Getting Started with Git using Team Foundation Server 2017

Lab version: 15.0.26228.0

Last updated: 3/2/2017

1. **TABLE OF CONTENTS**

[Overview 1](#_Toc476253081)

[Prerequisites 1](#_Toc476253082)

[About the Fabrikam Fiber Scenario 2](#_Toc476253083)

[Exercise 1: Getting Started with Git 2](#_Toc476253084)

[Task 1: Creating a Git Repository 2](#_Toc476253085)

[Task 2: Clone Git Repository 4](#_Toc476253086)

[Task 3: Commit Code and Link to Work Item 5](#_Toc476253087)

[Task 4: Synchronize Commits with Server 11](#_Toc476253088)

[Exercise 2: Git Branching and Merging 14](#_Toc476253089)

[Task 1: Branching 14](#_Toc476253090)

[Task 2: Merging 21](#_Toc476253091)

[Task 3: Managing Security and Permissions 26](#_Toc476253092)

[Task 4: Branch Policies 28](#_Toc476253093)

[Task 5: Code Review and Merge using Pull Requests 30](#_Toc476253094)

# Overview

* 1. In this lab, you’ll learn about Git support in Team Foundation Server 2017 and Visual Studio 2017. Git is a distributed version control system. Git repositories can live locally (such as on a developer’s machine) and can now be hosted by Team Foundation Server. You will learn how to establish a local Git repository, which can easily be synchronized with a centralized Git repository in Team Foundation Server. In addition, you will learn about Git branching and merging support.

## Prerequisites

* 1. In order to complete this lab you will need the Visual Studio 2017 virtual machine provided by Microsoft. For more information on acquiring and using this virtual machine, please see [this blog post](http://aka.ms/almvm).

## About the Fabrikam Fiber Scenario

* 1. This set of hands-on-labs uses a fictional company, Fabrikam Fiber, as a backdrop to the scenarios you are learning about. Fabrikam Fiber provides cable television and related services to the United States. They are growing rapidly and have embraced Windows Azure to scale their customer-facing web site directly to end-users to allow them to self-service tickets and track technicians. They also use an on-premises ASP.NET MVC application for their customer service representatives to administer customer orders.

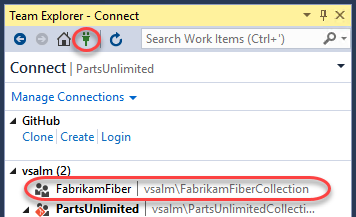
In this set of hands-on labs, you will take part in a number of scenarios that involve the development and testing team at Fabrikam Fiber. The team, which consists of 8-10 people has decided to use Visual Studio application lifecycle management tools to manage their source code, run their builds, test their web sites, and plan and track the project.

# Exercise 1: Getting Started with Git

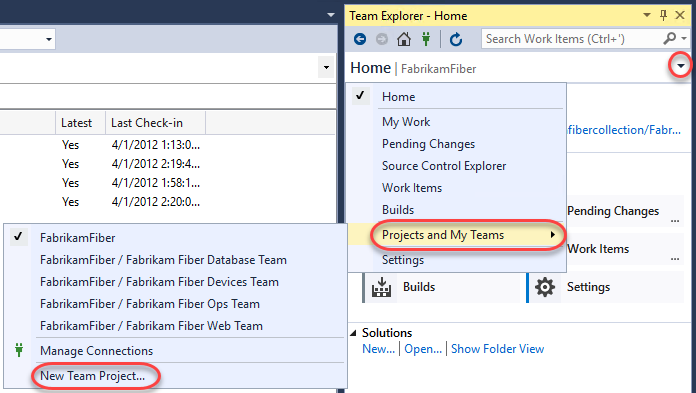
* 1. In this exercise, you will learn how to create, clone, and push commits to a Git repository with Team Foundation Server.

## Task 1: Creating a Git Repository

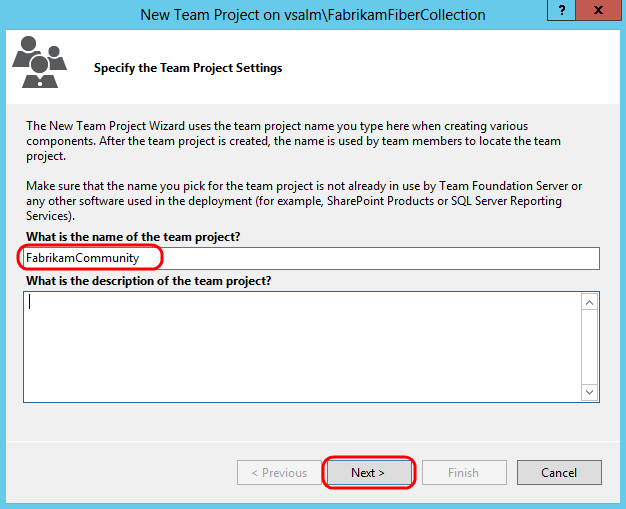
* 1. Log in as **Sachin Raj (VSALM\Sachin)**. All user passwords are **P2ssw0rd**.
  2. Launch **Visual Studio 2017** from the taskbar and open **Team Explorer**. You should now be connected to the **FabrikamFiber** team project. If you are not automatically connected to the **FabrikamFiber** project, click the **Manage Connections** and then double-click the **FabrikamFiber** project to do so.



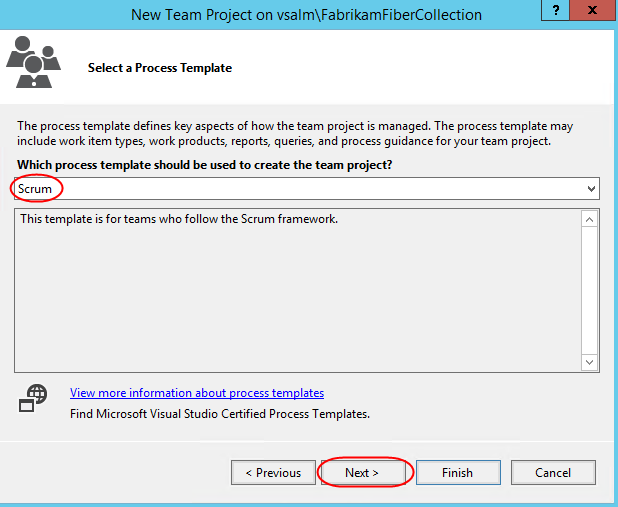
* 1. There are a few reasons why Fabrikam Fiber might want to use Git as their source control option within Team Foundation Server. One reason could be that they are collaborating with developers using a tool such as Xcode, which supports the Git protocol natively. Another reason could be that they have developers working offline (such as during a commute) who want to commit code locally when they are offline and check this code into Team Foundation Server when they get into the office. Microsoft now offers teams the ability to utilize Git without sacrificing the integrated application lifecycle management capabilities offered by Team Foundation Server. Visual Studio 2017 also provides developers with a great experience for working with any Git repository – whether it’s hosted by Team Foundation Server, a local repository, or another Git provider.
  2. From the **Home** dropdown menu, select **Projects and My Teams | New Team Project**.



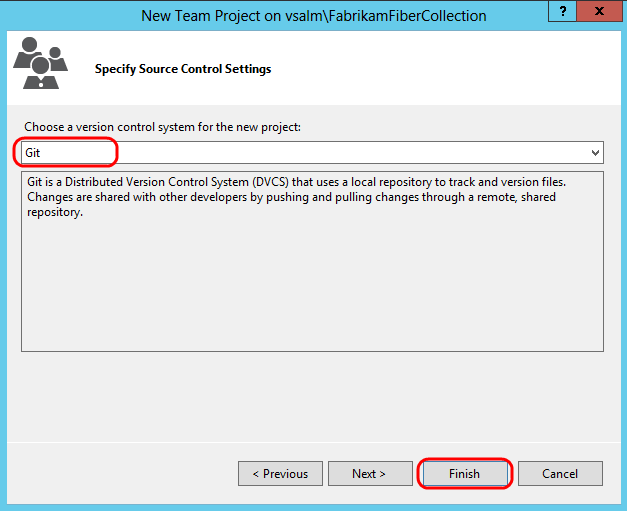
* 1. Name the new project **“FabrikamCommunity”** and click **Next**.



* 1. Select the **Scrum** process template and click **Next** to continue.



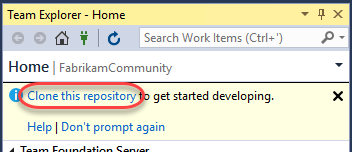
* 1. Select the **Git** version control system and then click **Finish**.



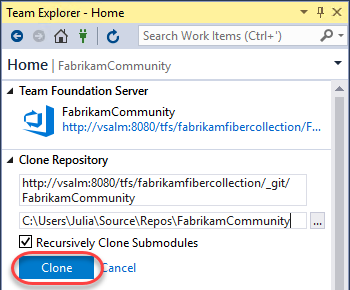
* 1. After the new Git team project has been created, click **Close** to return to Visual Studio.

## Task 2: Clone Git Repository

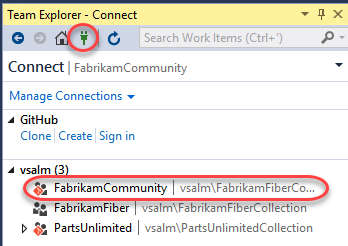
1. In **Team Explorer**, click **Clone this repository**.



1. Accept the default endpoint and repository location and then click **Clone**.

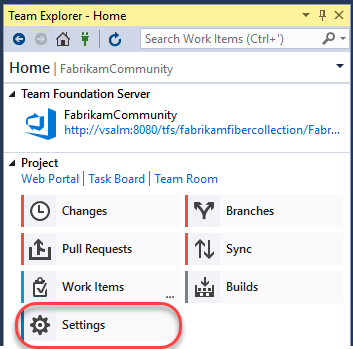


1. Click the **Manage Connections** button and double-click **FabrikamCommunity** to connect to the new team project.

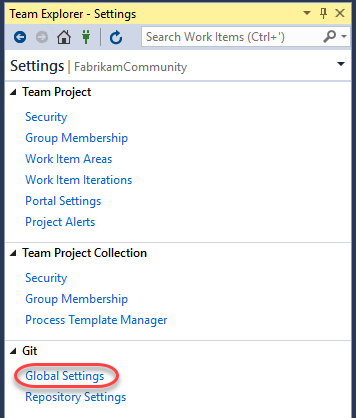


## Task 3: Commit Code and Link to Work Item

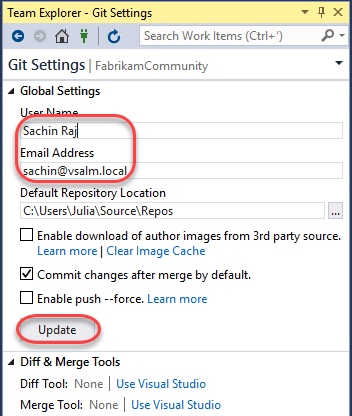
1. In **Team Explorer**, click the **Home** button, followed by **Settings**.



1. Click **Global Settings** under **Git**.



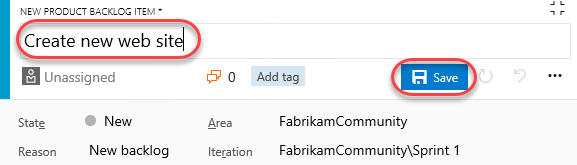
1. Enter an email address for Sachin Raj (**sachin@vsalm.local**) and then click **Update**.



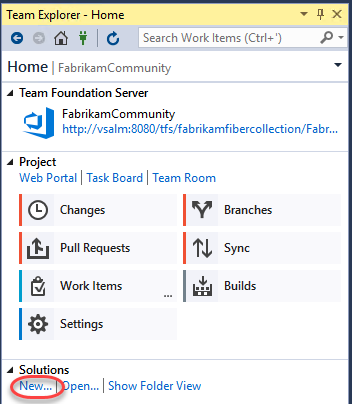
1. Click the **Home** button in **Team Explorer**.



1. Create a new work item for the product backlog by selecting **Team | New Work Item | Product Backlog Item** from the main menu. This will launch the **New Product Backlog Item** form in TFS via your browser.
2. Enter a title of **“Create new web site”** and then click the **Save** button. Take note of the **ID** once the work item is saved.



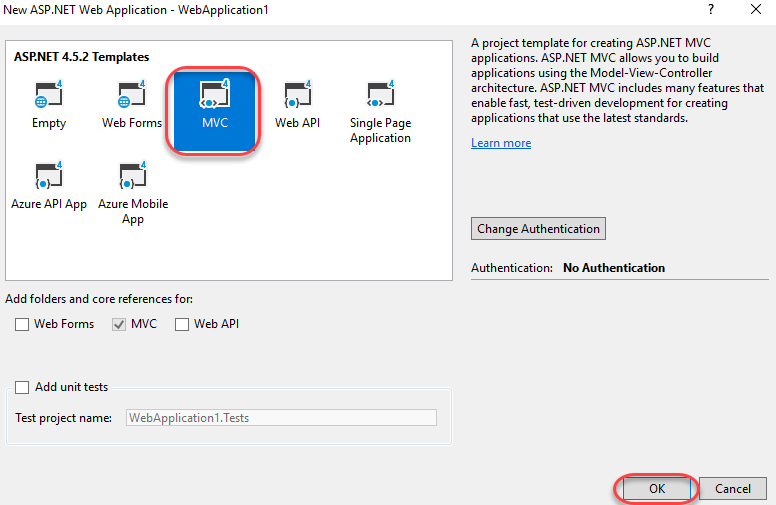
1. Return to Visual Studio. In **Team Explorer – Home**, click **New…** underneath the **Solutions** section.



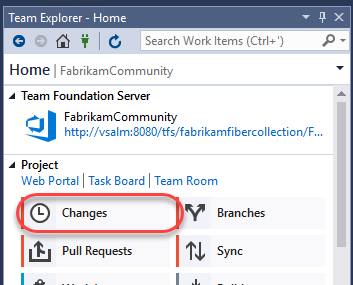
1. In the **New Project** window, select the **Visual C# | Web | ASP.NET Web Application** template and click **OK**.



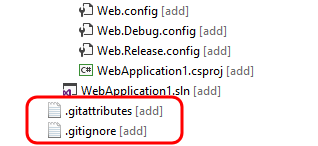
1. Select the **MVC** template and click **OK**.



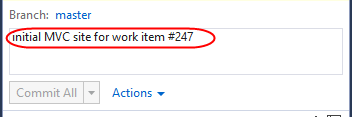
1. Once the project has finished creating, click **Changes** in **Team Explorer** to see the list of files to commit.



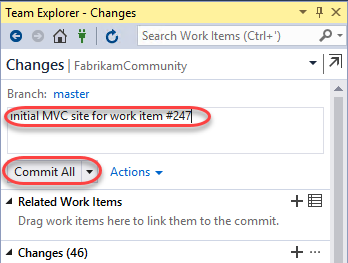
1. Scroll down the list of included changes to the end and note that .gitattributes and .gitignore files were automatically added to the project. The **.gitattributes** file contains various settings to control Git behavior whereas the **.gitignore** file specifies patterns and extensions to ignore when detecting changes.



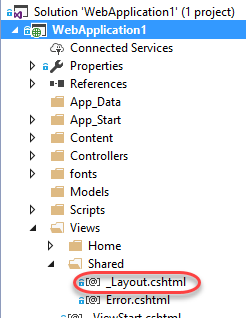
1. Enter a commit message of **“initial MVC site for work item #247”**. Typing **“#”** followed by the work item ID will automatically link the commit to the work item when pushed to the server. If the Product Backlog Item that you saved has a different ID, use that number instead.



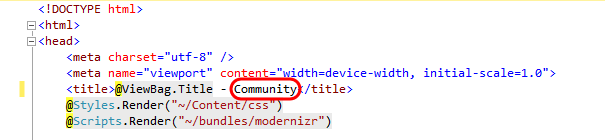
1. Commit the changes by clicking **Commit All**.



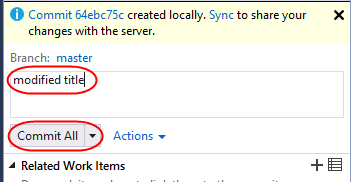
1. Let’s make a small change to the web site. In **Solution Explorer**, open **\_Layout.cshtml** from the **Views\Shared** folder.



1. Modify the end of the page title **“My ASP.NET Application”** to **“Community”** (around line **6**).

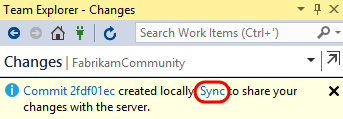


1. In **Team Explorer – Changes**, enter a commit message and then click **Commit All**. **Save** changes to files when prompted.

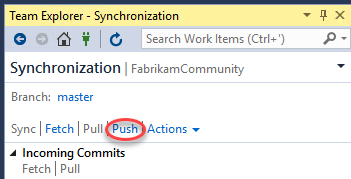


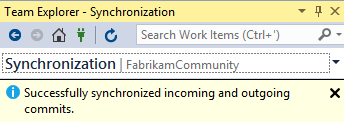
## Task 4: Synchronize Commits with Server

1. Navigate to the commits view by clicking **Sync**.

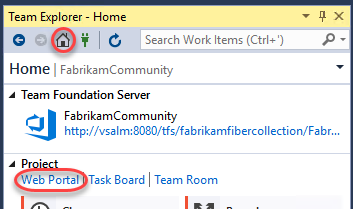


1. The **Team Explorer – Synchronization** view shows both incoming and outgoing commits. However, since this project has not yet been published to source control, you only need to click **push**.

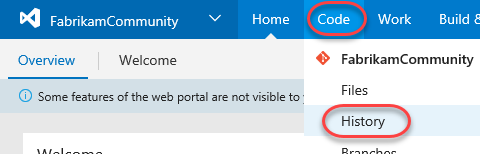




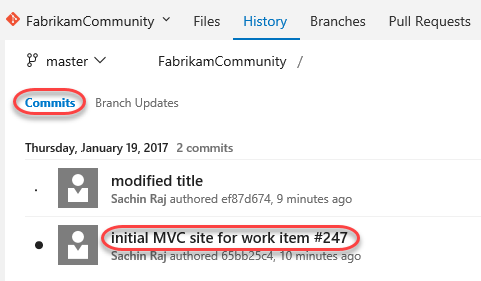
1. Finally, let’s take a quick peek at what these commits look like in the web portal. In **Team Explorer – Home**, click **Web Portal**.



1. From the **Code** dropdown in the web portal, select **History**.

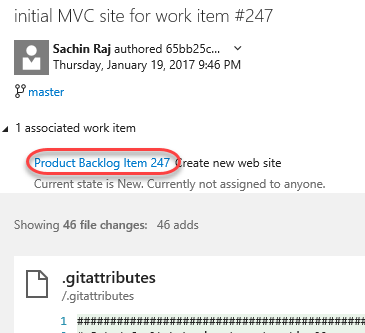


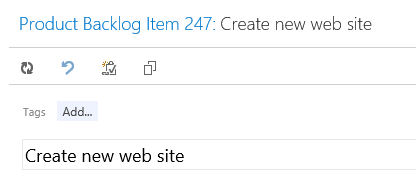
1. Select the **Commits** tab. Note that the relative size of the commits (in terms of number of modified files) can be determined by viewing the size of the circles rendered to the left of the commits. Click the first (bottom) commit.



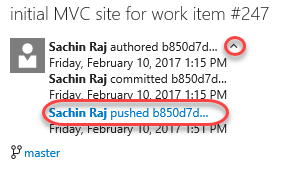
* 1. **Note:** It may take a few moments after pushing a commit before the commit size indicators show up. You can refresh the page if necessary.

1. Note that the **“Create new web site”** work item is linked to the commit. Click the link to open the work item.
   1. **Note:** It may take a few minutes before the work item gets linked to the commit. In the event that the link has not been made yet, go ahead and continue on with the rest of the lab.

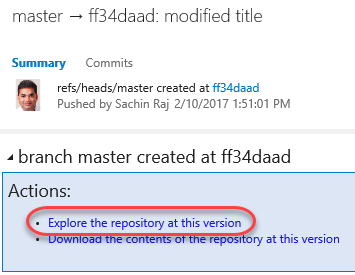




1. Press **Esc** to dismiss the work item dialog.
2. Expand the dropdown next to Sachin’s authored line item and click the **Sachin Raj pushed…** link



1. Although this version happens to be the latest, you would be able to explore or download the version at this particular commit by clicking one of the links.

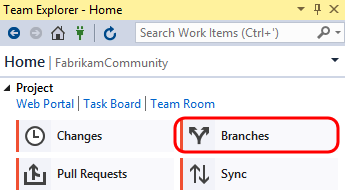


# Exercise 2: Git Branching and Merging

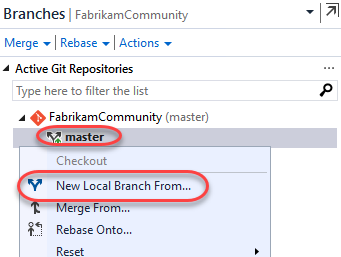
In this exercise, you will learn about Git branching and merging support in Visual Studio. In general, branching is often used to help switch development contexts and to isolate risk. Git branching is no different in that regard. Creating a Git branch is a lightweight (and therefore fast) operation, as you are simply creating a new reference to an existing commit. This is very different from Team Foundation Version Control (TFVC) branching where the entire source tree needs to be duplicated server-side. We will also take a quick look at the merging support for Git projects.

## Task 1: Branching

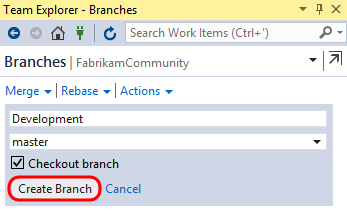
1. Return to Visual Studio and open **Team Explorer – Home**.
2. Click **Branches**.



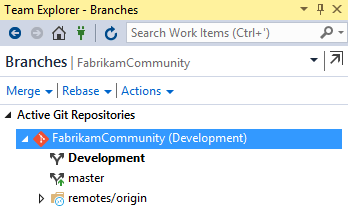
1. Let’s say that we would like to create a new branch to do some development work on the web site. Right-click the **master** branch node and select **New Local Branch From**.



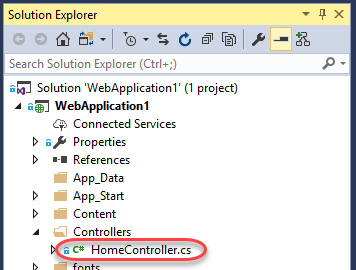
1. Provide a name of **Development**, select the **master** branch, select the **Checkout Branch** option, and then click **Create Branch**.



1. Note that you are now connected to the new local branch, and that it has not been published to the server yet.



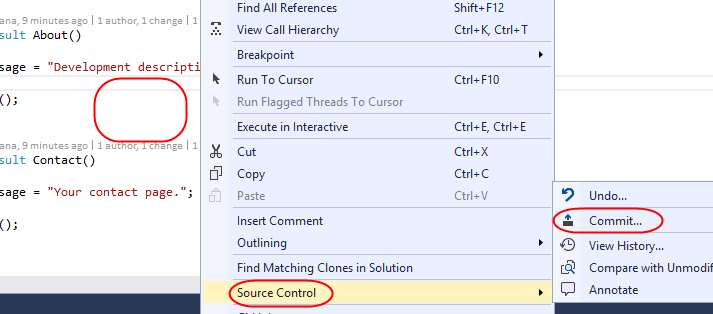
1. In **Solution Explorer**, open the **HomeController.cs** file from the **Controllers** folder.



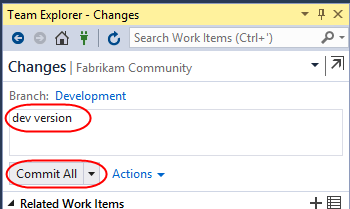
1. Modify the **About** method as shown in the following screenshot.



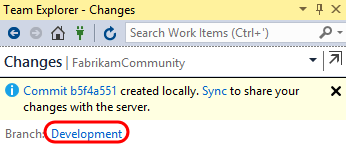
1. Right-click somewhere in the whitespace of the editor and select **Source Control | Commit**.



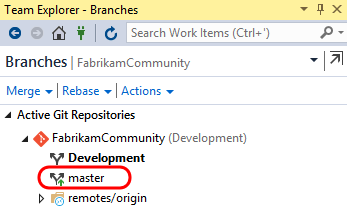
1. In **Team Explorer - Changes**, enter a commit message of “**dev version**” and click **Commit All**. Save the changes when prompted.

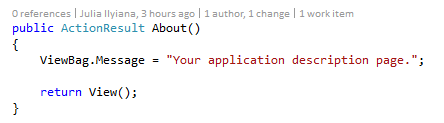


1. At this point, the changes have been committed locally. In the **Team Explorer – Changes** window, click the **Development** branch link to quickly navigate to the **Team Explorer - Branches** view.

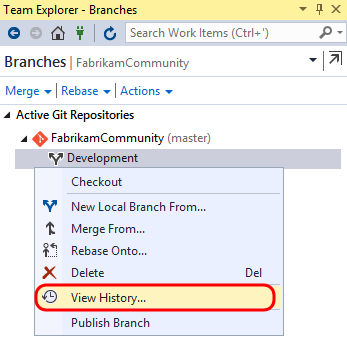


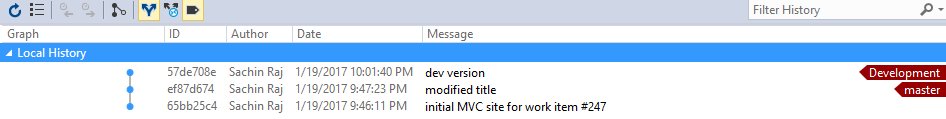
1. Double-click the **Master** branch and note that original version of the **HomeController.cs** file is shown in the code editor window.



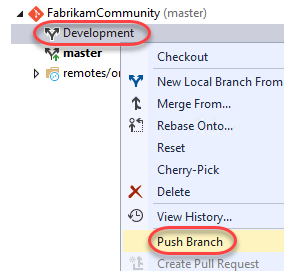


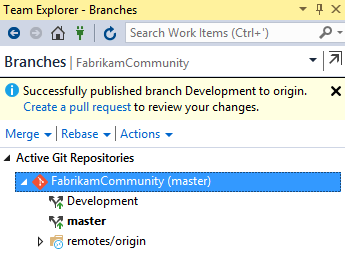
1. You don’t have to publish the branch to the server yet if you want to continue working locally. As you saw in the previous exercise, you can continue to work locally and add additional commits to the new branch. In **Team Explorer – Branches**, **right-click** the **Development** branch and select **View History**.



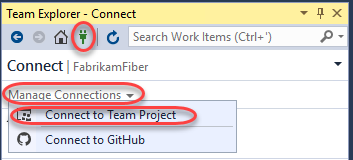


1. When you are ready, you can delete the branch, merge it back into your master branch, or push it to the server-side repository so that teammates can access it. Let’s go ahead and publish the branch by right-clicking the **Development** branch and selecting the **Push Branch** option.

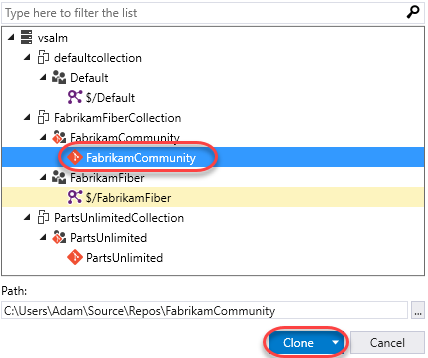




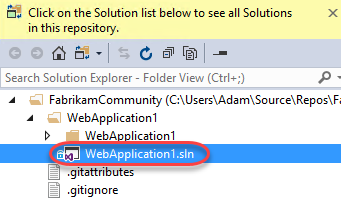
1. Open a **Remote Desktop** session to **VSALM**. Connect using user **Deniz Ercoskun (VSALM\Deniz)**. All user passwords are **P2ssw0rd**. Deniz is about to commit a change to the **master** branch before Sachin has a chance to merge his branch changes.
2. Launch **Visual Studio** from the taskbar.
3. Connect to the **FabrikamCommunity** team project (using the **Manage Connections** button as before, and this time select **Manage Connections | Connect to Team Project**).



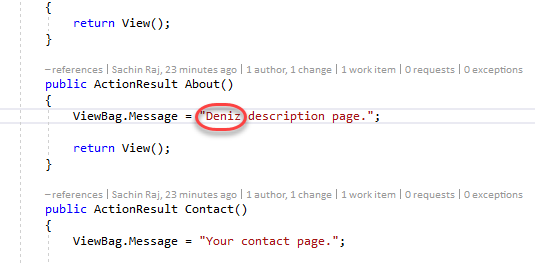
1. Select the **FabrikamCommunity** team project and then click **Clone**.



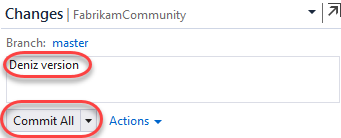
1. **Double-click** the **MvcApplication1.sln** solution shown in **Solution Explorer**.



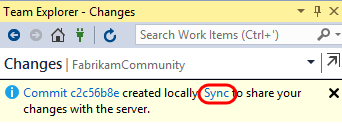
1. Modify the same **HomeController.cs** file that Sachin did, but this time change the text to be something different.



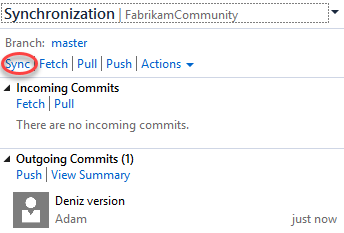
1. As before, right-click within the whitespace of the code editor and select **Source Control | Commit**.
2. In **Team Explorer – Changes**, enter a commit message of “**Deniz version**” and then click **Commit All**. Save changes when prompted. Note that Deniz has committed changes to the master branch.



1. Click **Sync** from the commit response.



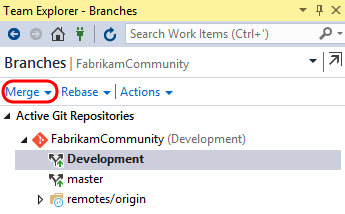
1. Click **Sync** to execute the actual sync.



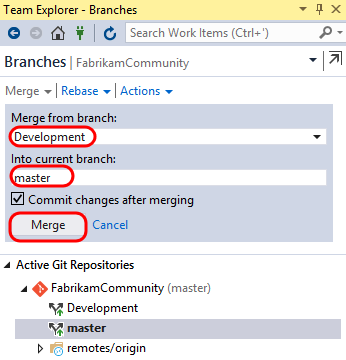
1. Switch users back to **Sachin** by minimizing the remote desktop session.

## Task 2: Merging

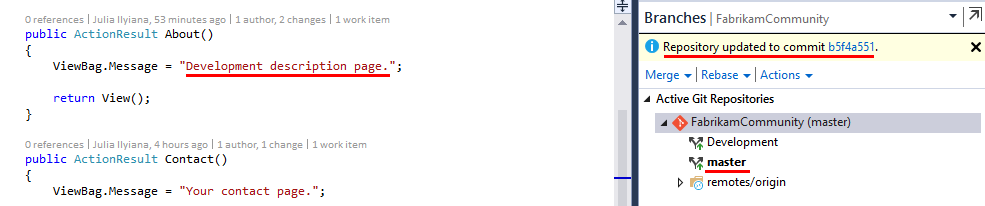
1. From Sachin’s perspective, he has so far created a local branch based off the master, made a change to a file, and then published that branch. He would then like to go ahead and merge his **Development** branch back into the **master** branch.
2. In **Team Explorer – Branches**, select the **Merge** dropdown.



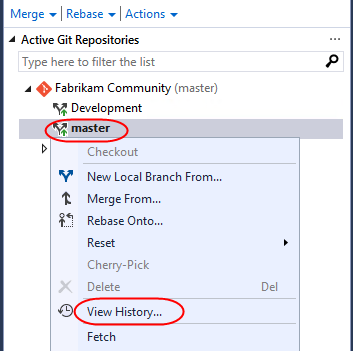
1. Select **Development** as the source and **Master** as the target. Click **Merge** to start the process.



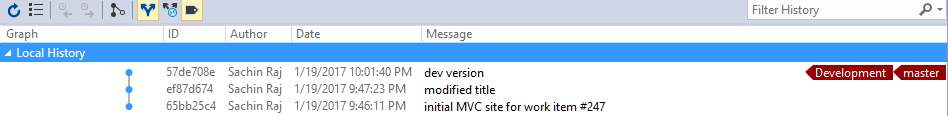
1. Note that the **Master** repository is currently selected and that **HomeController.cs** shows the **Development** version of the text. The merge was performed locally by updating the **Master** branch to point to the latest commit of the **Development** branch.



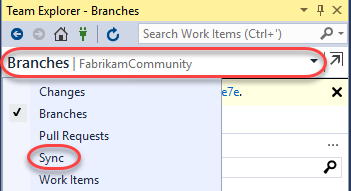
1. **Right-click** the **Master** branch in **Team Explorer – Branches** and select the **View History…** option.



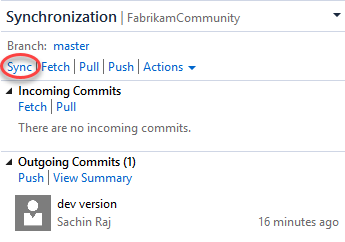
1. The history view should look identical to the one you saw earlier, except this time both the Development and Master branch designators (in red) point to the same commit.



1. Still unaware of Deniz’s change that she pushed to the **Main** branch earlier, Sachin will now attempt to push his commit. Navigate to **Team Explorer – Synchronization** via the Team Explorer dropdown.

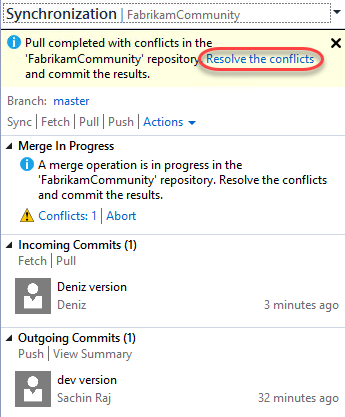


1. Click **Sync** to attempt a pull and a push with the server.

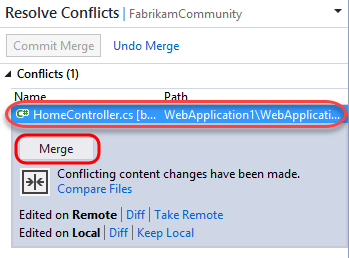


* 1. **Note:** If you see a popup notifying that an open file has been changed externally, click **Yes** as this is expected.

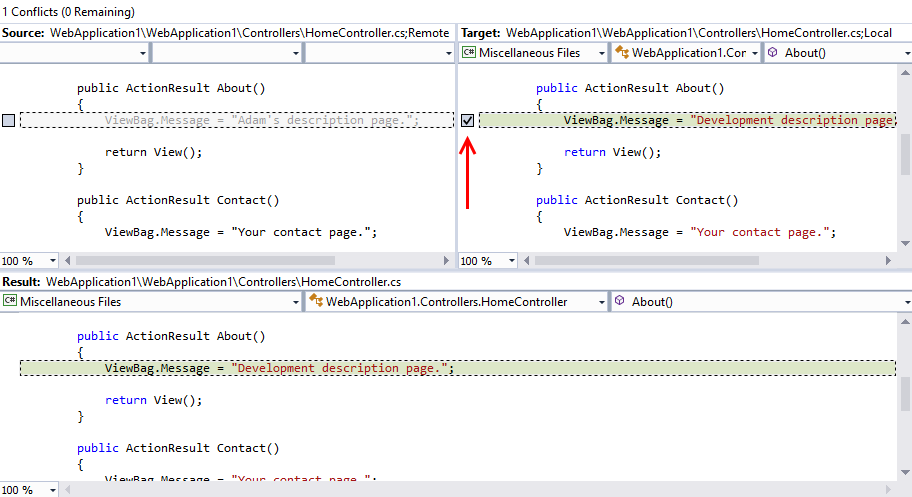
1. Visual Studio reports that we can’t push our commit yet due to a conflict. Click **Resolve the Conflicts**.



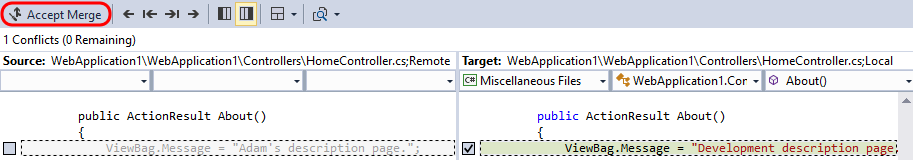
1. In the **Team Explorer – Resolve Conflicts** view, select **HomeController.cs** from the **Conflicts** section and click **Merge**.



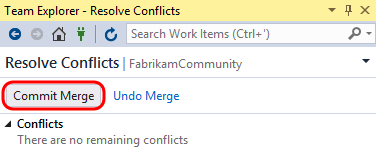
1. The **Merge** window used for Git conflict resolution is very similar to the one used with Team Foundation Version Control. We will go ahead and assume that Sachin’s change is correct, so check the box shown in the top-right pane.



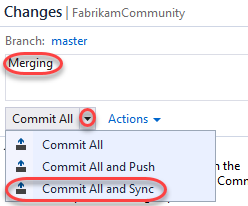
1. Click **Accept Merge**.



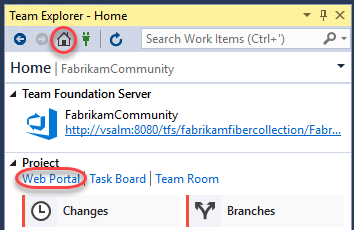
1. Click **Commit Merge**.



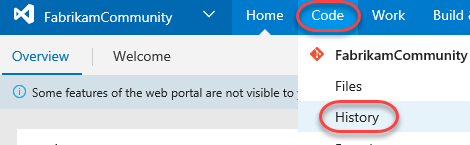
1. In the **Team Explorer – Changes** view, note that conflicts have been resolved but the changes still need to be committed. Enter a message and then click the **Commit All** dropdown and select **Commit All and Sync**.



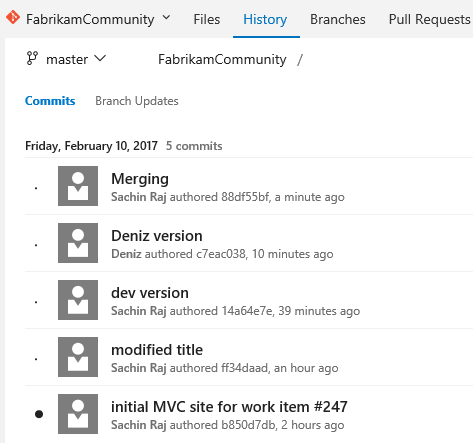
1. Navigate to **Team Explorer – Home** and click **Web Portal**.



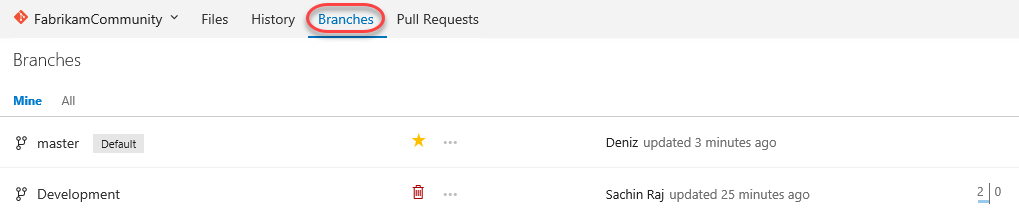
1. Select the **Code | History** from the navigation.



1. Here we can see a full list of our commits so far.

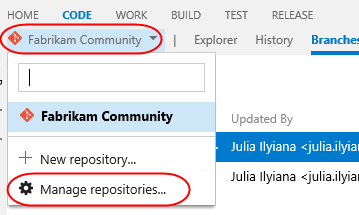


1. Select the **Branches** tab to view all branches published to the repository.

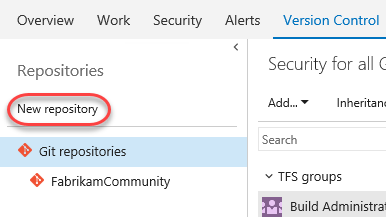


## Task 3: Managing Security and Permissions

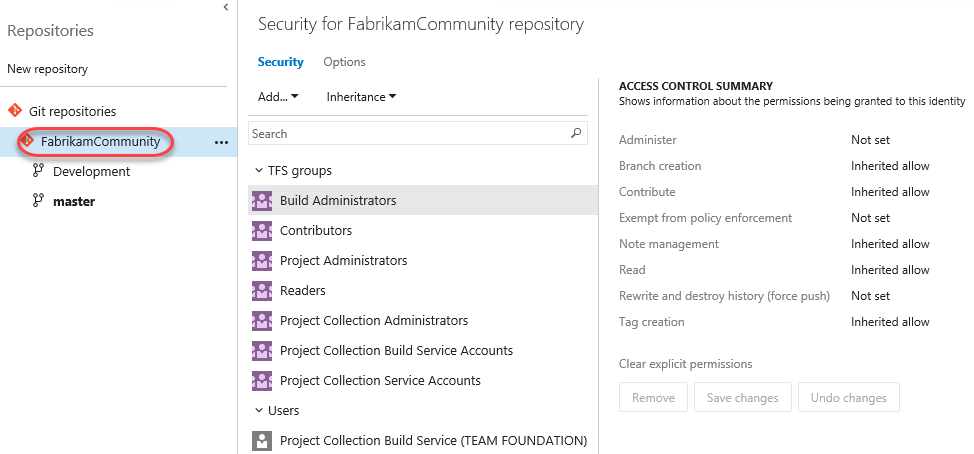
1. Now let’s take a quick peek at managing security and permissions for Git repositories hosted in Team Foundation Server. Select the **FabrikamCommunity** dropdown and then **Manage Repositories**.



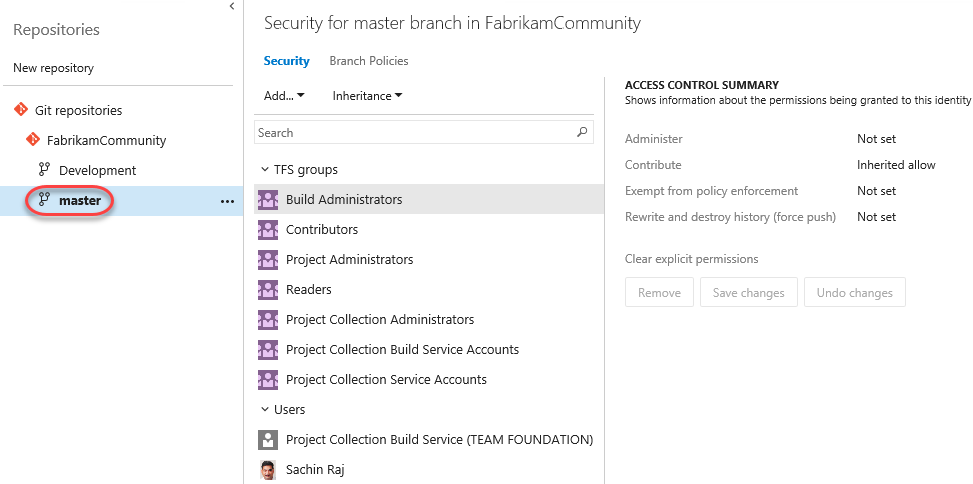
1. The first thing to note is that you can easily create additional Git repositories within the same team project.



1. Select the **FabrikamCommunity** repository node. You can manage repository level security here for your users and security groups.



1. Select the **Master** branch node. Security level settings that affect only the currently selected branch can be made here, providing fine-grained control for your repository if needed.

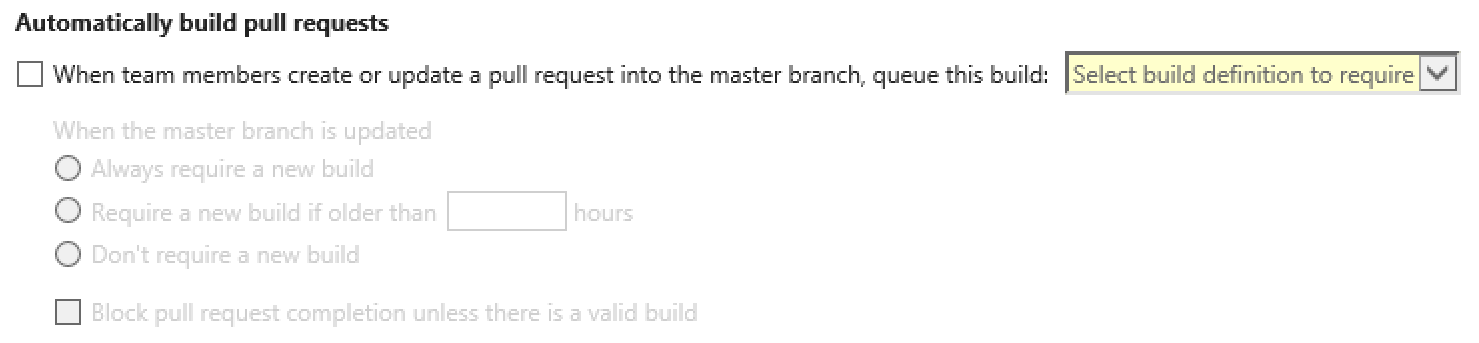


## Task 4: Branch Policies

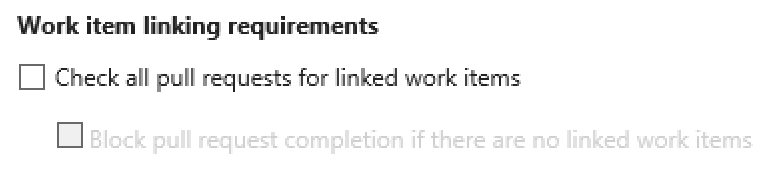
1. When you want people on your team to review code in a Git team project, you can use a pull request to review and merge the code. Pull requests enable developers working in topic branches to get feedback on their changes from other developers prior to submitting the code into the master branch. Any developer participating in the review can see the code changes, leave comments in the code, and give a "thumbs up" approval if they're satisfied with those changes.
2. Although it is possible to utilize pull requests without any further configuration, let’s take a quick look at how to setup branch policies. Select the **Branch Policies** tab.



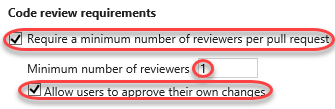
1. You can make use of branch policies that effectively put in a gate that helps prevent inadvertent or low quality commits by automatically initiating a build, or by requiring code reviews by certain individuals. For example, you can use **Automatically build pull requests** to block pull requests if the pull request changes don’t build.



1. You can also require that every pull request has at least one linked work item and block them if they don’t.

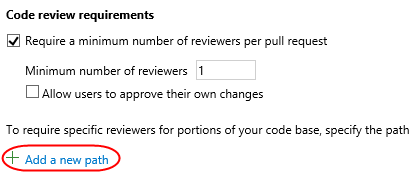


1. Locate the **Code review requirements** section and select the option to **Require…reviewers per pull request** and set the minimum number of reviewers to **“1”**. Also check **allow users to approve their own changes**.



* 1. **Note:** It is also possible to configure branch policy such that a build is triggered whenever updates are made to the master branch. You can even have the merge fail when the build fails. This is useful for teams looking to adopt continuous integration.

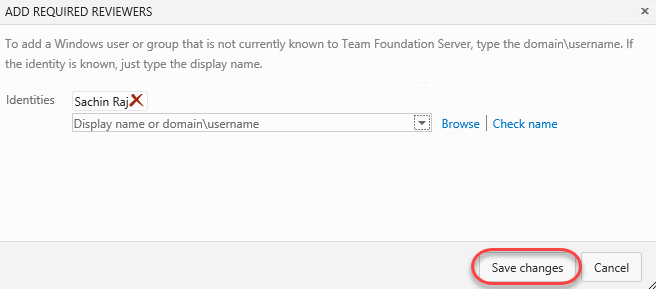
1. It is also possible to require specific reviewers for specific portions of your code base. For example, let’s say that Sachin needs to sign off on all changes made to the ASP.NET MVC controllers. Click **Add a new path**.



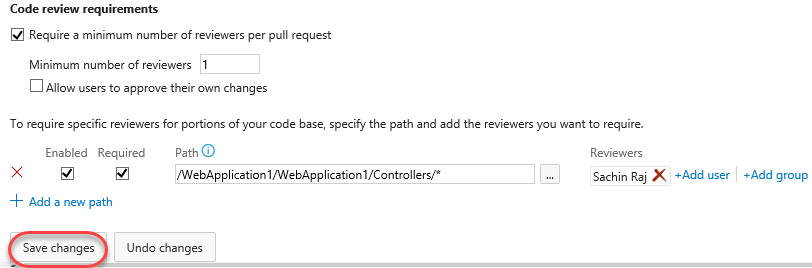
1. Leave the default options of **Enabled** and **Required** selected, set the **Path** to be everything in the **Controllers** folder at **“/WebApplication1/WebApplication1/Controllers/\*”**, and then click **Add User**.



1. Select **Sachin Raj** from the **Add Required Reviewers** window and then click **Save Changes**. Now Sachin must sign off on any pull requests that involve changes to code in this folder.

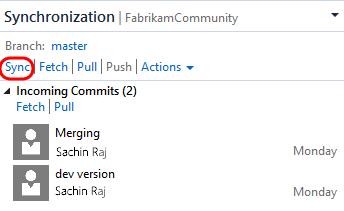


1. Click **Save Changes** to update the master branch policies. Close the administration browser tab.

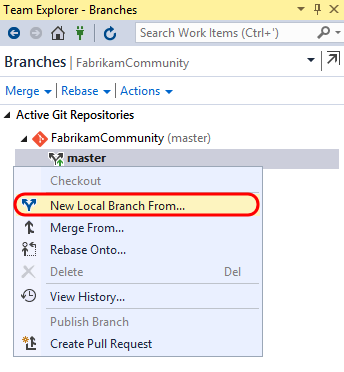


## Task 5: Code Review and Merge using Pull Requests

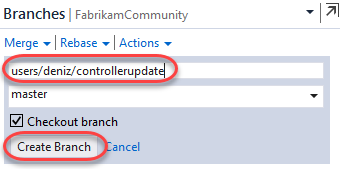
1. Switch back to the **Deniz** RDP session.
2. Click **Sync** to ensure that the local copy of **master** matches what’s on the server. If you have **HomeController.cs** open in the editor, you may be prompted to reload it.



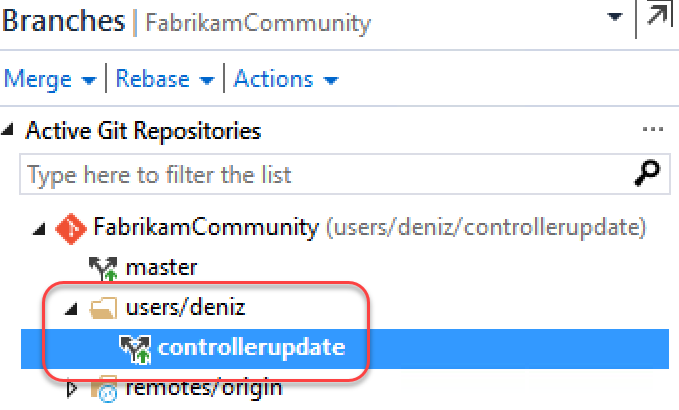
1. Now let’s say that Deniz is working on the project and needs to update some of the controller code. To do this, she will first create a topic branch based on master. Navigate to **Team Explorer - Branches**, right-click the **master** branch, and select **New Local Branch From**…



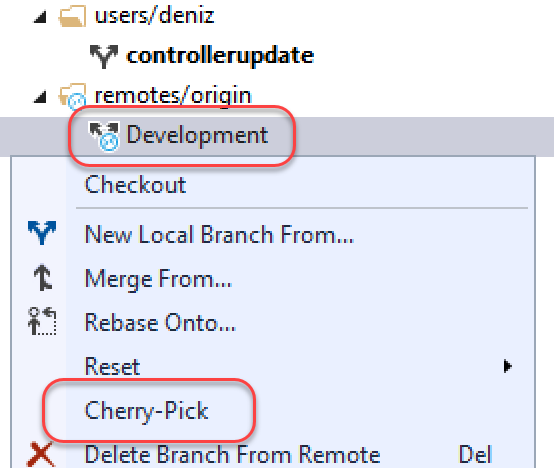
1. For the branch name, use something like **“users/deniz/controllerupdate”.** Use the default option to **Checkout branch**. Click **Create Branch**.



1. The new branch will now appear using the folder structure you specified. This can make dealing with many branches much easier.



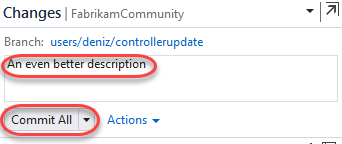
1. Locate the **Development** branch under **remotes/origin** and right-click it to expose some additional branch management options. Note that you have the option to **cherry-pick** commits to apply to the current branch (but don’t do it now). Press **Esc** to dismiss the context menu.



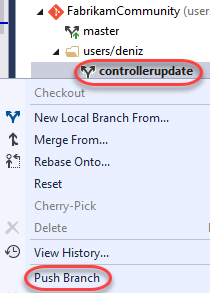
1. Update the **About** method from **HomeController.cs** with a new message, something like **“Deniz’s enhanced description page.”**



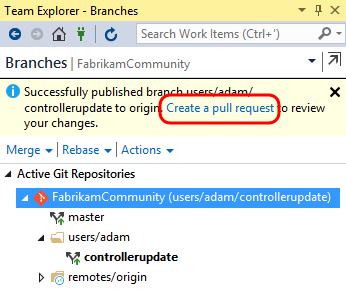
1. In **Team Explorer - Changes**, provide a commit message and then **Commit All**. Save the changes when prompted.



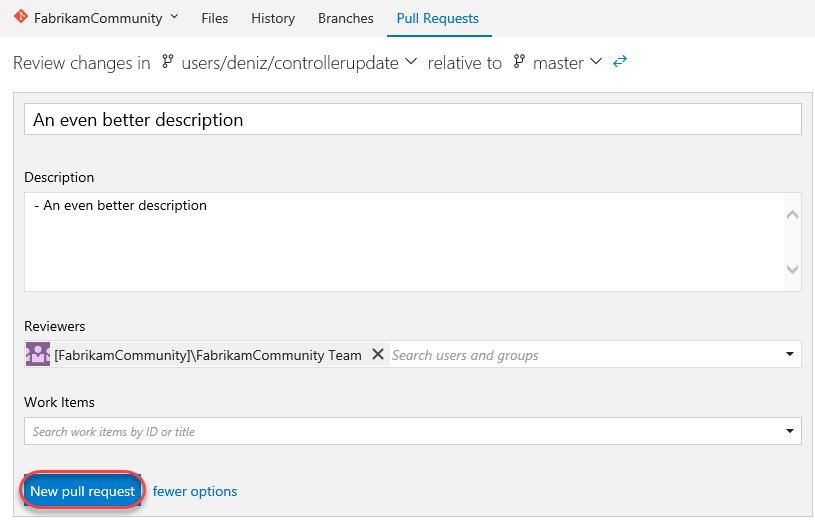
1. In **Team Explorer - Branches**, right-click the topic branch and select **Push Branch**.



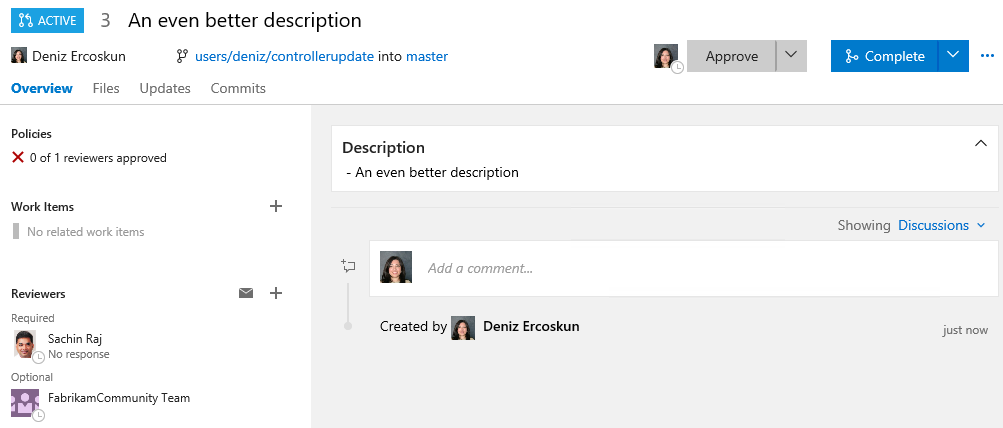
1. After successfully publishing the branch, click **Create a pull request**. This will open the pull request on the portal via your browser.



1. Click **New pull request** to create a pull request with the default options. Note that you could optionally add additional users to review or associate work items if desired.



1. After the pull request is created, the pull request view shows what merge is proposed, the provided description, and a set of tabs listing associated files and commits. It also indicates that there are no merge conflicts detected, and that due to branch policy, **Sachin** must approve the changes in order to proceed.



1. To demonstrate the branch policy in action, let’s say Deniz attempts to complete the pull request by herself. Click **Complete** and then **Complete Merge**.



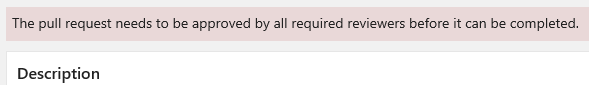
1. Note that Deniz is notified that the request first needs to be approved by at least one reviewer first.



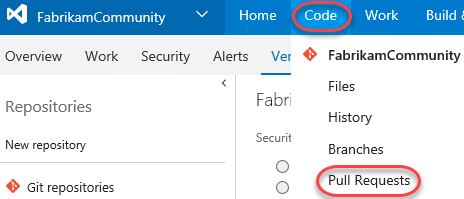
1. Click **Approve**, followed by **Complete**, and then **Complete Merge** in the dialog.



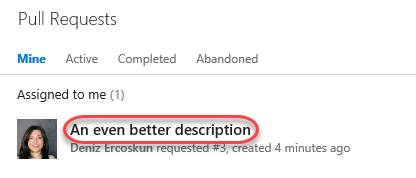
1. Note that although we’ve fulfilled the reviewer quantity, we didn’t fulfill the requirement for **Sachin** to approve any changes to the controllers.



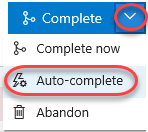
1. At this point in the workflow, **Sachin** needs to be notified of this pull request through some communication channel, whether that is in person, through Skype, through team room notification, or via TFS pull request alert. For the purposes of this short exercise, however, we will just skip to Sachin checking pull requests for this project.
2. Switch users back to **Sachin** by minimizing the RDP window.
3. In the web portal, navigate to the **FabrikamCommunity** project and select **Code | Pull Requests** view.



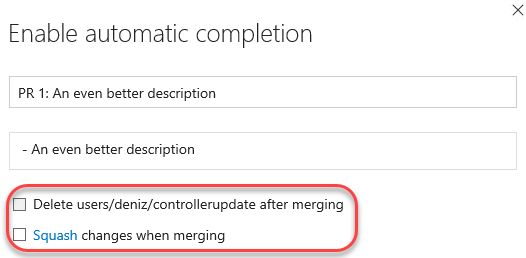
1. Select the link provided on the pull request from Deniz.



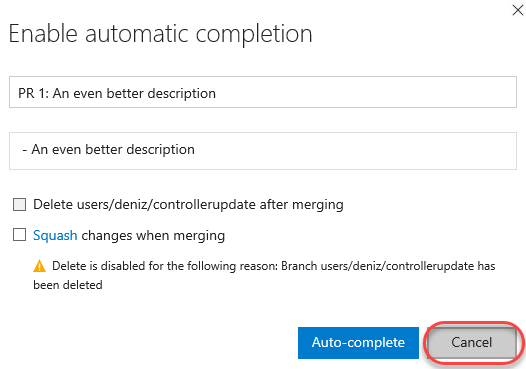
1. Sachin can now review all of the files and commits associated with the pull request and make a decision. It is also possible for Sachin to have a conversation with Deniz (and perhaps other reviewers) in order to help make the decision, or perhaps even request additional work be performed before the pull request will be approved.
2. Let’s assume that Sachin is ready to approve the request as-is. There are a couple ways to do this. Select **Complete | Auto-complete**.



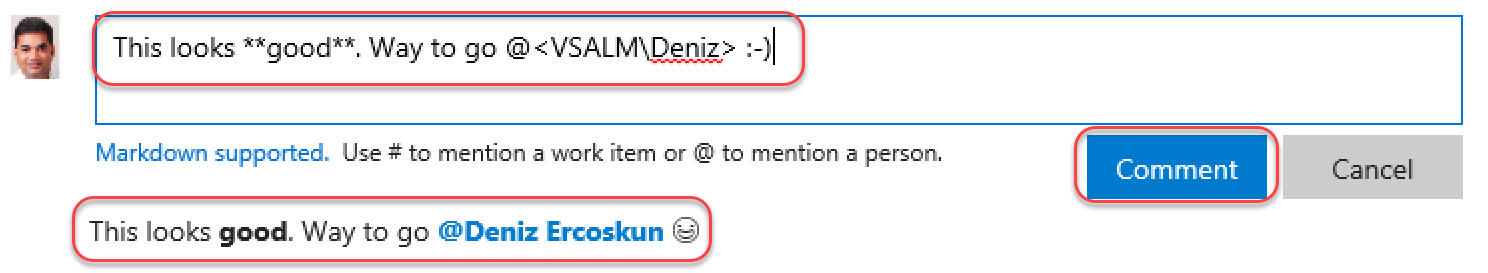
1. The automatic completion feature enables you to approve pull request for automatic completion before all policies have been applied. For example, if a successful build is required before the PR can be completed, this step enables that to happen automatically after all policies have succeeded. You can also specify whether to delete the branch after merging and/or if you want to **squash changes**. Selecting the option to squash changes will merge all commits from the target branch into a single commit in order to help keep things a little tidier.



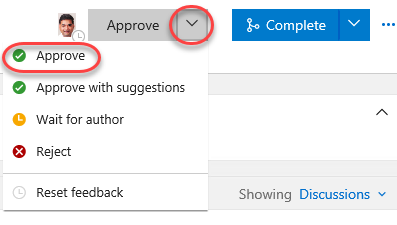
1. In this case, we won’t use automatic completion, so click **Cancel**.



1. Before approving, provide some positive feedback to **Deniz**, such as **“This looks \*\*good\*\*. Way to go @<VSALM\Deniz> :-)”** and click **Comment**. Note that you can use markdown and emojis, as well as @ reference other users. You can also use # references for work items, such as “#47”. As you type, there is a live preview below so that you can see exactly what will show up.



1. Select **Approve | Approve**.



1. Now that all policies have been fulfilled, click **Complete** and **Complete merge** to finish the process.



1. Switch users back to **Deniz**.
2. Refresh the pull request view and note that it has been updated as expected with the actions made by Sachin.

