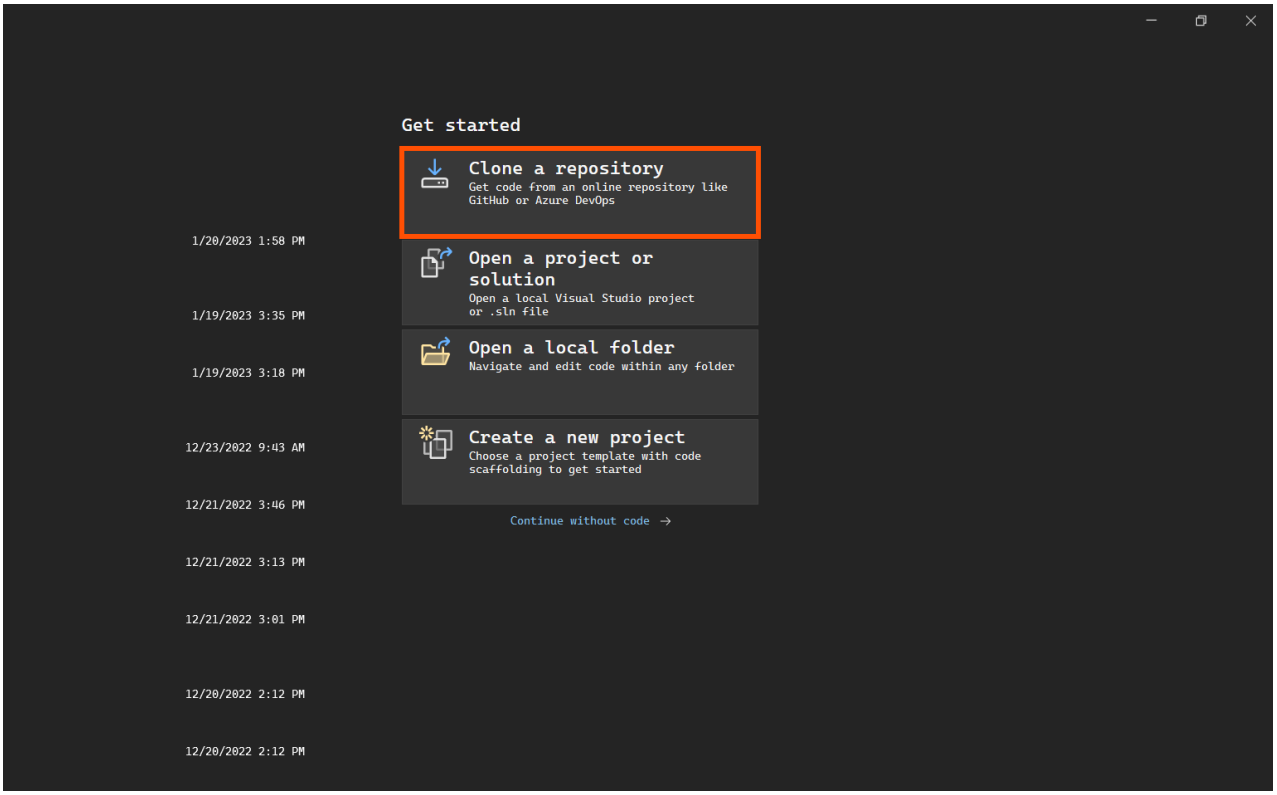
2.2.1 - Cloning and Pulling a Repository

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.

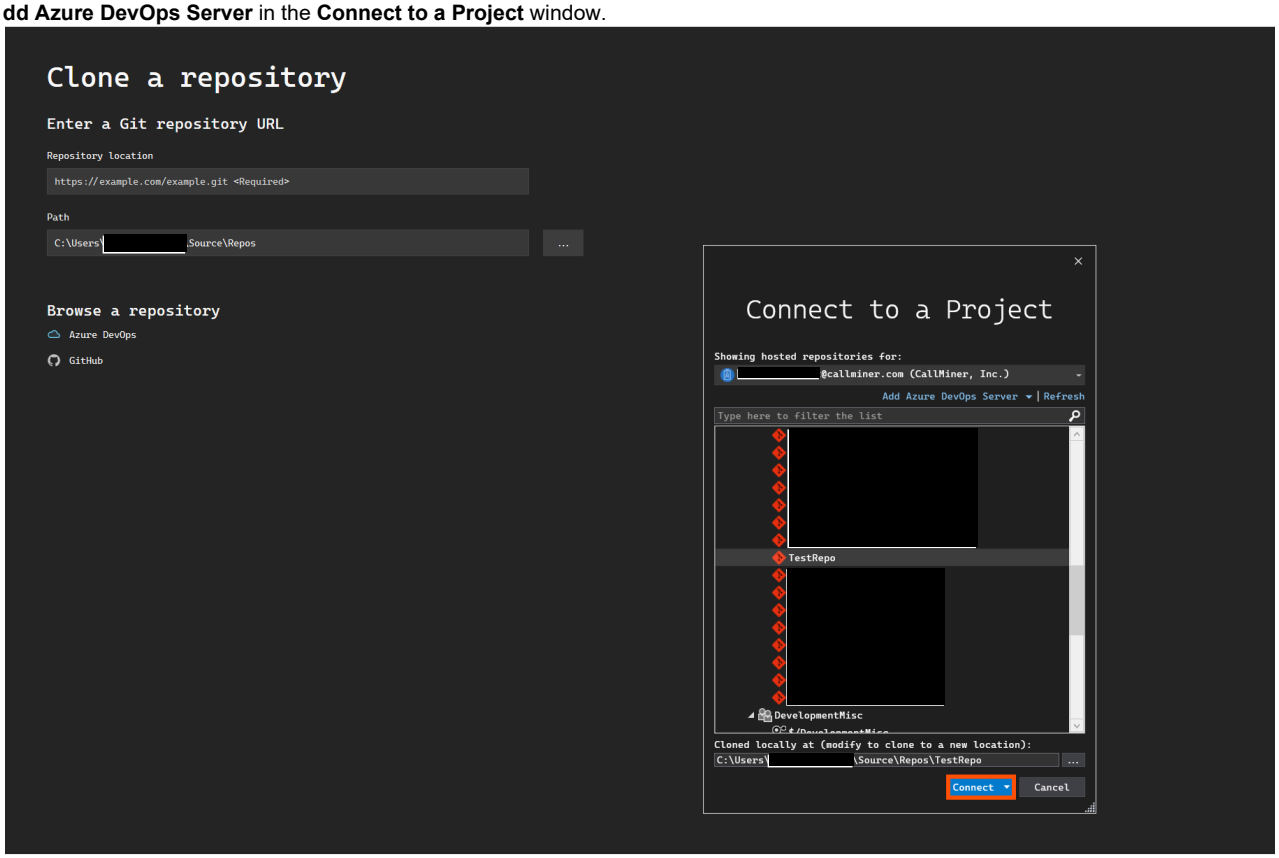
> ⚠ **Important**: When pulling into your working branch, you will see a warning telling you that line ending changes do not match. To get rid of the warning, you need to commit and push changes. **There are no actual errors occurring**; the auto-line-ending configuration developed by RM will fix the 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**.

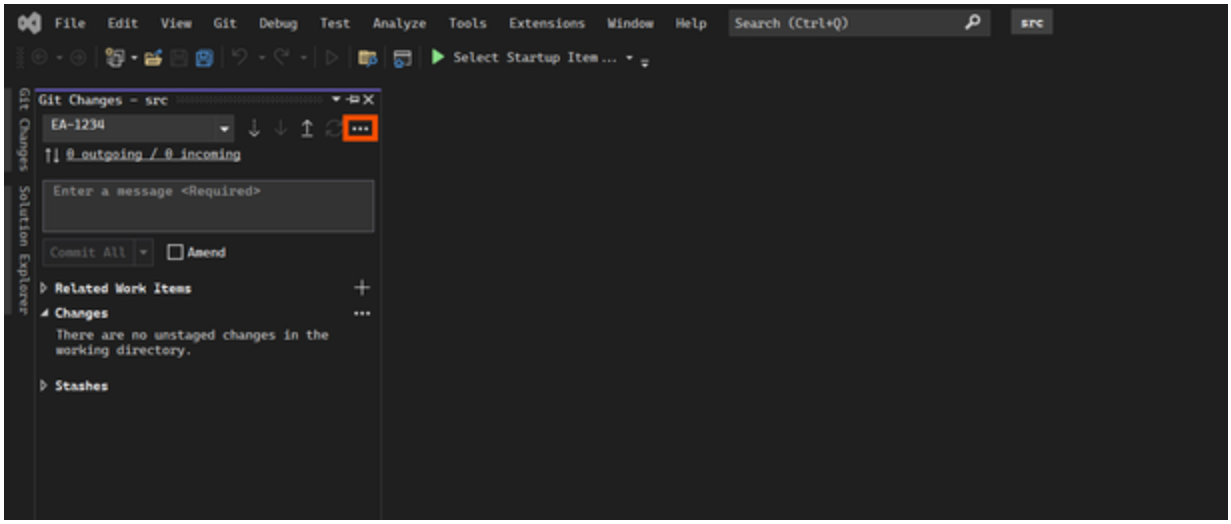


3. Choose a project to connect to. Enter the location of the subfolder you created in step 1 in the **Cloned locally at** field, then click **Connect**. If you're not connected to the Azure DevOps server, enter the URL for the server in the **Repository location** field or by clicking **Add Azure DevOps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

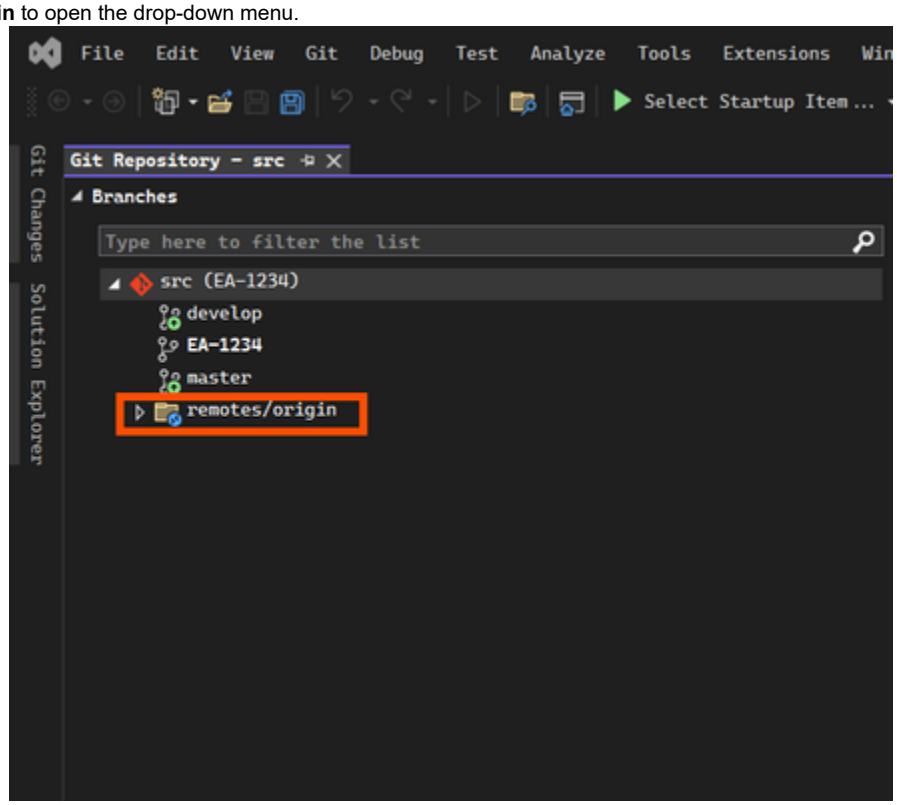


Once you clone a repo, you have the latest version of whatever exists in that repo; however, later you'll need to pull latest from `dev` to make sure that everything you have in your working branch is current.

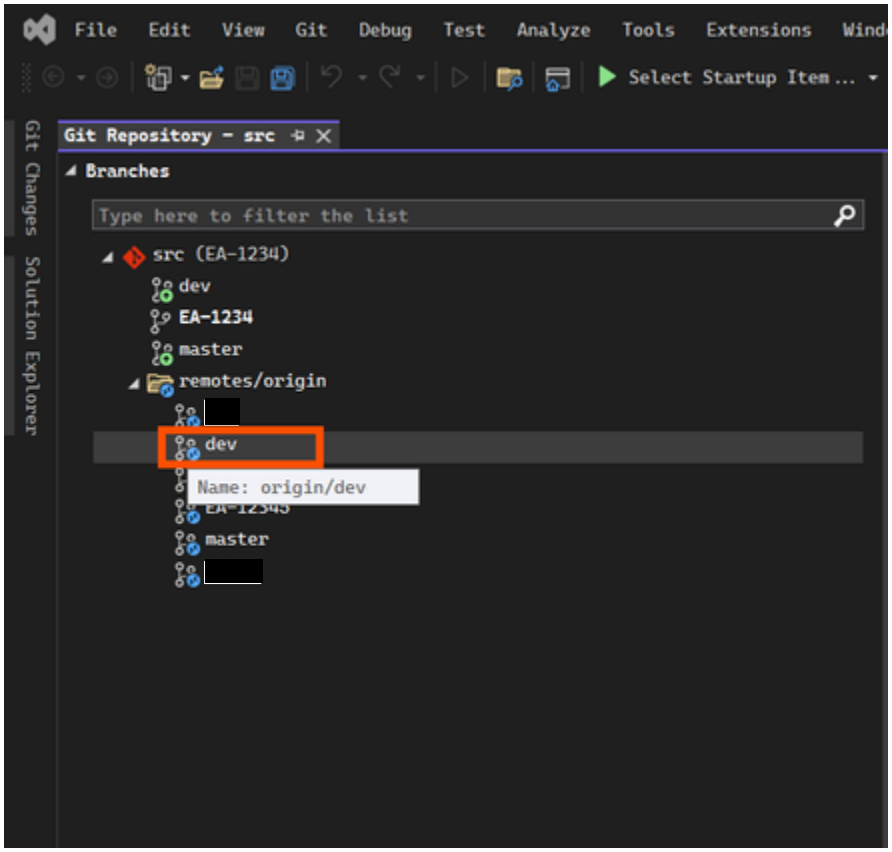
1. Click the dot menu and choose **Manage Branches**.



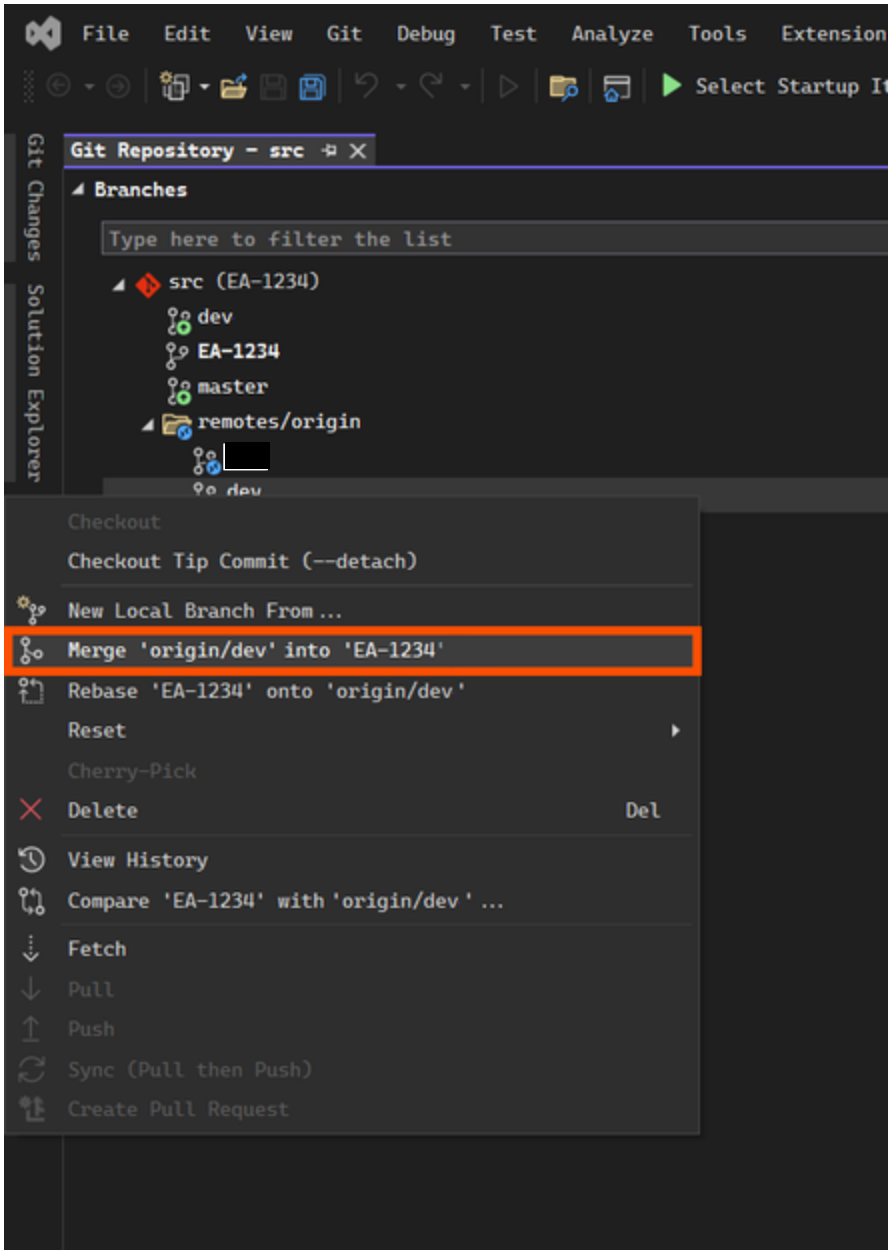
2. Click **remotes/origin** to open the drop-down menu.



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.

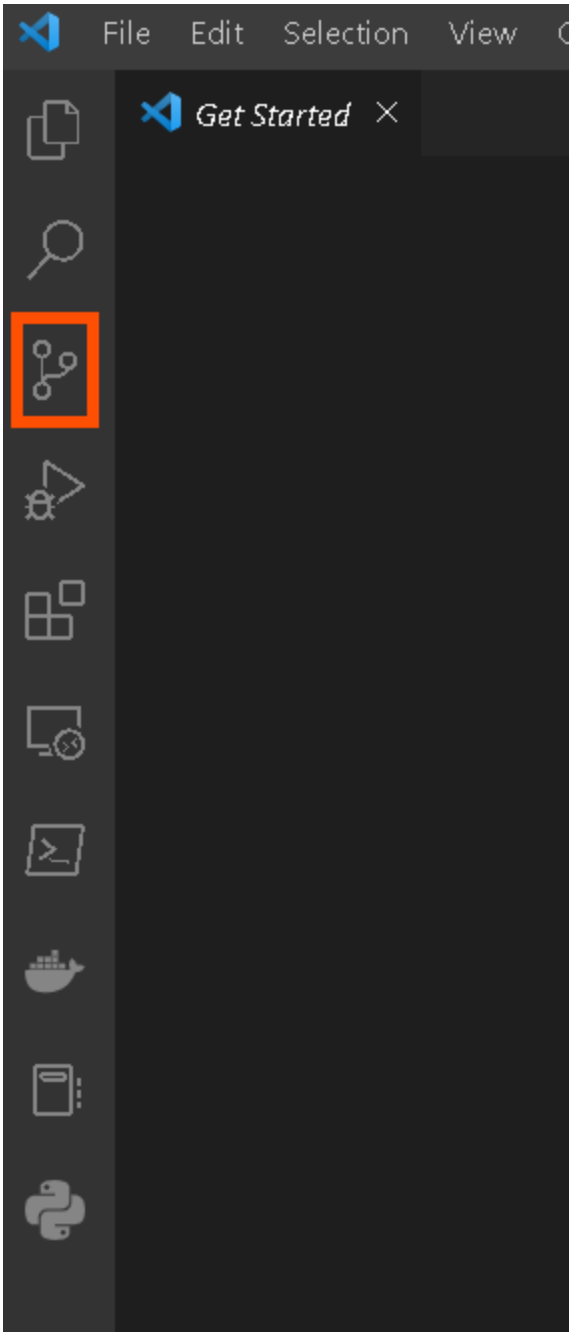


You now have the latest version of `dev` in your working branch.

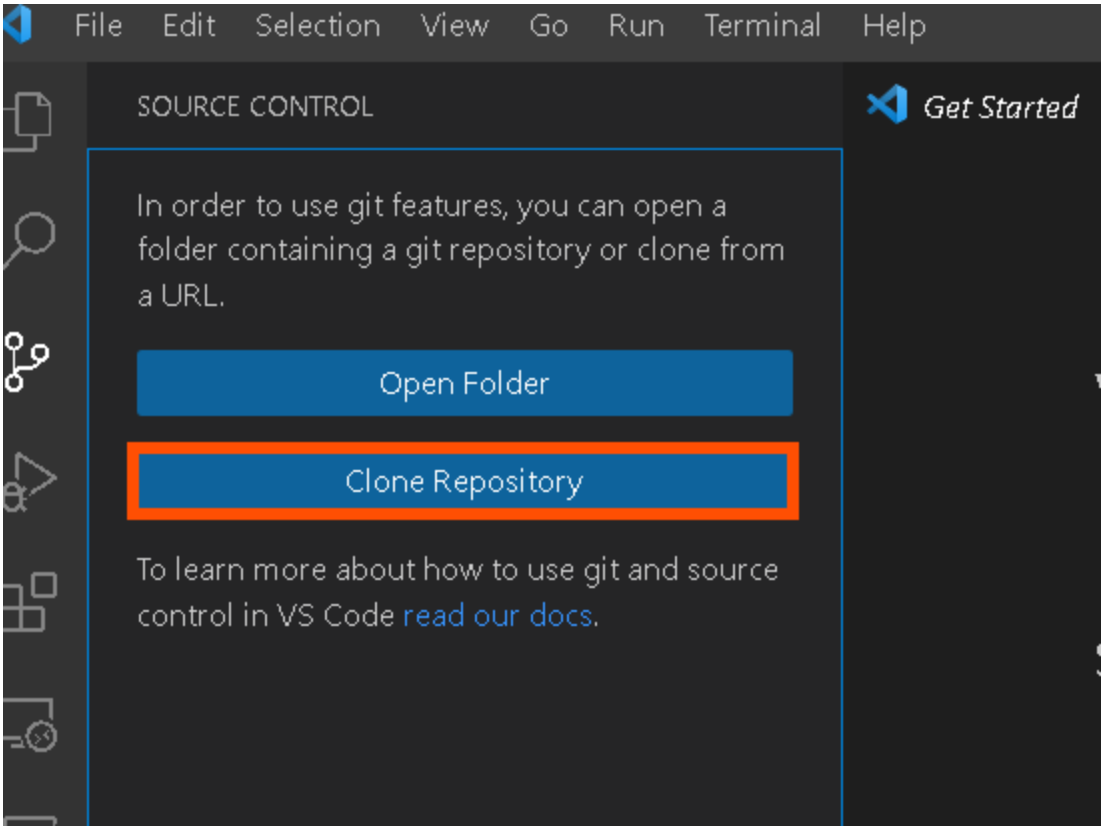
Visual Studio Code

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

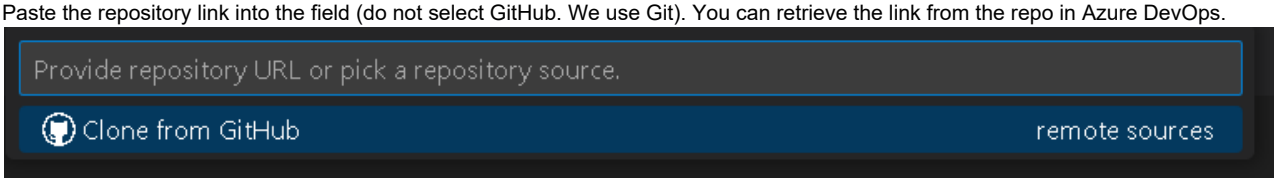
1. For organizational purposes, you can create a folder on your machine that you designate as your testing folder. Create a subfolder in this folder named after a project you plan to work on. This will be the folder you'll clone a repo to.
   a. For example, your subfolder directory could follow this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



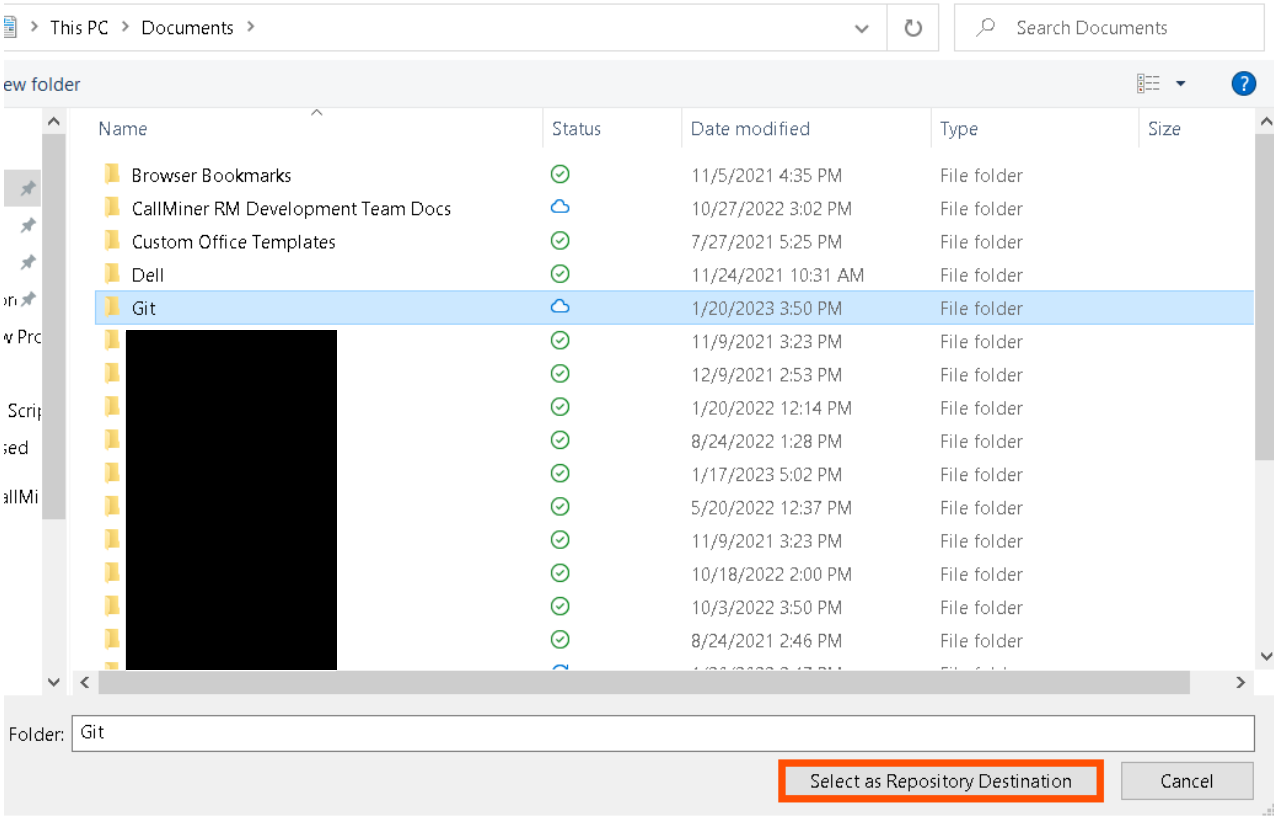
3. Click **Clone Repository**. However, if you don't see the option, click the **Refresh** link.



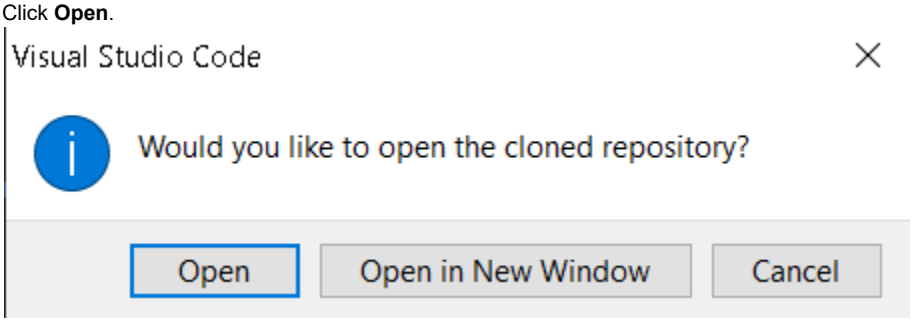
4. Paste the repository link into the field (do not select GitHub. We use Git). You can retrieve the link from the repo in Azure DevOps.



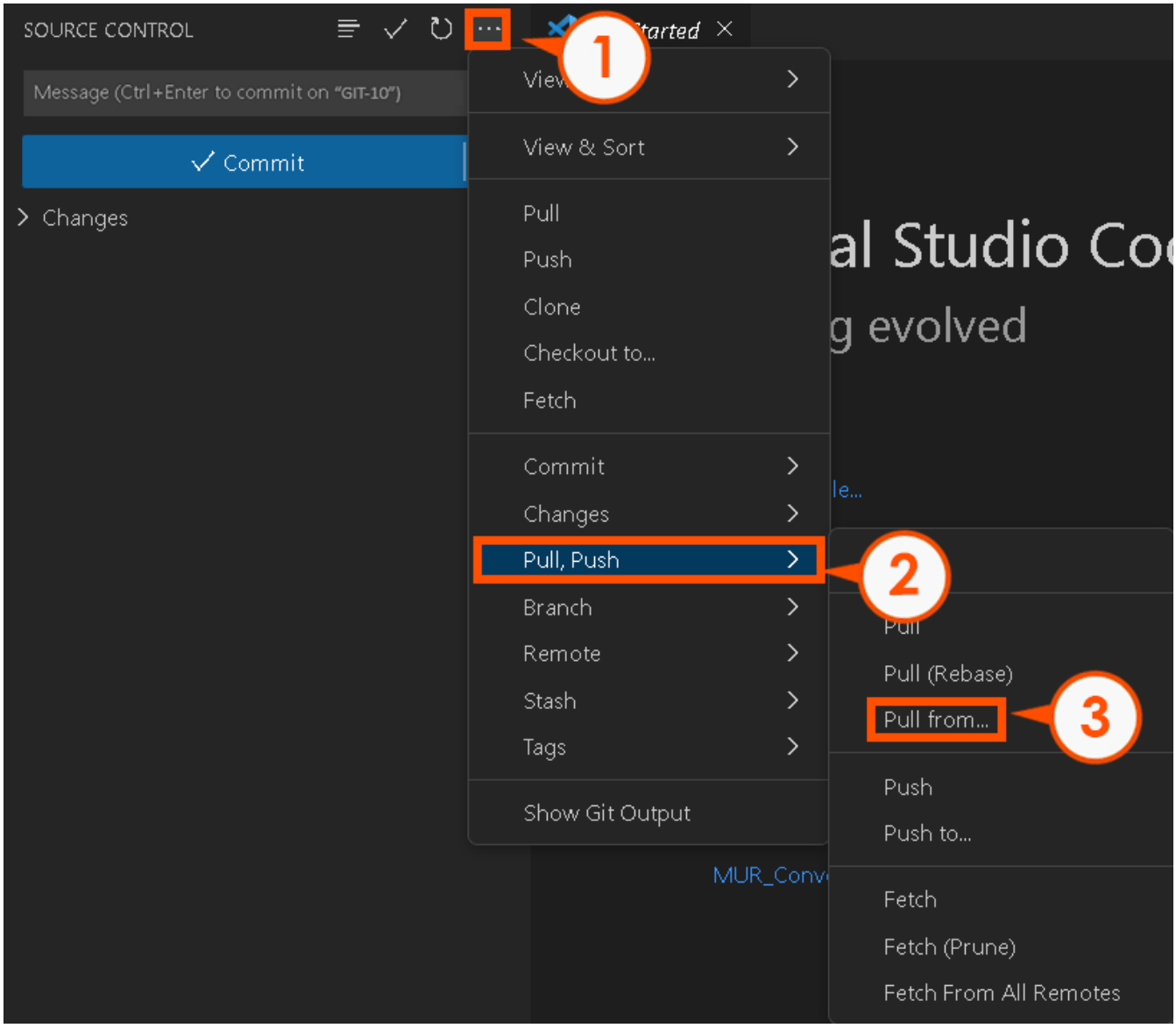
5. Find a location in your system to store the repo, or use the subfolder you created in step 1, then click **Select as Repository Destination**.



6. Click **Open**.

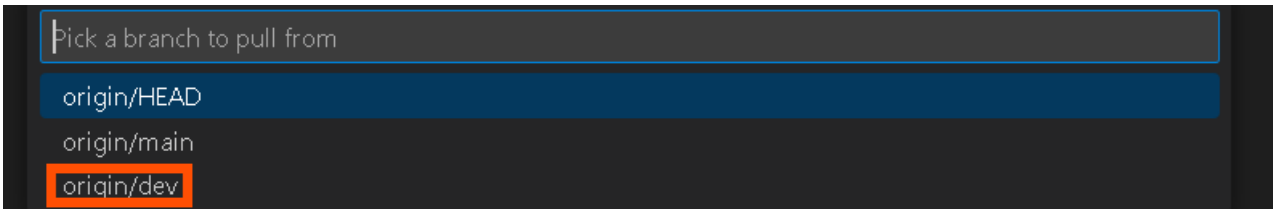


You have successfully cloned your repo. Just like in Visual Studio, once you need to get the latest updates from `dev`, you'll need to pull into your working branch. Follow the steps below to pull from `dev`:



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.

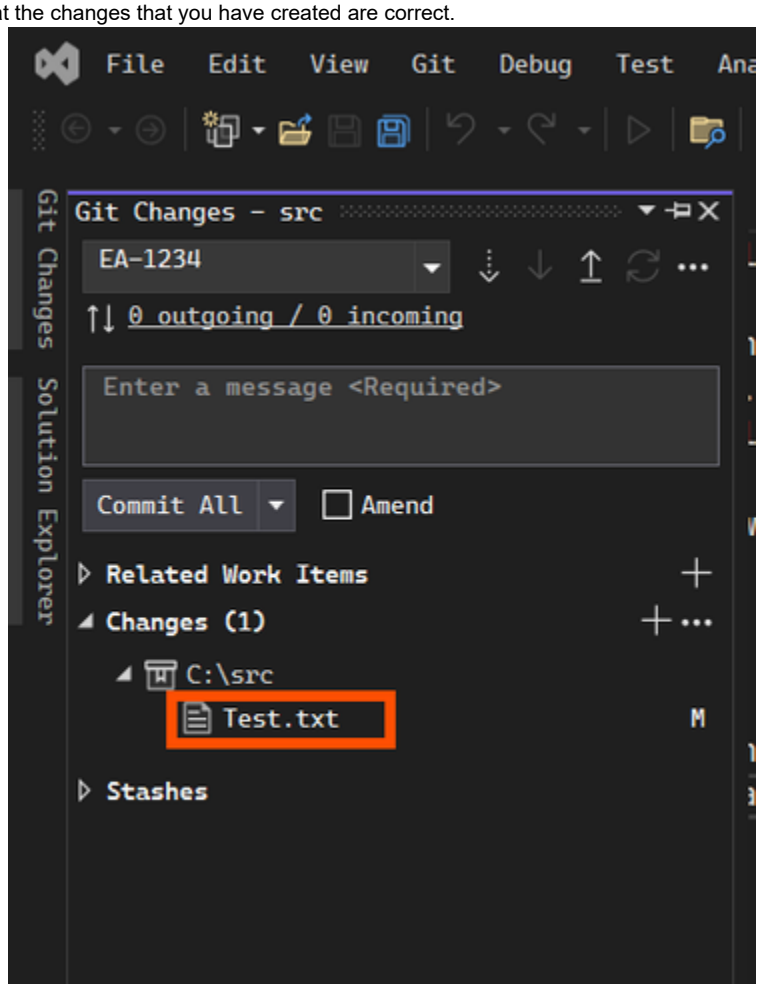
2.2.2 - Pushing

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.

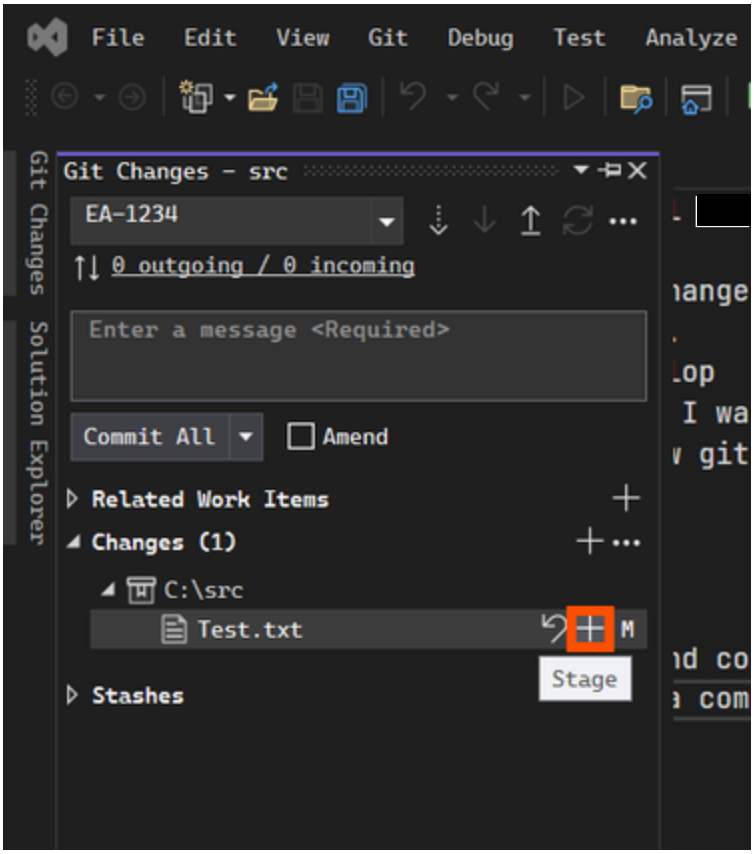
When the time comes to push the latest version of your files to a remote repo, you need to **push** your files. Follow the steps below to learn how to do this in both Visual Studio and Visual Studio Code:

Visual Studio

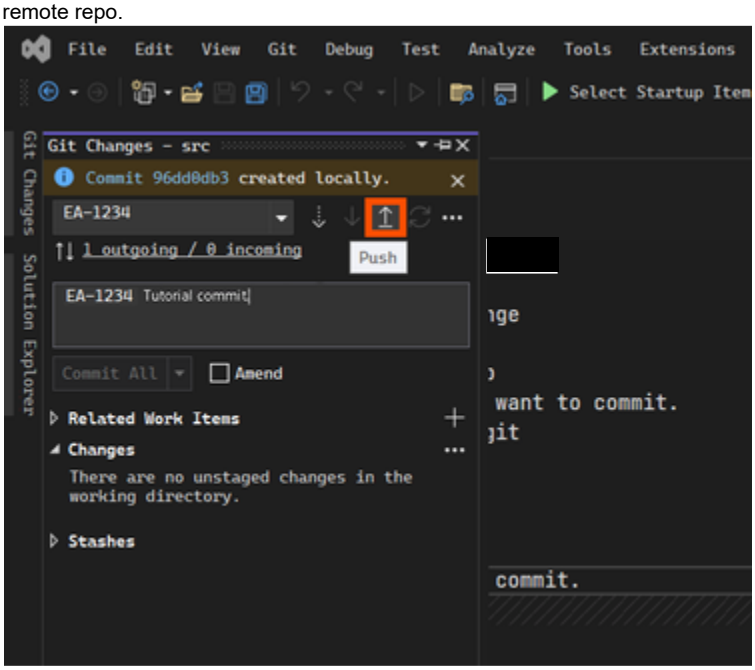
1. In the **Git Changes** tab, select the remote repo you'd like to push to, then locate your file that you have modified. Double-click the file to open a diff and verify that the changes that you have created are correct.



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



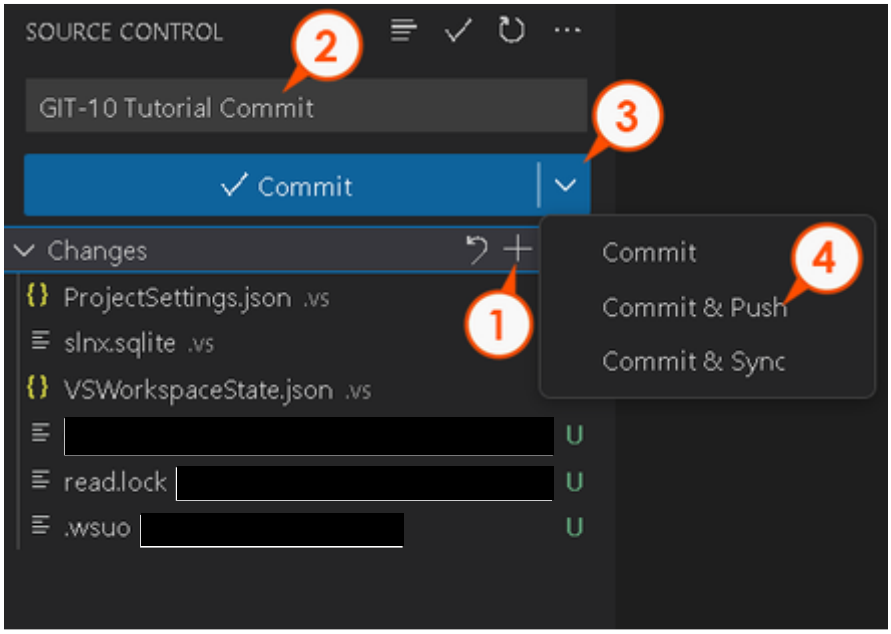
3. Enter a commit message that includes your Jira ticket number (remember that this should be the name of your branch), then click the **Push** button to push into the remote repo.



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:

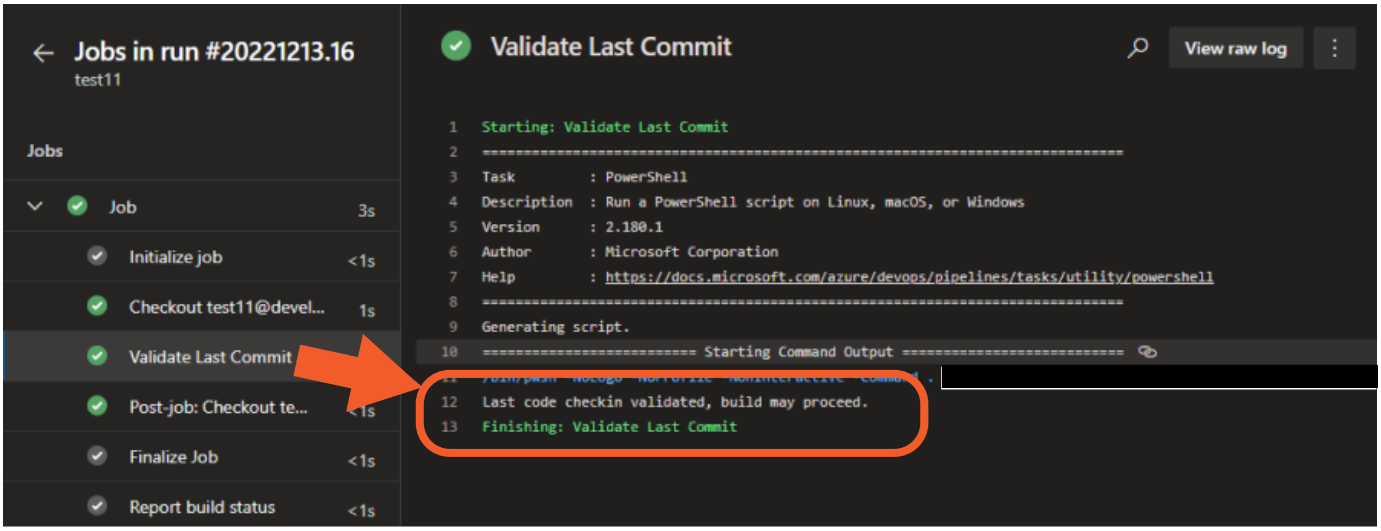
- Have a Jira ticket referenced in the comment so that it can be linked between the commit and ticket.
- Be related to a Jira Story or a Bug.
- Relate to a Jira ticket that has been sized.

Your code check-ins should not:

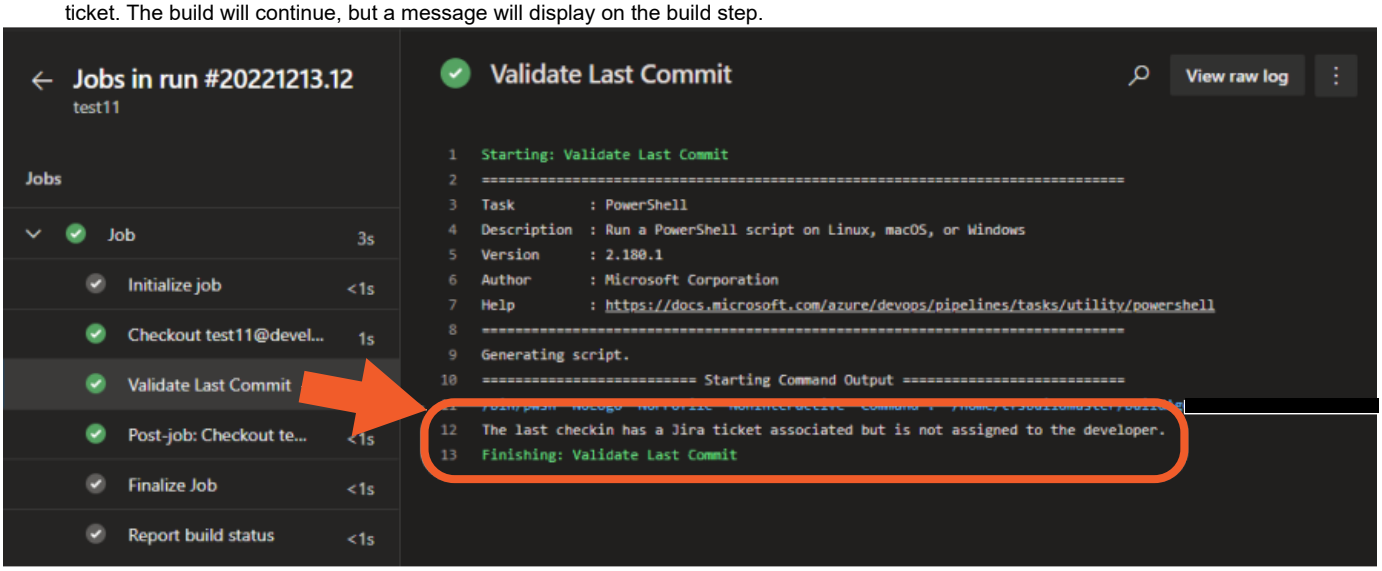
- Be attempted without attaching a Jira ticket number.
- Be a ticket that isn't a Story or a Bug (e.g. an Epic, an Initiative, etc.).
- Be attached to a Jira ticket that hasn't been sized.

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

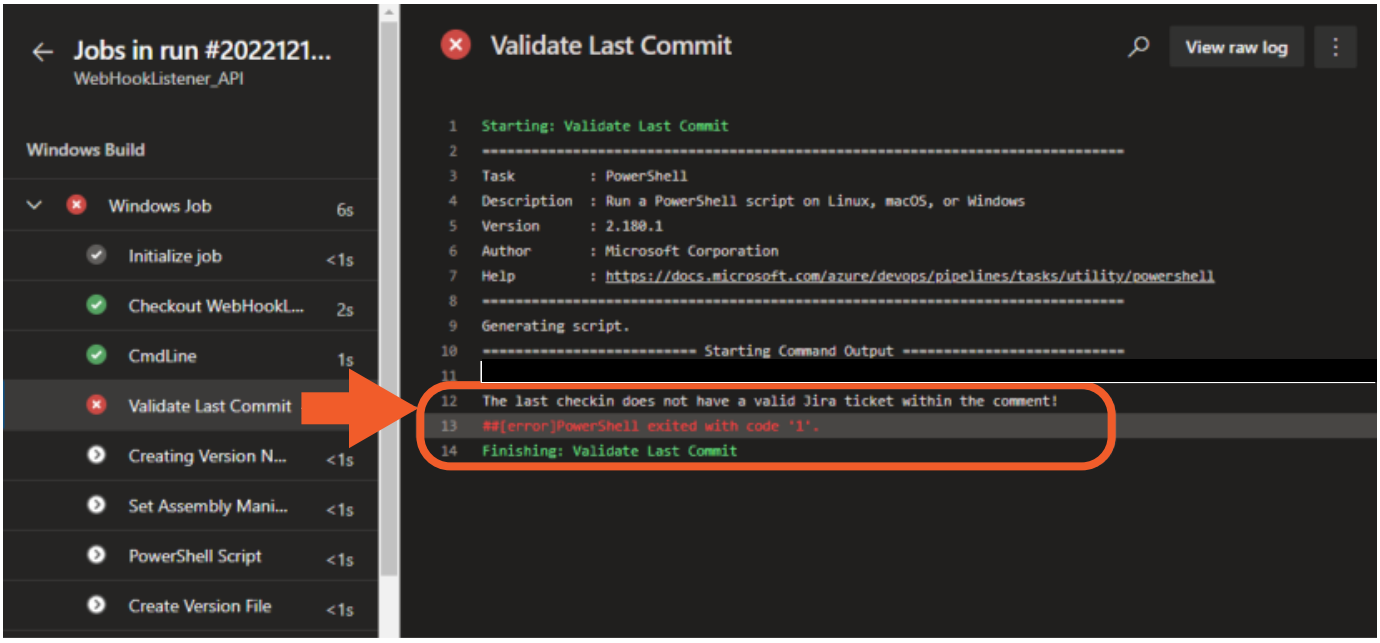
- ✅ If the developer enters a valid Jira ticket number, and they are assigned that ticket:
  - **Step Behavior**: Step will pass, build will proceed.
  - **Step Message**: Last code check-in validated, build may proceed.
  - **Requirement**: Every commit should have a Jira ticket referenced in the comment, 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



- ✅ 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.

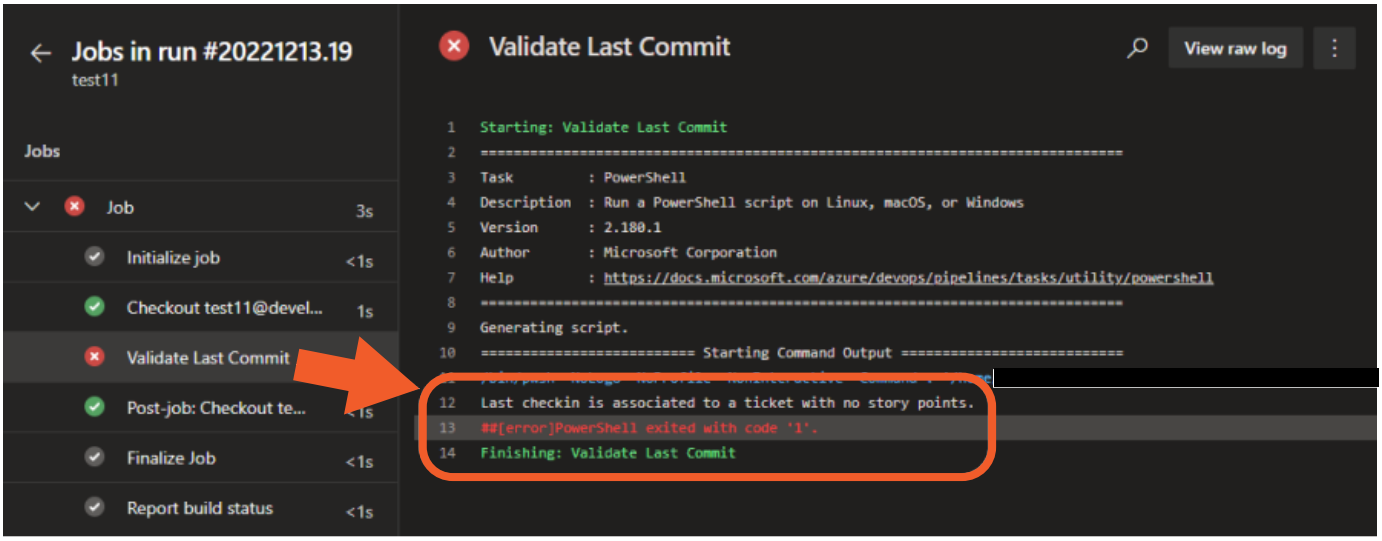


- ❌ If the developer checks-in code, but does not link it to a Jira ticket:
  - **Step Behavior**: Step will fail and build will not proceed.
  - **Step Message**: The last check-in does not have a valid Jira ticket within the comment!
  - **Requirement**: Every commit needs to be associated to a Jira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



- ❌ 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.

- ❌ 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.



2.4 - Code Reviews and Merging Your Code

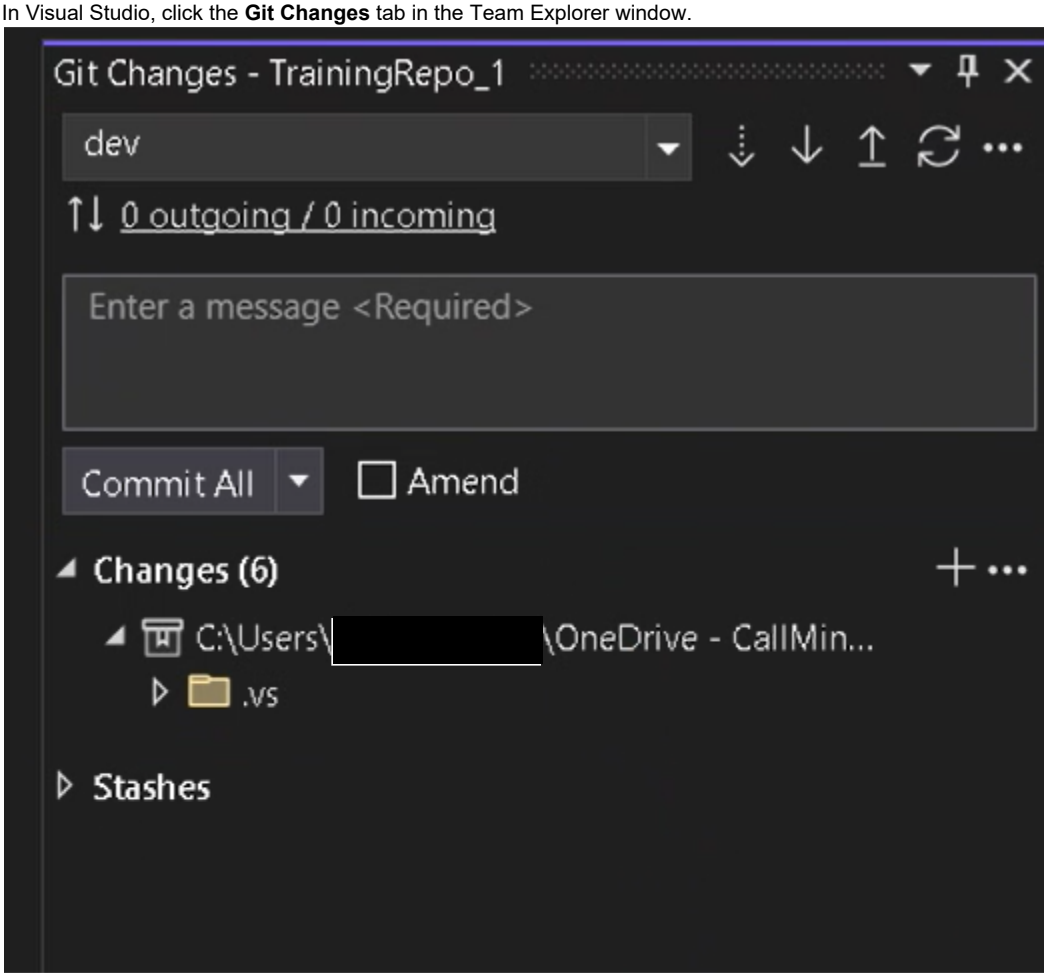
The steps to perform a code review and to merge your code into `dev` are located in the expandable boxes below.

> **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.

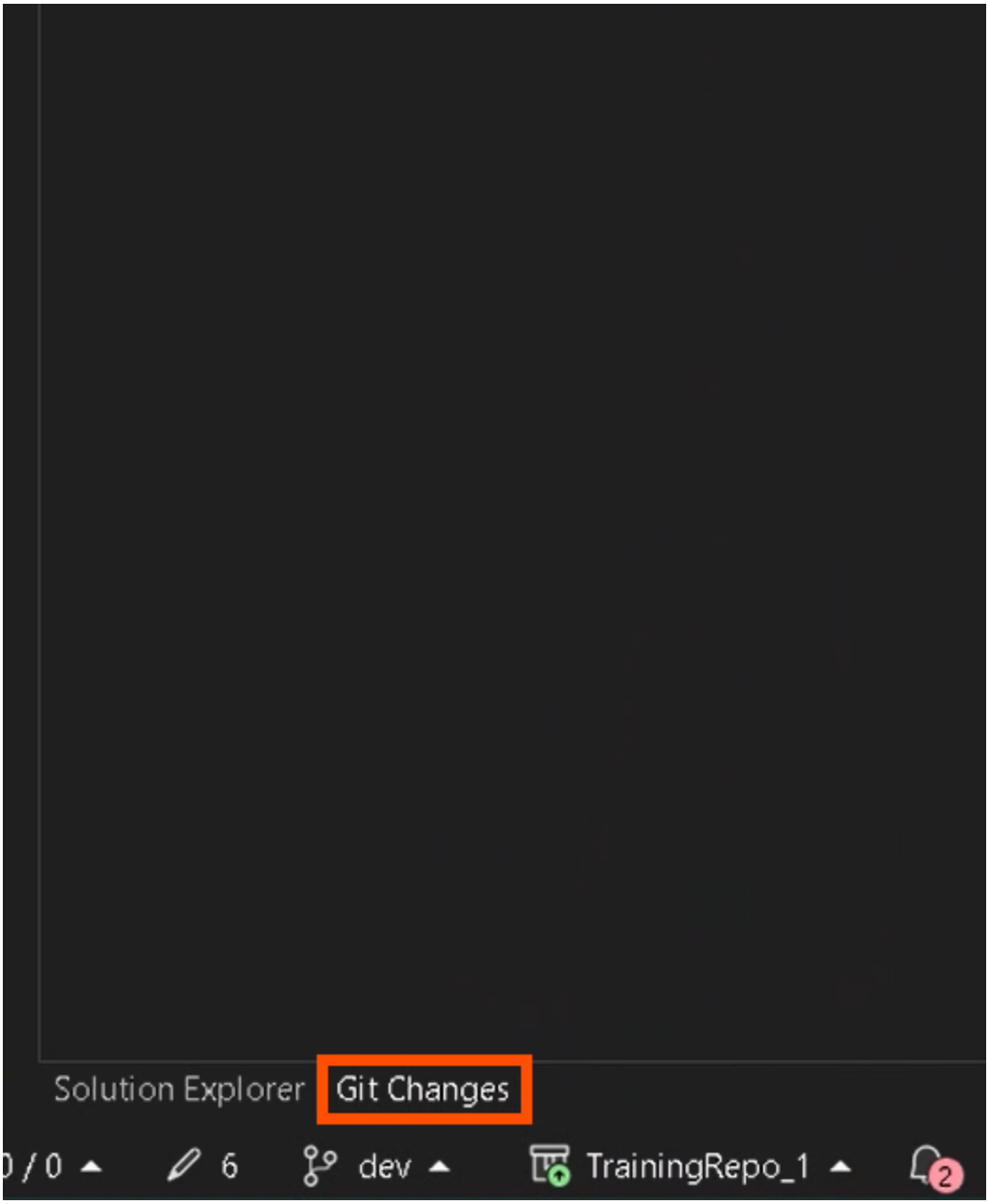
<details>
<summary>Section 1a: Code Changes and Creating a New Branch in Visual Studio</summary>

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.

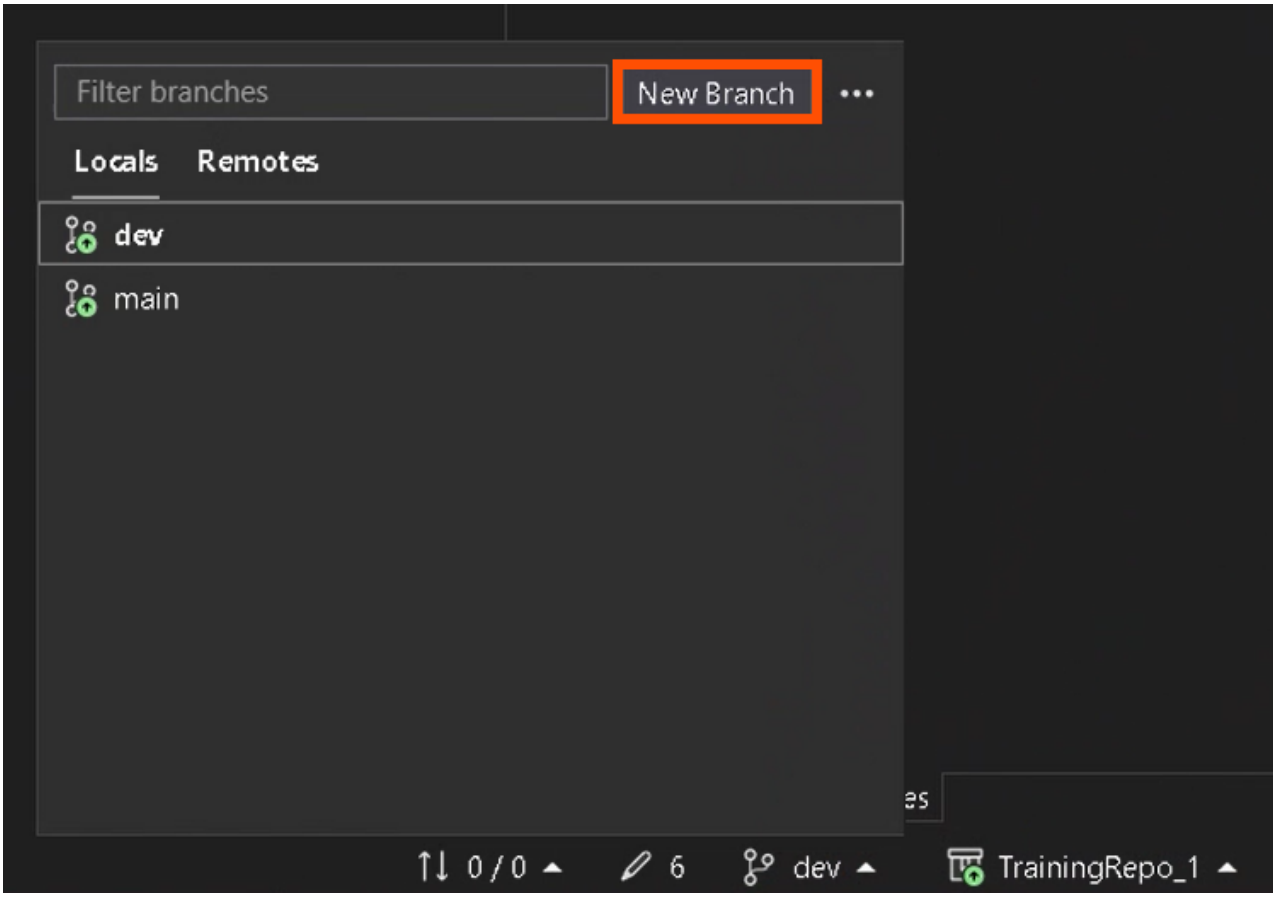
1. In Visual Studio, click the **Git Changes** tab in the Team Explorer window.



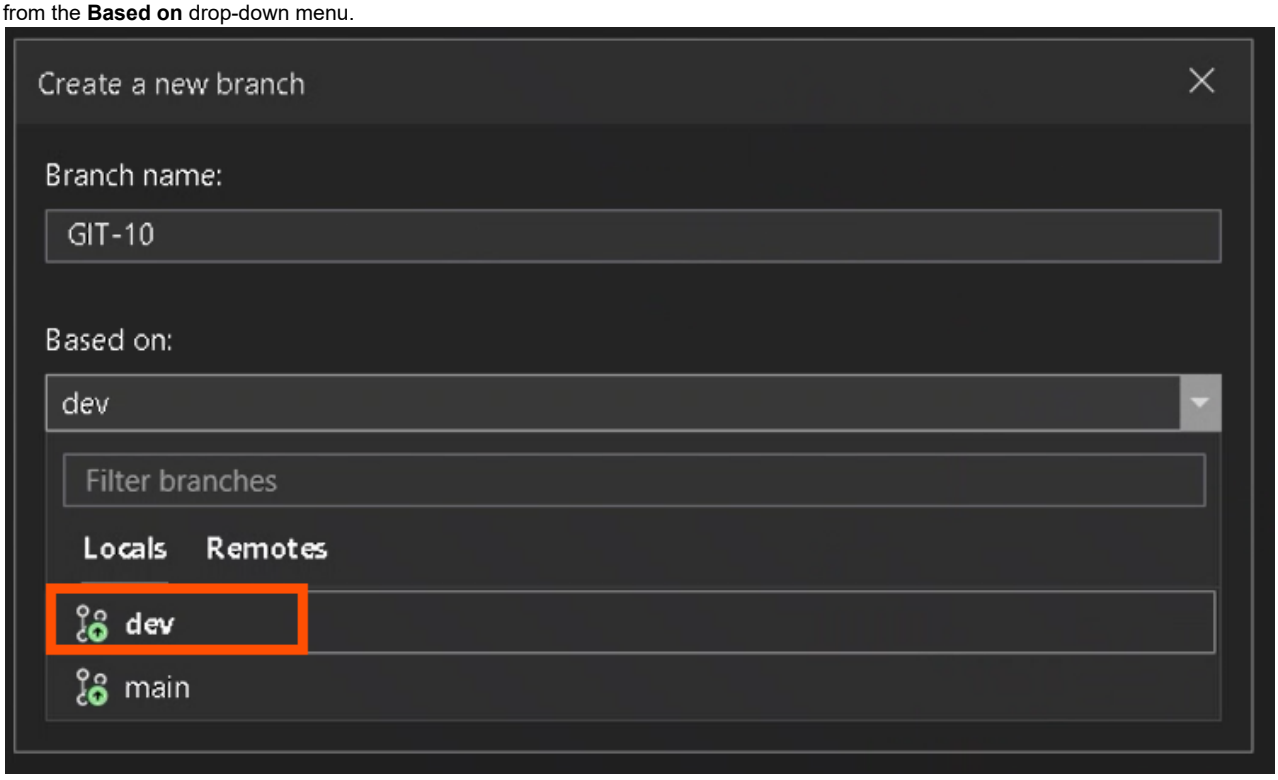
</details>



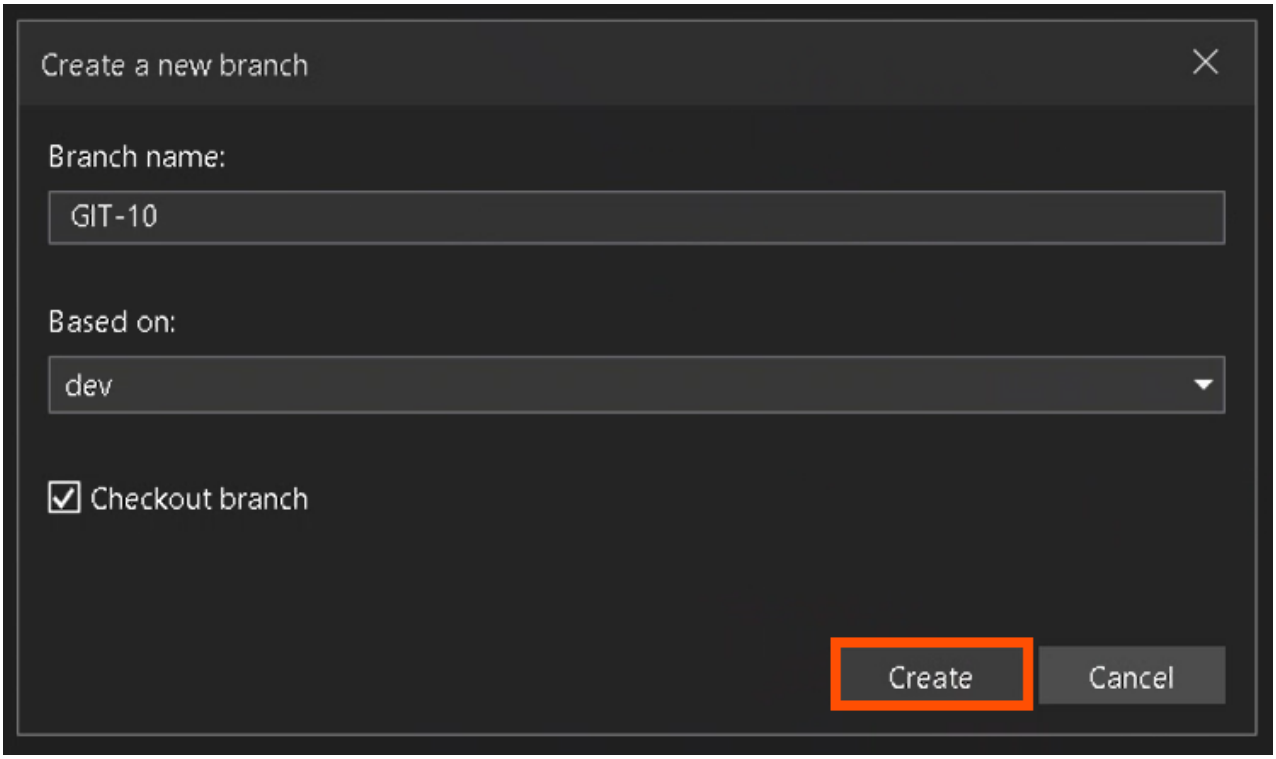
2. Click **New Branch**.



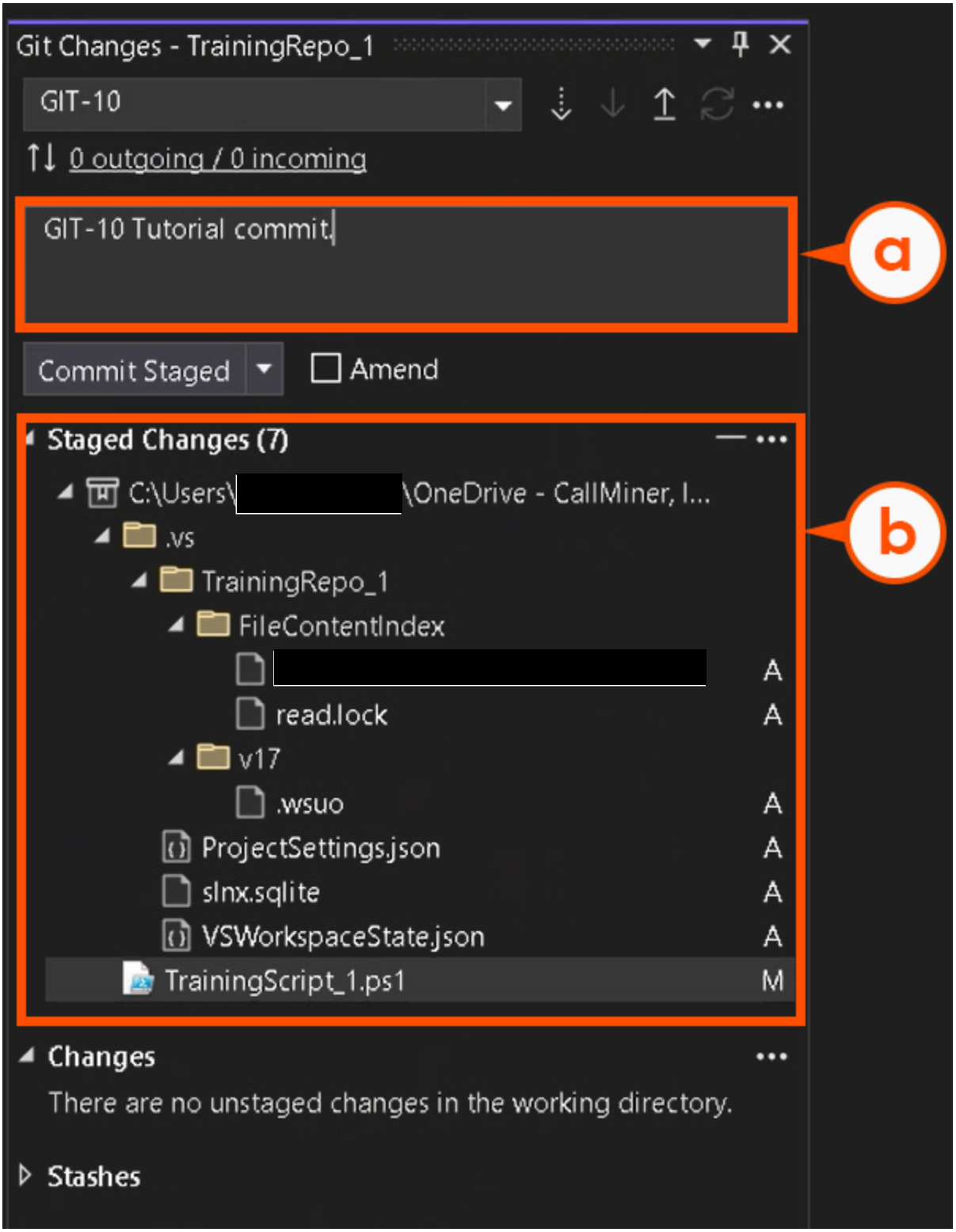
3. Enter your new branch name (be sure to follow the Git -- Branch Naming Conventions) in the **Branch name** field, then select `dev` from the **Based on** drop-down menu.



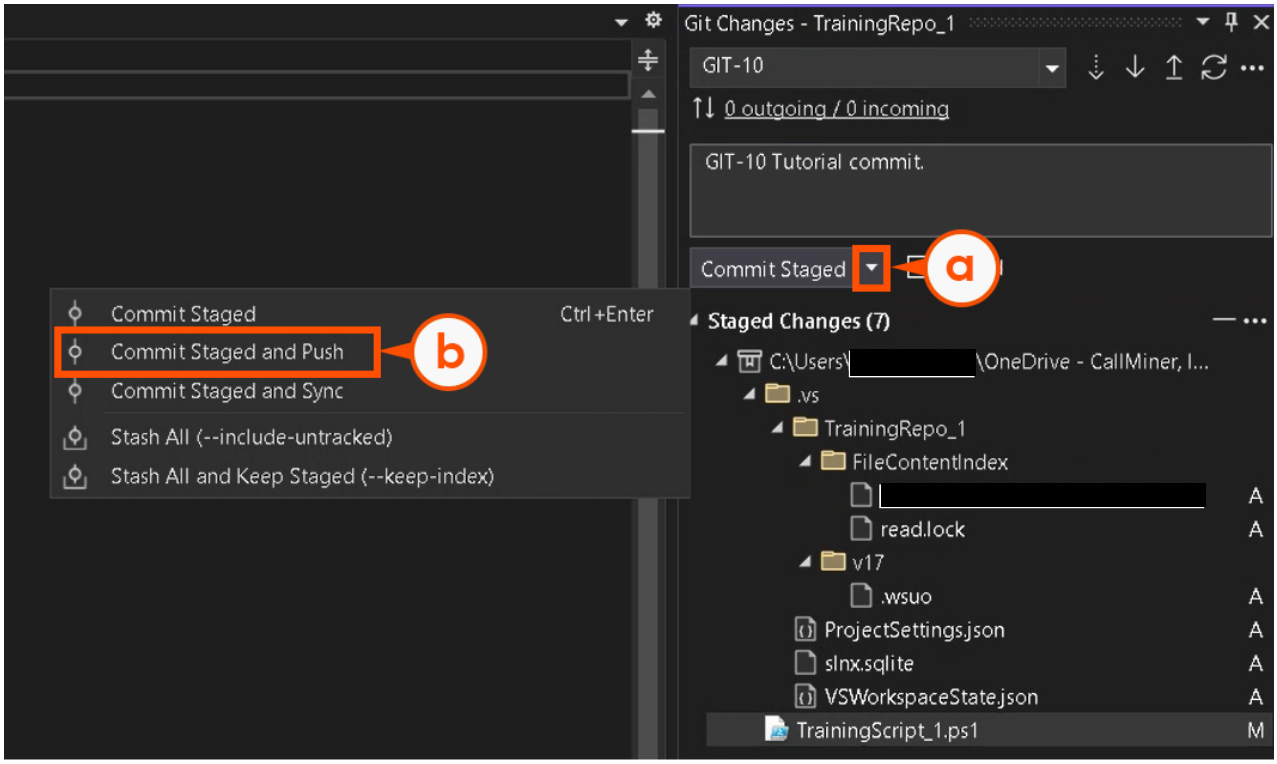
4. Make sure the **Checkout branch** box is checked, then click **Create** to finish creating your branch.



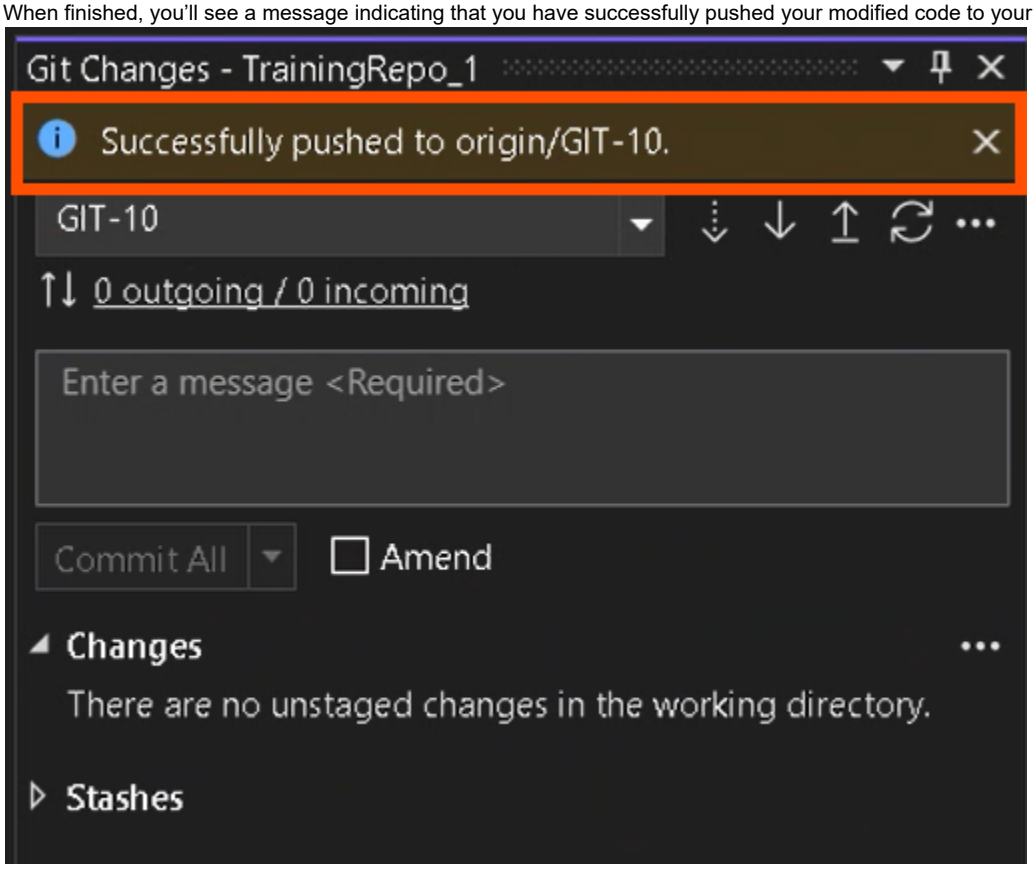
5. Modify your code as you normally would in Visual Studio. When you're ready to begin the check-in process, return to the Git Changes tab and stage your change.
6. Enter your comment in the comment field (**a**). It is VERY important that you include the Jira ticket you are associating with this code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

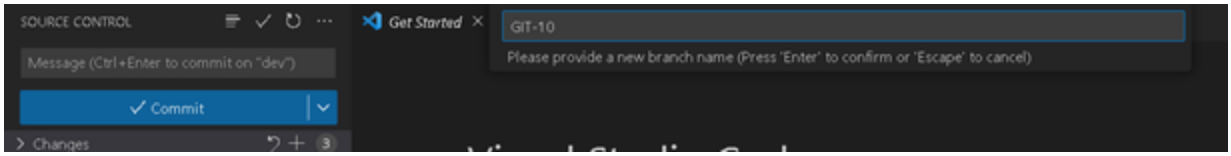


You have completed creating your own branch in Visual Studio.

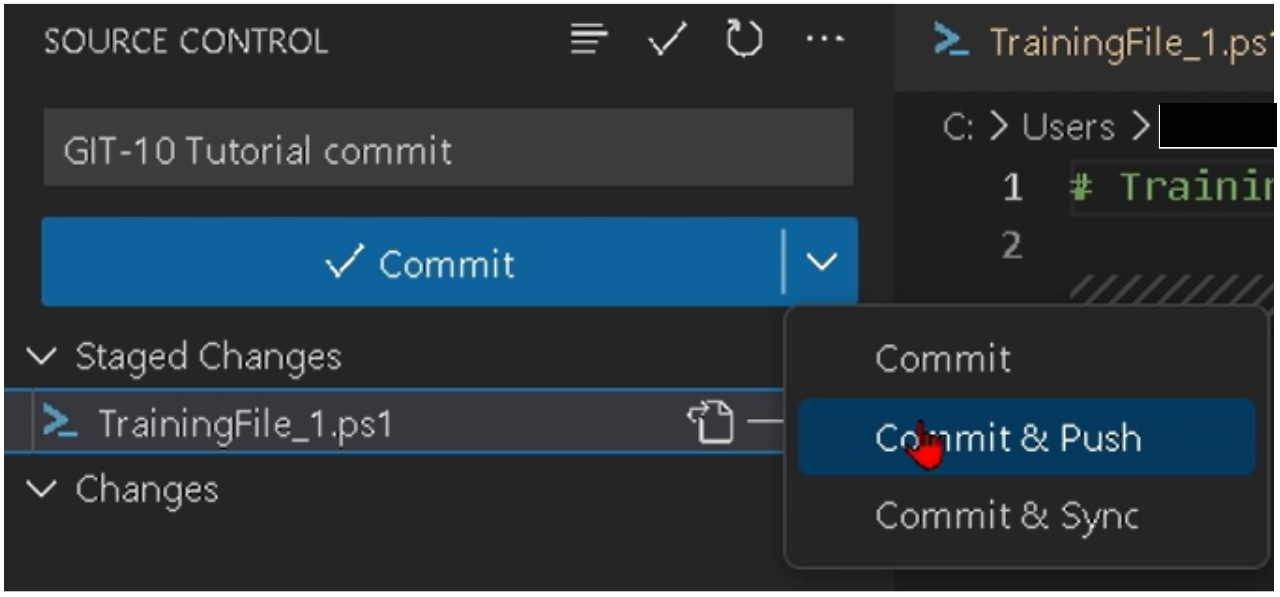
Section 1b: Code Changes and Creating a New Branch in Visual Studio Code

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**.

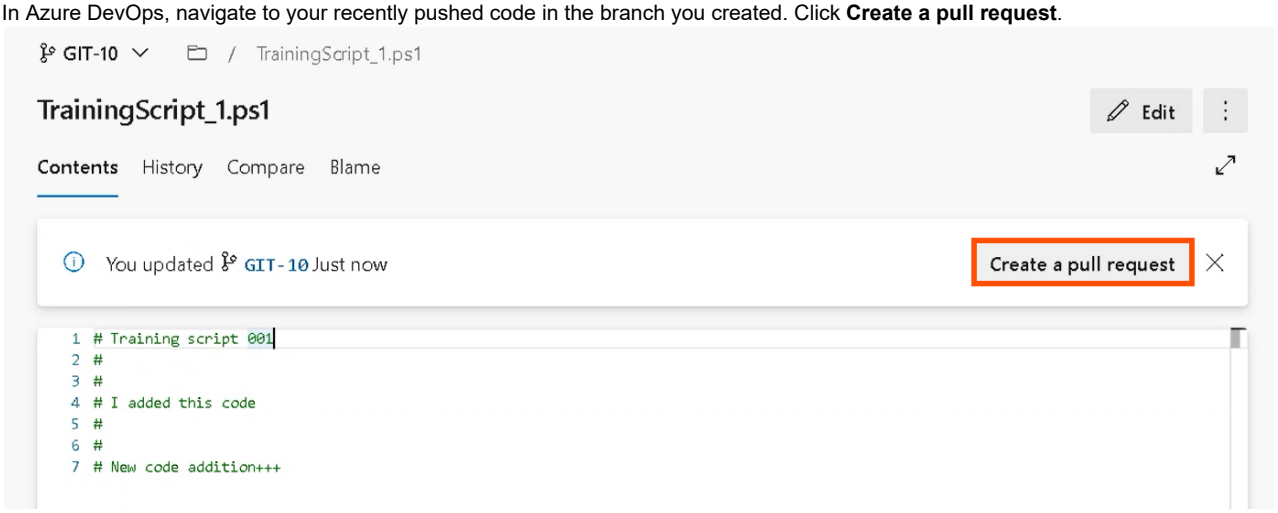


You have completed creating your own branch in Visual Studio Code. You can now move on to the next section in the process.

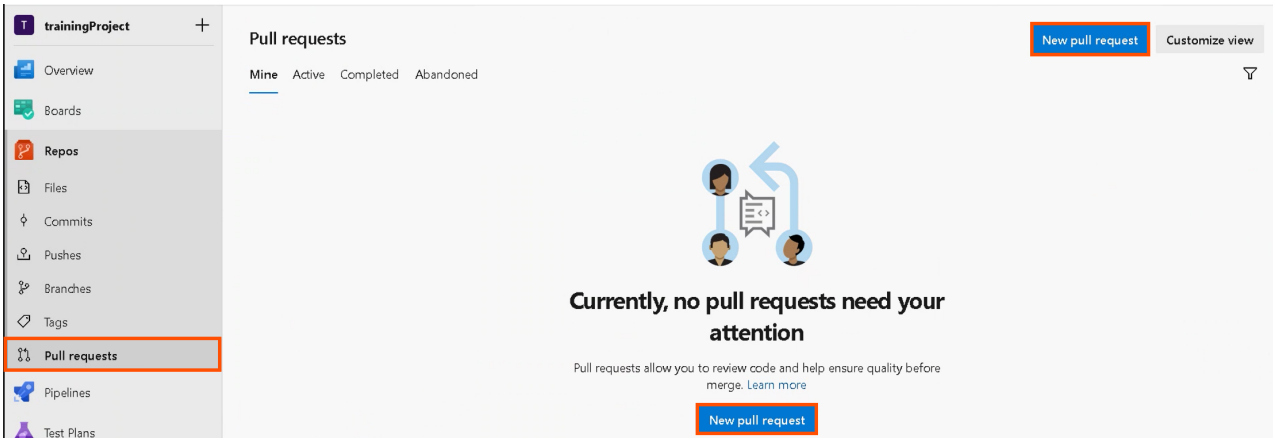
Section 2: Creating a Pull Request

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

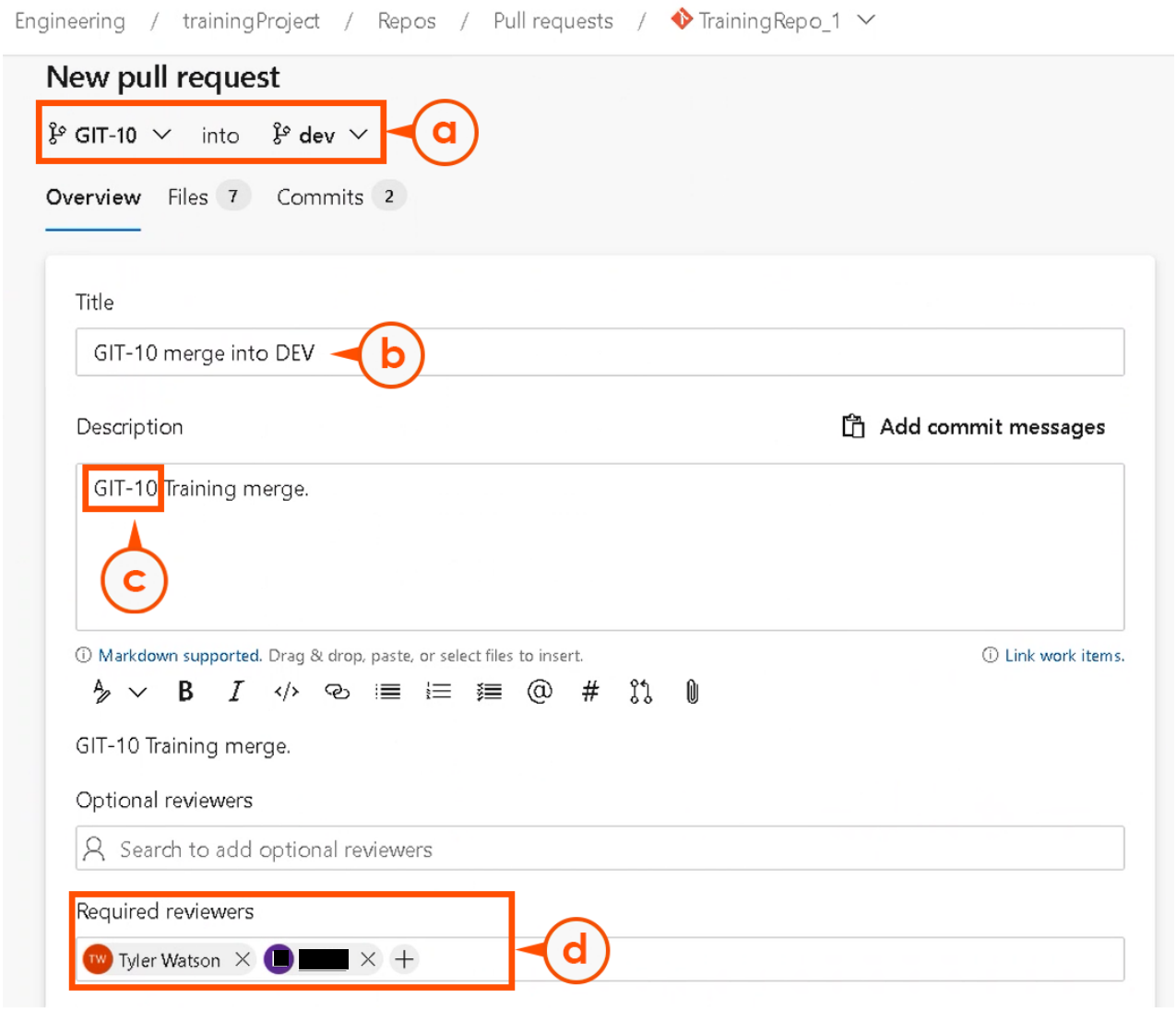
1. In Azure DevOps, navigate to your recently pushed code in the branch you created. Click **Create a pull request**.



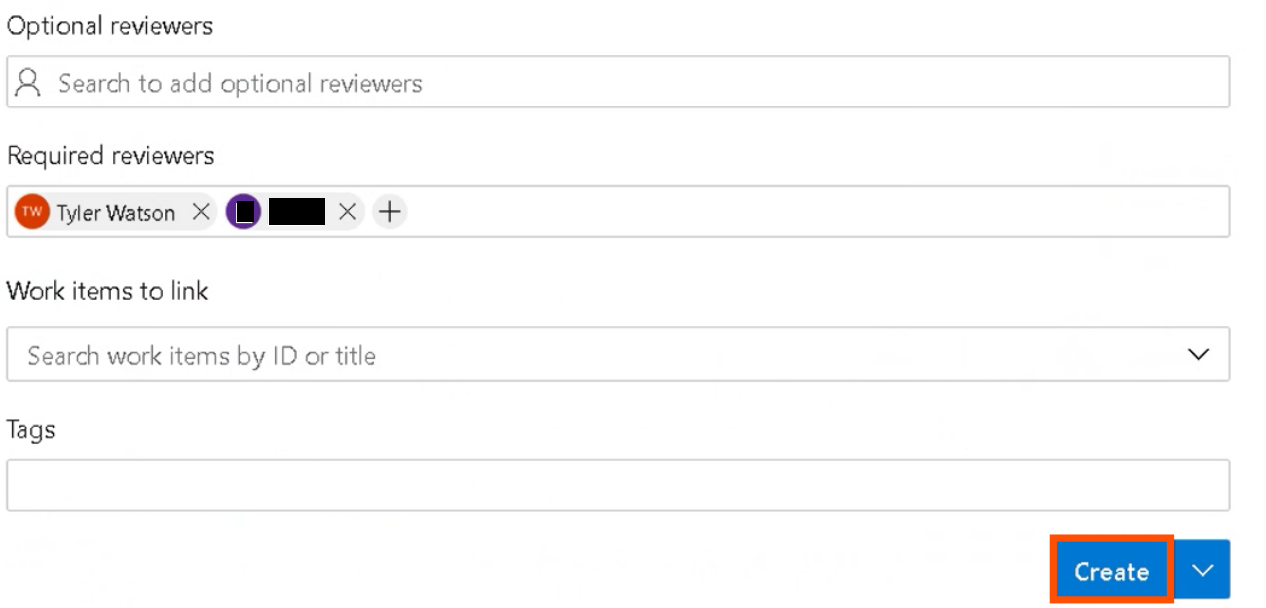
You can also create a new pull request by clicking **Pull Requests** in the **Repos** part of your dashboard, then clicking one of the **New pull request** buttons.



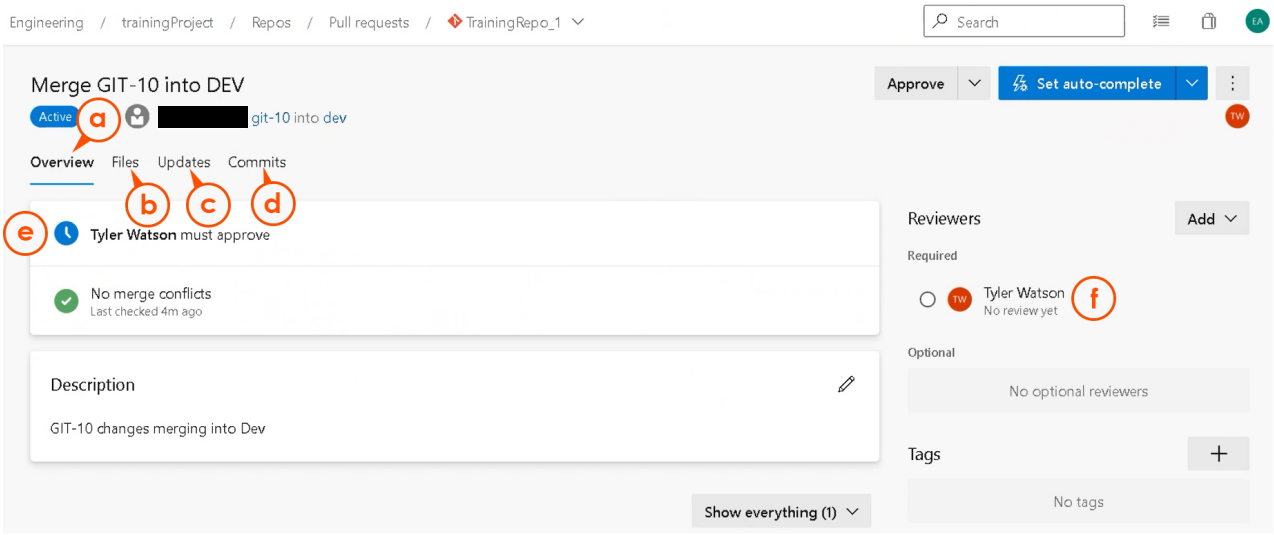
2. In the **New pull request** page, configure the following settings:



   a. Your source should be your newly created branch, while your target should be `dev`.
   b. Your title should include the name of the Jira ticket associated with your code commit.
   c. Your description should include the name of the Jira ticket associated with your code commit.
   d. 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**.



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



The options are as follow:

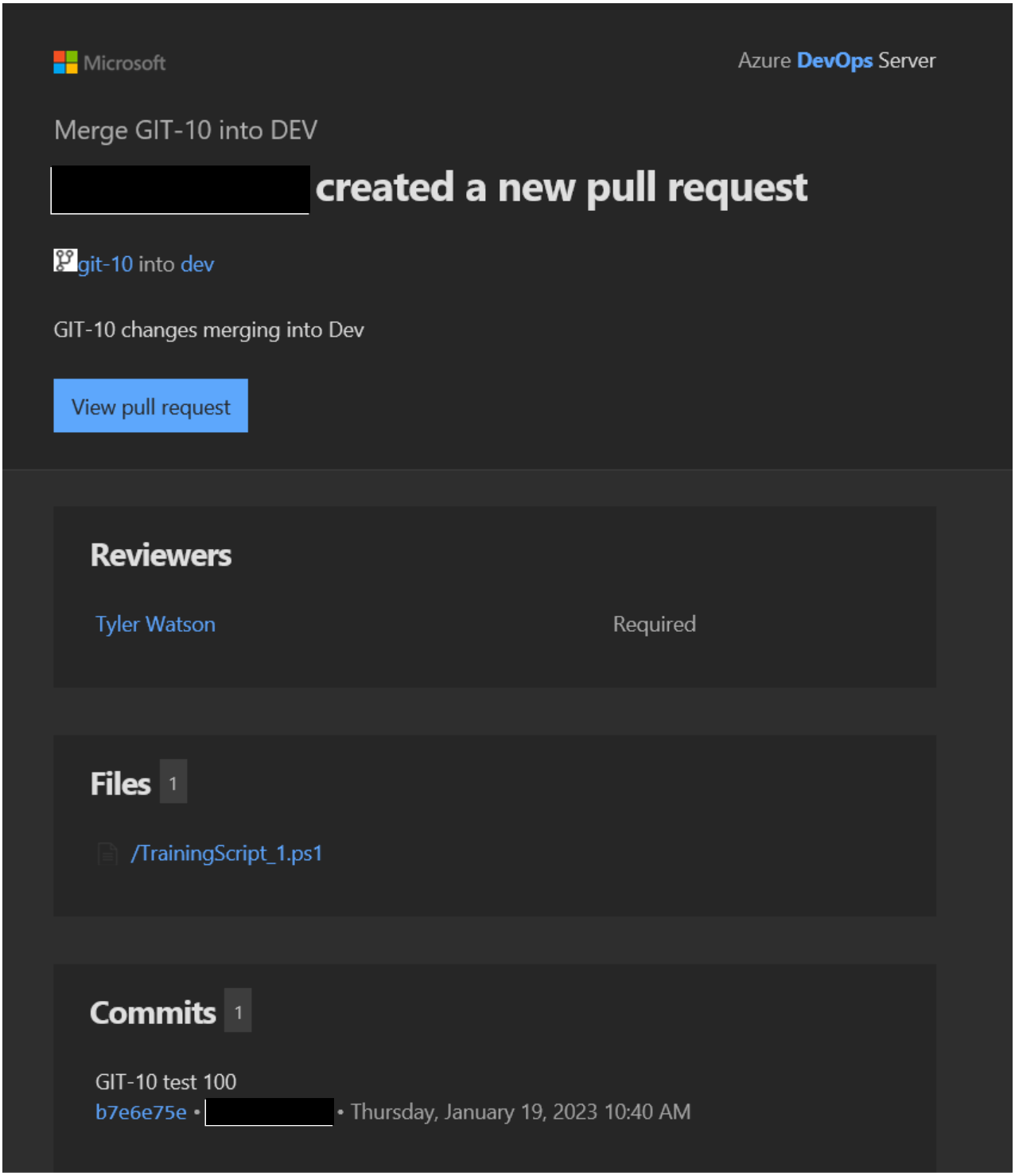
a. **Overview**: Provides an overview of the pull request.
b. **Files**: Contains all of the code files that have been modified and committed. By clicking on **Files**, you will be able to view all of them and review the changes made. Additionally, the review team that you have assigned can add comments 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.

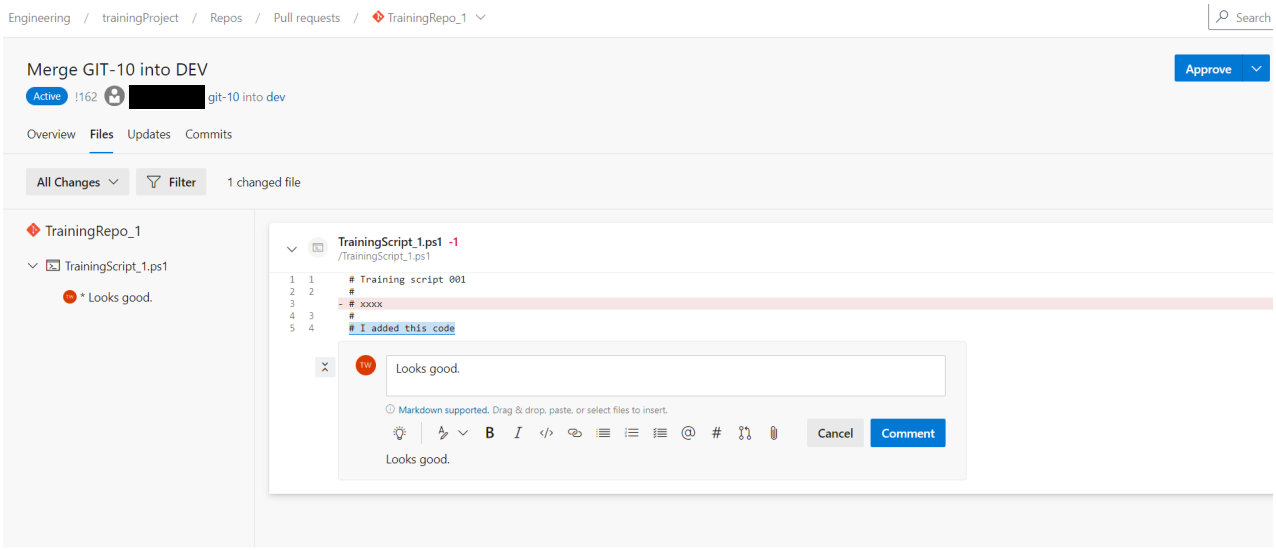
Section 3: Review and Approve the Code Changes

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:

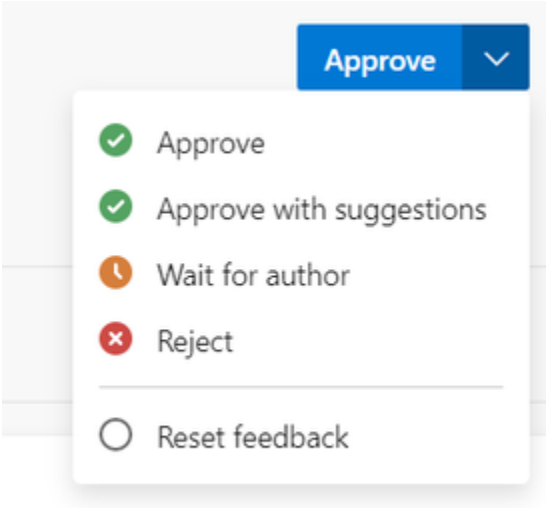
1. You, as the reviewer, will receive an email notifying you that you've been requested to review a pull request made by another dev. Click **View pull request** in the notification email.



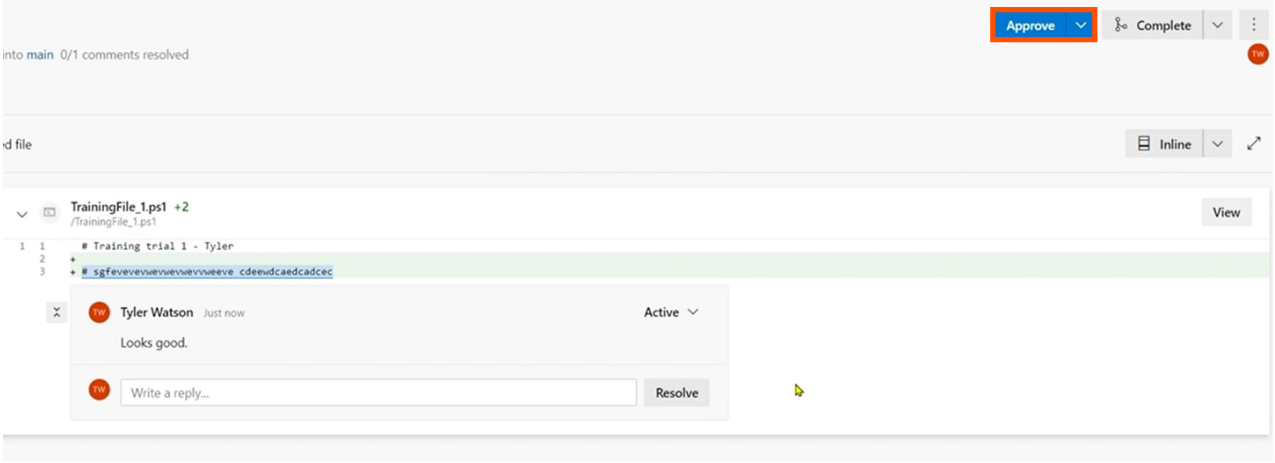
2. To view the requested piece of code, click **Files** in the **Pull Request** page, then comment with any feedback you may have about the code.



3. 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.



4. Once you see no problems with the code, click **Approve** to approve the code.
5. 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.

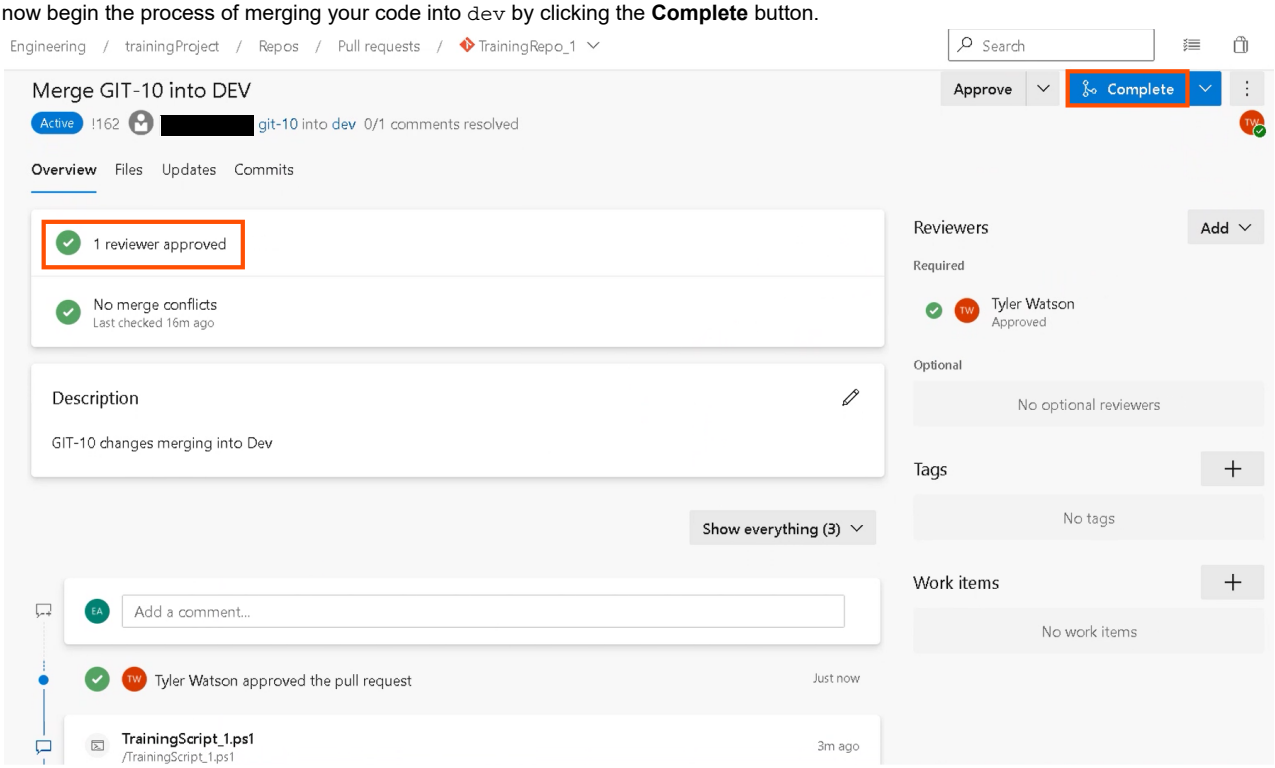


You have now completed reviewing the code and can move on to the next step in the process.

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.



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

Merge type

Merge (no fast forward)

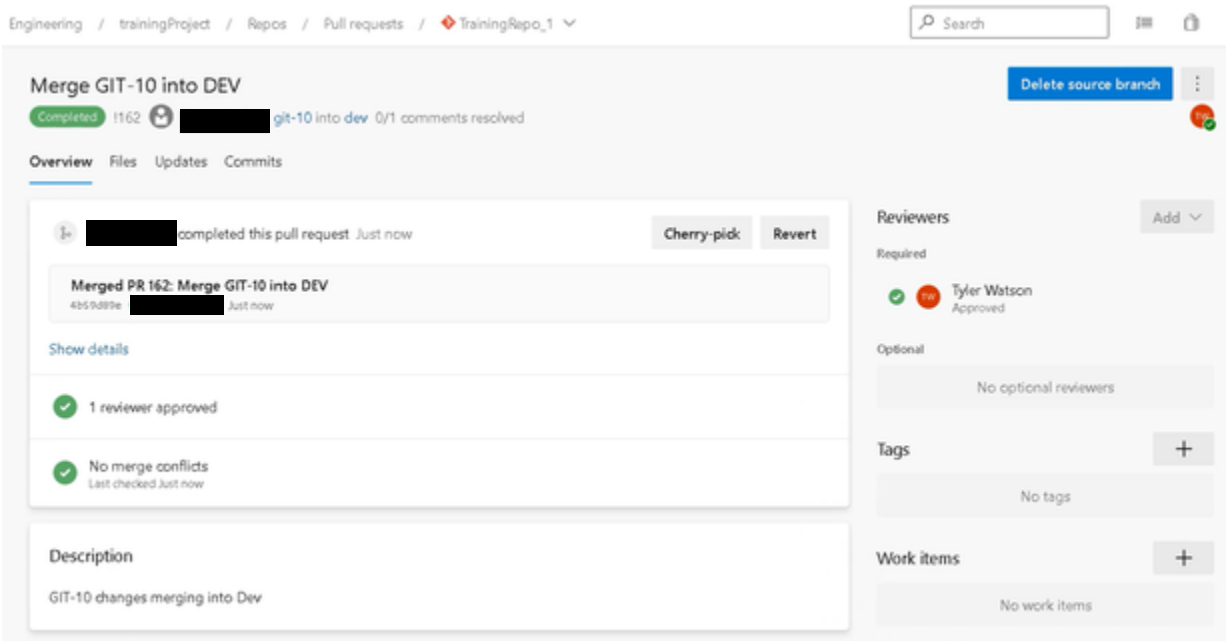
Post-completion options

- [x] Complete associated work items after merging
- [x] 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`.



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

3.1 - Team Resources

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