

# Contents

Start using Azure Repos

What is Azure Repos?

Sign up and invite some teammates

Code with Git

Default Git permissions (Security)

Key concepts

Resources

Visual Studio IDE

Visual Studio Code

Visual Studio for Mac

Manage projects

# Start using Azure Repos

1/25/2019 • 2 minutes to read • [Edit Online](#)

Use this guide to sign up and start using Azure Repos. If you're new to Azure Repos, see [What is Azure Repos?](#).

Azure DevOps Services includes free unlimited private Git repos, so Azure Repos is easy to try out. Git is the most commonly used version control system today and is quickly becoming the standard for version control. Git is a distributed version control system, meaning that your local copy of code is a complete version control repository. These fully functional local repositories make it easy to work offline or remotely. You commit your work locally, and then sync your copy of the repository with the copy on the server.

Git in Azure Repos is standard Git. You can use the clients and tools of your choice, such as Git for Windows, Mac, partners' Git services, and tools such as Visual Studio and Visual Studio Code.

Start with [Sign up and invite some teammates](#).

After you sign up, learn how to [code with Git](#). Whether your code is in Azure Repos, in a GitHub repo, or on your local computer, this guide shows you how to get started.

Other resources to get you up and running:

- [Web portal navigation](#)
- [Security & identity](#)
- [About projects and scaling your organization](#)

Azure Repos is a set of version control tools that you can use to manage your code. Whether your software project is large or small, using version control as soon as possible is a good idea.

Version control systems are software that help you track changes you make in your code over time. As you edit your code, you tell the version control system to take a snapshot of your files. The version control system saves that snapshot permanently so you can recall it later if you need it. Use version control to save your work and coordinate code changes across your team.

Even if you're just a single developer, version control helps you stay organized as you fix bugs and develop new features. Version control keeps a history of your development so that you can review and even roll back to any version of your code with ease.

Azure Repos provides two types of version control:

- [Git](#): distributed version control
- [Team Foundation Version Control \(TFVC\)](#): centralized version control

## Git

Git is the most commonly used version control system today and is quickly becoming the standard for version control. Git is a distributed version control system, meaning that your local copy of code is a complete version control repository. These fully functional local repositories make it easy to work offline or remotely. You commit your work locally, and then sync your copy of the repository with the copy on the server.

Git in Azure Repos is standard Git. You can use the clients and tools of your choice, such as Git for Windows, Mac, partners' Git services, and tools such as Visual Studio and Visual Studio Code.

- [Connect your favorite development environment](#)
- [Review code with pull requests](#)
- [Protect branches with policies](#)
- [Extend pull request workflows with pull request status](#)
- [Isolate code with forks](#)

### Connect your favorite development environment

Connect your favorite development environment to Azure Repos to access your repos and manage your work.

[Command-line](#)

[Visual Studio Code](#)

[Visual Studio](#)

[Xcode](#)

[Eclipse](#)



### Review code with pull requests

Review code with your team and make sure that changes build and pass tests before it's merged.

[Create a pull request](#)

[Link work items to pull requests](#)

[Set up branch policies](#)

## Squash merge pull requests

## Git branch and pull request workflows

### Leave comments



### Vote on the changes

### Protect branches with policies

There are a few critical branches in your repo that the team relies on to always be in good shape, such as your `master` branch. [Require pull requests](#) to make any changes on these branches. Developers who push changes directly to the protected branches will have their pushes rejected.

Add conditions to your pull requests to enforce a higher level of code quality in your key branches. A clean build of the merged code and approval from multiple reviewers are extra requirements that you can set to help protect your key branches.

## Branch policies overview

## How to configure branch policies

### Branch permissions

### Extend pull request workflows with pull request status

Pull requests and branch policies enable teams to enforce many best practices related to reviewing code and running automated builds. But many teams have additional requirements and validations to perform on code. To cover these individual and custom needs, Azure Repos offers pull request statuses.

Pull request statuses integrate into the PR workflow. They allow external services to programmatically sign off on a code change by associating simple success/failure information with a pull request.

## Pull request status overview

### Create a PR status server with Node.js



### Use Azure Functions to create custom branch policies

### Configure a branch policy for an external service

### Isolate code with forks

Forks are a great way to isolate experimental, risky, or confidential changes from the original codebase. A fork is a complete copy

of a repository, including all files, commits, and (optionally) branches. The new fork acts as if someone cloned the original repository and then pushed to a new, empty repository.

After a fork has been created, new files, folders, and branches are not shared between the repositories unless a pull request carries them along. After you're ready to share those changes, it's easy to use [pull requests](#) to push the changes back to the original repository.

[Learn more about forks](#)

## TFVC

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

Get started by creating a project, configuring your workspace, and reviewing and sharing your code. You can use any one of these clients or IDEs:

[Visual Studio](#)

[Xcode](#)

[Eclipse](#)

[Learn more about TFVC](#)

# Sign up for free and invite others to collaborate on your project

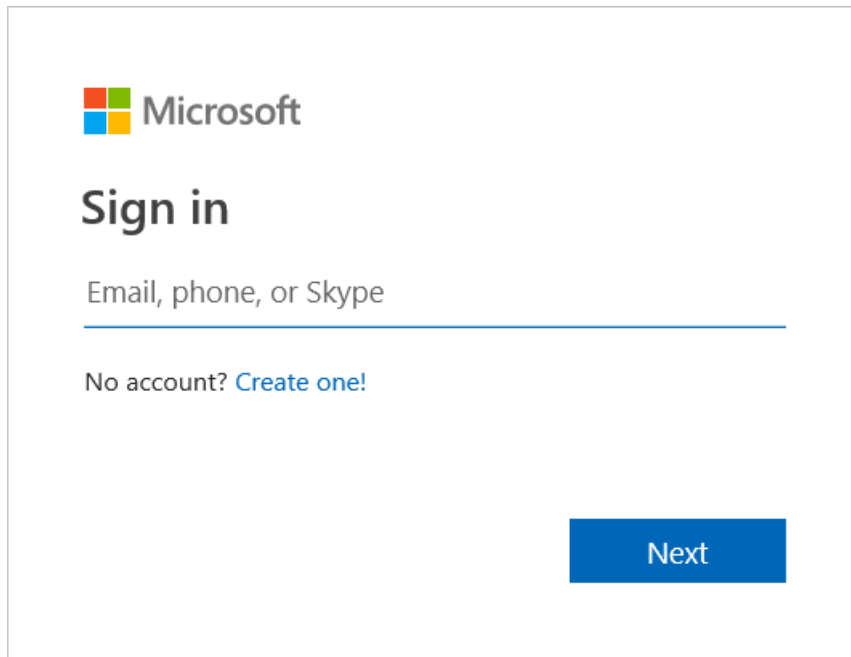
1/31/2019 • 2 minutes to read • [Edit Online](#)

Sign up for Azure DevOps to upload and share code in free, unlimited private Git repositories.

Then, connect to your favorite development tool like Eclipse, Xcode, Visual Studio, IntelliJ, or Android Studio to work on apps anytime, anywhere.

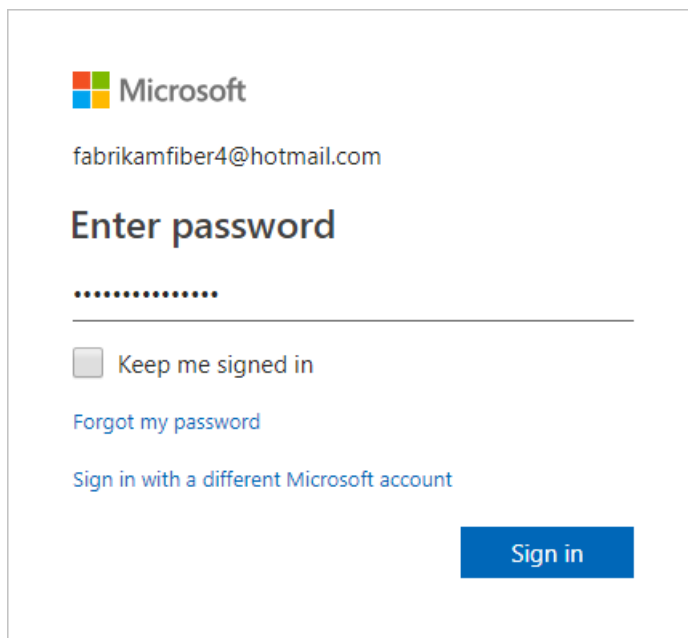
## Sign up for Azure DevOps with a personal Microsoft account

1. Select the sign-up link for [Azure DevOps](#).
2. Enter your email address, phone number, or Skype ID for your Microsoft account. If you're a Visual Studio subscriber and you get Azure DevOps as a benefit, use the Microsoft account associated with your subscription. Select **Next**.

A screenshot of the Microsoft sign-in page. At the top left is the Microsoft logo. Below it, the text "Sign in" is displayed in a large, bold font. Underneath "Sign in" is a text input field with the placeholder text "Email, phone, or Skype". Below the input field is a link that says "No account? [Create one!](#)". At the bottom right of the form is a blue button with the text "Next" in white.

3. Enter your password and select **Sign in**.

If you don't have a Microsoft account, you can [create a Microsoft account](#) at this time.



Microsoft

fabrikamfiber4@hotmail.com

## Enter password

.....

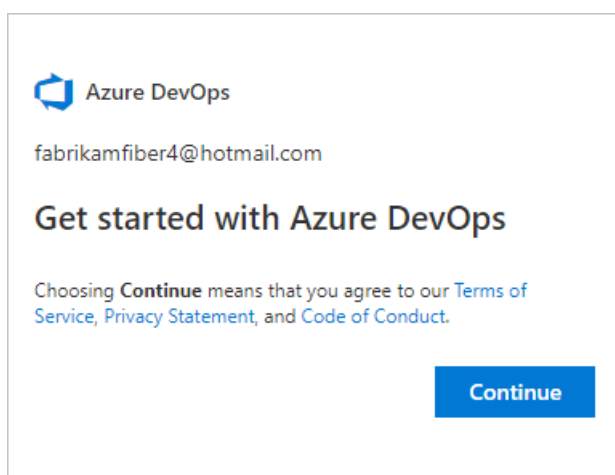
☐ Keep me signed in

[Forgot my password](#)

[Sign in with a different Microsoft account](#)

**Sign in**

4. To get started with Azure DevOps, select **Continue**.



Azure DevOps

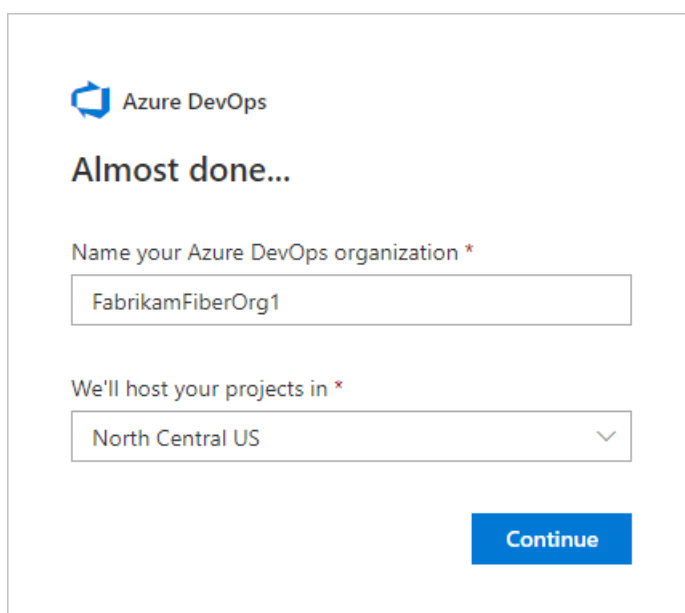
fabrikamfiber4@hotmail.com

## Get started with Azure DevOps

Choosing **Continue** means that you agree to our [Terms of Service](#), [Privacy Statement](#), and [Code of Conduct](#).

**Continue**

5. Enter a name for your organization. The name can't contain spaces or special characters (such as / \ [ ] : | < > + = ; ? or \*), can't end in a period or comma, must be less than 256 characters, and must be unique within the DevOps namespace. You can also choose between several locations for where you want your data hosted. Select **Continue**.



Azure DevOps

## Almost done...

Name your Azure DevOps organization \*

FabrikamFiberOrg1

We'll host your projects in \*

North Central US

**Continue**

You see the following dialog box as your organization is created.

Congratulations, you're now an organization owner!

To sign in to your organization at any time, go to `https://dev.azure.com/{yourorganization}`.

6. Enter a name for your project and select the visibility. The name can't contain spaces or special characters (such as `/ : \ ~ & % ; @ ' " ? < > | # $ * } { , + = [ ]`), can't begin with an underscore, can't begin or end with a period, and must be 64 characters or less. Visibility can be either public or private. With public visibility, anyone on the internet can view your project. With private visibility, only people who you give access to can view your project. Select **Create project**.

## Invite team members

Give a team member access to your organization by adding their email address to your organization.

1. Sign in to your organization (`https://dev.azure.com/{yourorganization}`).

2. Select  **Organization settings**.

3. Select **Users** > **Add new users**.

4. Complete the form by entering or selecting the following information:

- **Users:** Enter the email addresses (Microsoft account) for the users. You can add several email addresses by separating them with a semicolon (;). An email address appears in red when it's accepted.
- **Access level:** Leave the access level as **Basic** for users who will contribute to the code base. To learn more, see [About access levels](#).
- **Add to project:** Select the project you named in the preceding procedure.
- **DevOps Groups:** Leave as **Project Contributors**, the default security group for users who will contribute to your project. To learn more, see [Default permissions and access assignments](#).

### NOTE

You must add email addresses for [personal Microsoft accounts](#) unless you plan to use [Azure Active Directory \(Azure AD\)](#) to authenticate users and control organization access. If a user doesn't have a Microsoft account, ask the user to [sign up](#) for a Microsoft account.

5. When you're done, select **Add** to complete your invitation.



# Code with Git

1/31/2019 • 10 minutes to read • [Edit Online](#)

After you create a new organization and team project in Azure DevOps Services, you can begin sharing your code with others.

In this article, we'll show you how to start working with your code in Azure Repos with a simple walkthrough that covers:

- Installing Git command line tools
- Cloning a Git repository
- Working in a branch
- Sharing your changes
- Creating a pull request

## Install Git command-line tools

Install one of the following Git command-line tools:

- To install Git for Windows, including Git Credential Manager, see [Install Git Credential Manager for Windows](#).
- To install Git for macOS and Linux, see [Install Git Credential Manager for macOS and Linux](#).

## Get your code

To get a copy of the source code, you clone the Git repo that contains the code. Cloning creates both a local copy of the source code so you can work with it, and all the version control information so Git can manage the source code.

If you're just getting started with Azure Repos, your code might be in one of several places:

- [I just created my organization in Azure DevOps, so I don't have any code](#)
- [The code is in my \(or my organization's\) Azure Repos Git repo](#)
- [The code is in another Git repo such as GitHub or another Azure Repos Git repo](#)
- [The code is on my local computer and not yet in version control](#)

### **I just created my organization in Azure DevOps, so I don't have any code**

If you just signed up for Azure DevOps Services, by default you have a project named `MyFirstProject` and a Git repo named `MyFirstProject`. If you want to work in that repo, you can [clone it](#) and then add your code to that repo.

If you want to make a new repo, follow the steps in [Create a new Git repo in your project](#) and then [clone](#) the new repo and add your code there.

### **The code is in my (or my organization's) Azure Repos Git repo**

If the code is in your (or your organization's) Azure Repo, you can clone the Git repo to your local computer and start working with it by jumping down to [Clone the repo](#).

### **The code is in another Git repo**

If the code is in another Git repo, such as a GitHub repo or a different Azure Repo instance, you can import it into a new or existing empty Git repo. Follow the steps in [Import a Git repo](#), and then return to this article and jump down to [Clone the repo](#).

### **The code is on my local computer and not yet in version control**

If your code is not yet in version control, you have a couple of options:

- Create a new repository and add your code there. To do this, follow the steps in [Create a new Git repo in your project](#) and then come back to this article and jump down to [Clone the repo](#).
- Add your code to an existing repository. To do this, jump down to [Clone the repo](#).

After the repository is cloned, we'll show you how to add your existing code to the repo.

## Clone the repo

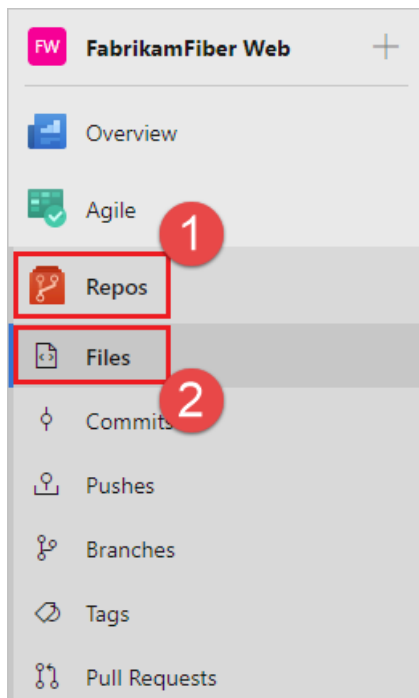
To work with a Git repo, you clone it to your computer. Cloning a repo creates a complete local copy of the repo for you to work with. Cloning also downloads all [commits](#) and [branches](#) in the repo and sets up a named relationship with the repo on the server. Use this relationship to interact with the existing repo, pushing and pulling changes to share code with your team.

### NOTE

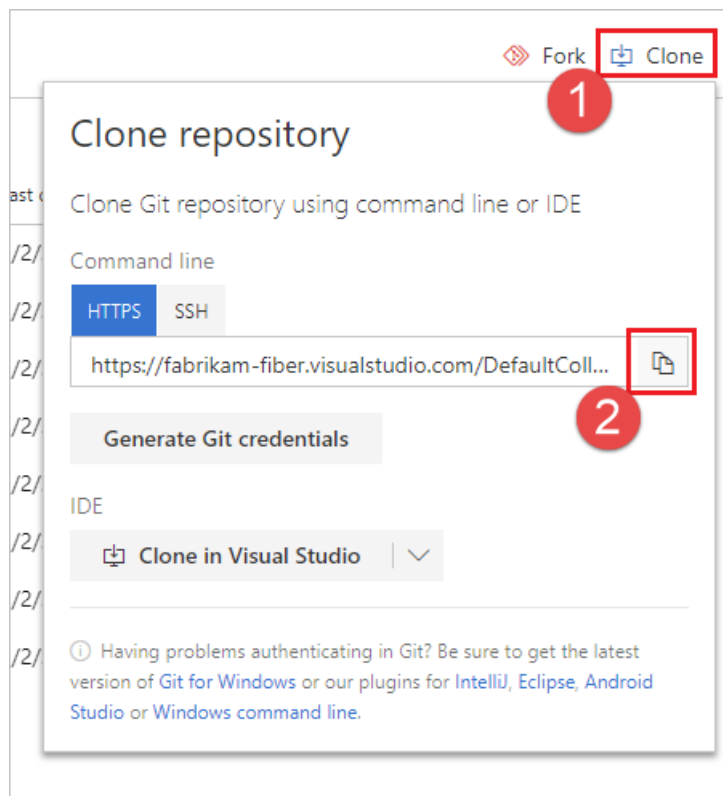
Your web portal uses either the **New navigation** or **Previous navigation** user interface. Choose the **New navigation** tab if the **New Navigation** feature is enabled. You'll see a vertical sidebar along with other navigational features when **New Navigation** has been enabled for the signed-in user or the organization. Choose **Previous navigation** when you see a top-level, blue-bar—indicating that **New navigation** isn't enabled. For more information, see [Web portal navigation](#).

- [New navigation](#)
- [Previous navigation](#)

1. From your web browser, open the team project for your organization and select **Repos** > **Files**. If you don't have a team project, [create one now](#).



2. Select **Clone** in the upper-right corner of the **Files** window and copy the clone URL.



3. Open the Git command window (Git Bash on Git for Windows) and browse to the folder where you want the code from the repo stored on your computer. Run `git clone` followed by the path copied from the **Clone URL** in the previous section, as shown in the following example.

```
git clone https://dev.azure.com/contoso-ltd/MyFirstProject/_git/contoso-demo
```

Git downloads a copy of the code, including all [commits](#) and [branches](#) from the repo, into a new folder for you to work with.

4. Switch your directory to the repository that you just cloned.

```
cd fabrikam-web
```

Keep this command window open, because you'll use it in the following steps.

## Work in a branch

Git [branches](#) isolate your changes from other work being done in the project. The recommended [Git workflow](#) uses a new branch for every feature or fix that you work on.

Create branches by using the `branch` command. This command creates a reference in Git for the new branch and a pointer back to the parent commit so Git can keep a history of changes as you add commits to the branch.

Git always adds new commits to the current local branch. Check what branch you're working on before you commit so that you don't commit changes to the wrong branch.

Switch between local branches by using the `checkout` command. Git will change the files on your computer to match the latest commit on the checked-out branch.

In this step, we'll create a working branch and make a change to the files on your computer in that branch.

Use the `branch` command to create the branch and `checkout` to switch to that branch. In the following example, the new branch is named `users/jamal/feature1`.

```
git branch users/jamal/feature1
git checkout users/jamal/feature1
```

When you create a branch from the command line, the branch is based on the currently checked-out branch. If you just cloned the repository, the default branch (typically `master`) is checked out. Because you just cloned, your local copy of `master` has the latest changes.

If you're working with a previously cloned repository, ensure that you have checked out the right branch (`git checkout master`) and that it's up to date (`git pull origin master`) before you create your new branch.

```
git checkout master
git pull origin master
git branch users/jamal/feature1
git checkout users/jamal/feature1
```

You can replace the first three commands in the previous example with the following command, which creates a new branch named `users/jamal/feature1` based on the latest `master` branch.

```
git pull origin master:users/jamal/feature1
```

Switch back to the Git Bash window that you used in the previous section. Run the following commands to create and check out a new branch based on the master branch.

```
git pull origin master:users/jamal/feature1
git checkout feature1
```

Browse to the location of the repository on your local computer, make an edit to one of the files, and save it. If you're adding code from your local computer to the repository, you can add it here by copying it to the folder where you cloned the repository.

## Share your changes

When you're happy with the changes on your local computer, you can share them back to the remote repository.

1. Commit your changes by entering the following command in the Git command window:

```
git add .
git commit -m "My first commit"
```

The `git add .` command stages your files, and `git commit -m "My first commit"` commits the staged files with the specified commit message.

2. Push your changes to the Git repo on the server by entering the following command in the Git command window:

```
git push origin users/jamal/feature1
```

Your code is now shared to the remote repository, in a branch named `users/jamal/feature1`. To merge the code from your working branch into the `master` branch, use a pull request.

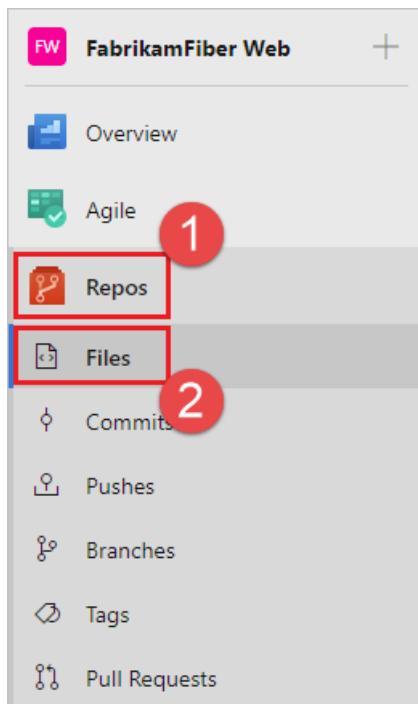
## Review and merge your changes with a pull request

Pull requests combine the review and merge of your code into a single collaborative process. After you're done fixing a bug or new feature in a branch, create a new pull request. Add the members of the team to the pull request so they can review and vote on your changes. Use pull requests to review works in progress and get early feedback on changes. There's no commitment to merge the changes because you can abandon the pull request at any time.

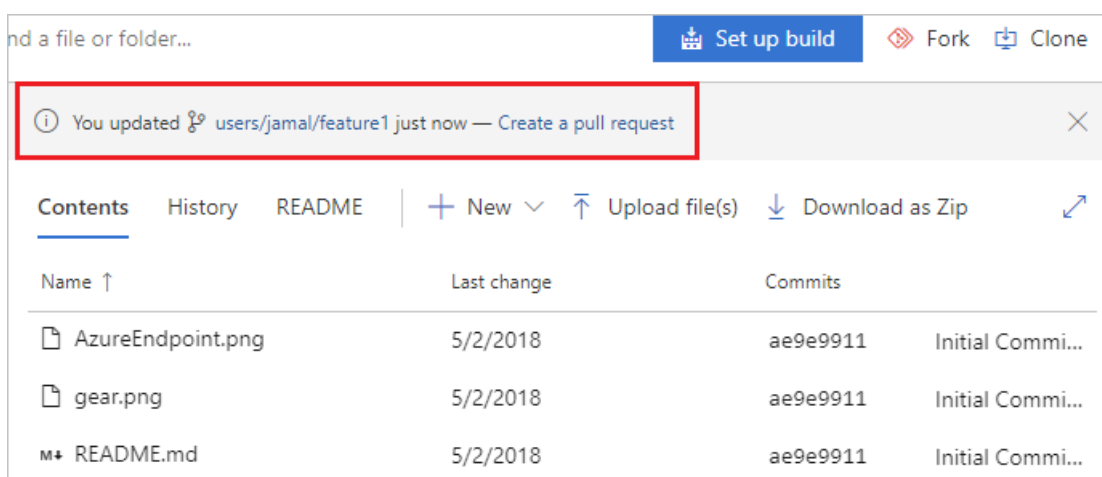
This example shows the basic steps of creating and completing a pull request.

- [New navigation](#)
- [Previous navigation](#)

1. From your web browser, open the team project for your organization and select **Repos** > **Files**. If you kept your browser open after getting the clone URL, you can just switch back to it.



2. Select **Create a pull request** in the upper-right corner of the **Files** window. If you don't see a message like **You updated users/jamal/feature1 just now**, refresh your browser.



3. New pull requests are configured to merge your branch into the default branch, which in this example is `master`. The title and description are pre-populated with your commit message.

## 🔗 New Pull Request

🔗 users/jamal/feature1 ▾ into 🔗 master ▾ ↔

Title \*

My first commit

Add label

Description

My first commit

*Markdown supported.*

Aa ▾ **B** *I* 🔗 </> ☰ ☷ ☹ @ # 🔗

My first commit

Reviewers

Search users and groups to add as reviewers

Work Items

Search work items by ID or title ▾

Create

You can [add reviewers](#) and [link work items](#) to your pull request.

You can review the files included in the pull request at the bottom of the **New Pull Request** window.

Create

Files (1) Commits (1)

Showing 1 file change: 1 edit

M README.md +1 -1  
/README.md

```

...  ...
13 13
14 14 -----
15 15
16 16 - Please follow below exercises inorder to deploy your application, :
16 16 + Please follow below exercises inorder to deploy your application:
17 17
18 18 ## Exercise 1: Endpoint Creation
19 19
...  ...

```

Select **Create** to create the pull request.

- You can view the details of your pull request from the **Overview** tab, and view the changed files, updates, and commits in your pull request from the other tabs. Select **Complete** to begin the process of completing the pull request.

5 ACTIVE My first commit

Jamal Hartnett users/jam... into master

Approve Complete

Overview Files Updates Commits

Description

My first commit

Show everything

Add a comment...

Created by Jamal Hartnett just now

Work Items

No related work items

Reviewers

No reviewers

Labels

Add label

5. Select **Complete merge** to complete the pull request and merge your code into the `master` branch.

Complete pull request

Merged PR 5: My first commit

My first commit

☐ Complete linked work items after merging

☒ Delete users/jamal/feature1 after merging

☐ Squash changes when merging [Learn more](#)

Complete merge Cancel

#### NOTE

This example shows the basic steps of creating and completing a pull request. To learn more about pull requests, including voting and reviewing, commenting, autocomplete, and more, see [Pull requests overview](#).

Your changes are now merged into the `master` branch, and your `users/jamal/feature1` branch is deleted on the remote repository. To delete your local copy of the branch, switch back to your Git Bash command prompt and run the following commands.

```
git checkout master
git pull origin master
git branch -d users/jamal/feature1
```

The `git checkout master` command switches you to the `master` branch. The `git pull origin master` command pulls down the latest version of the code in the master branch, including your changes and the fact that `users/jamal/feature1` was merged. The `git branch -d users/jamal/feature1` command deletes your local copy of

that branch.

Now you're ready to create a new branch, write some code, and do it again.

## Try this next

[Set up continuous integration and delivery](#) or [learn more about working with a Git repo](#).



# Default Git repository and branch permissions

1/25/2019 • 3 minutes to read • [Edit Online](#)

## Azure DevOps Services | Azure DevOps Server 2019 | TFS 2018 | TFS 2017 | TFS 2015 | TFS 2013

After you've been added as a team member, you are a member of the Contributors group. This membership allows you to contribute to a Git repository. The most common built-in groups include Readers, Contributors, and Project Administrators. These groups are assigned the default permissions for contributing to a branch or repository.

From the project admin content, on the **Version Control** page, you can [set permissions on a repository](#).

From the **Code > Branches** page, you can [set permissions for a specific branch and set branch policies](#).

Set permissions across all Git repositories by making changes to the top-level **Git repositories** entry. Individual repositories inherit permissions from the top-level **Git Repositories** entry. Branches inherit a subset of permissions from assignments made at the repository level. For branch permissions and policies, see [Set branch permissions](#) and [Improve code quality with branch policies](#).

TASK	READERS	CONTRIBUTORS	BUILD ADMINS	PROJECT ADMINS
Clone, fetch, contribute to pull requests, and explore the contents of a repository	✓	✓	✓	✓
Contribute to a repository, create branches, create tags, manage notes		✓	✓	✓
Create, delete, and rename repositories				✓
Edit policies, Manage permissions, Remove others' locks				✓
Bypass policies when completing pull requests, Bypass policies when pushing, Force push (rewrite history, delete branches and tags) ( <i>not set for any security group</i> )				

Set permissions across all Git repositories by making changes to the top-level **Git repositories** entry. Individual repositories inherit permissions from the top-level **Git Repositories** entry. Branches inherit a subset of permissions from assignments made at the repository level. For branch permissions and policies, see [Set branch permissions](#) and [Improve code quality with branch policies](#).

By default, the project-level Readers groups have read-only permissions.

TASK	CONTRIBUTORS	BUILD ADMINS	PROJECT ADMINS
<b>Branch Creation:</b> At the repository level, can push their changes to branches in the repository. Does not override restrictions in place from <a href="#">branch policies</a> . At the branch level, can push their changes to the branch and lock the branch.	✓	✓	✓
<b>Contribute:</b> At the repository level, can push their changes to branches in the repository. Does not override restrictions in place from <a href="#">branch policies</a> . At the branch level, can push their changes to the branch and lock the branch.	✓	✓	✓

<b>Note Management:</b> Can push and edit Git notes to the repository. They can also remove notes from items if they have the Force permission.	✓	✓	✓
<b>Tag Creation:</b> Can push tags to the repository, and can also edit or remove tags if they have the Force permission.	✓	✓	✓
<b>Administer:</b> Delete and rename repositories If assigned to the top-level <b>Git repositories</b> entry, can add additional repositories. At the branch level, users can set permissions for the branch and unlock the branch. The Administer permission set on an individual Git repository does not grant the ability to rename or delete the repository. These tasks require Administer permissions at the top-level <b>Git repositories</b> entry.			✓
<b>Rewrite and destroy history (force push):</b> Can force an update to a branch and delete a branch. A force update can overwrite commits added from any user. Users with this permission can modify the commit history of a branch.			✓

The Project Collection Build Service can read from all repositories by default. Any pipeline which runs with project collection scope can potentially read any repository in the organization/collection. You can remove this permission for a repository: set "Read" to "Deny" for the Project Collection Build Service.

## Related articles

- [Set repository permissions for Git or TFVC](#)
- [Set permissions for a specific branch and set branch policies](#)
- [Git permissions prior to TFS 2017 Update 1](#)
- [Default permissions and access](#)
- [Permissions and groups reference](#)

# Key concepts

1/31/2019 • 6 minutes to read • [Edit Online](#)

Here you'll find definitions of key concepts and artifacts used in Azure Repos.

## Branch

Branches are lightweight references that keep a history of commits and provide a way to isolate changes for a feature or a bug fix from your master branch and other work. Committing changes to a branch doesn't affect other branches. You can push and share branches with other people on your team without having to merge the changes into master.

Switching between branches is quick and easy. Git doesn't create multiple copies of your source code when you're working with branches - it uses the history information stored in commits to re-create the files in a branch when you start working on it.

Learn more: [branches](#), [branch organization](#), and [how we use branches at Microsoft](#).

## Branch policies

Branch policies are an important part of the Git workflow. You use them to help protect the important branches in your development, like `master`. Branch policies enable you to:

- Isolate work in progress from the completed work in your master branch.
- Guarantee that changes build before they get to master.
- Limit who can contribute to specific branches.
- Enforce who can create branches and the naming guidelines for the branches.
- Automatically include the right reviewers for every code change.
- Enforce best practices with required code reviewers.

Learn more: [branch policies](#).

## Clone

Create a complete local copy of an existing Git repo by cloning it. Cloning a repo downloads all [commits](#) and [branches](#) in the repo and sets up a named relationship with the existing repo that you cloned. Use this relationship to interact with the existing repo, [pushing](#) and [pulling](#) changes to share code with your team.

Learn more: [cloning](#).

## Commit

A commit is a group of changes saved to your local repository. You can share these changes to the remote repository by [pushing](#).

Learn more: [commits](#).

## Fork

A fork is a complete copy of a repository, including all files, commits, and (optionally) branches. The new fork acts as if someone cloned the original repository and then pushed to a new, empty repository. After a fork has been

created, new files, folders, and branches are not shared between the repositories unless a pull request carries them along. When you're ready to share those changes, it's easy to use pull requests to push the changes back to the original repository.

Learn more: [forks](#)

## Git

Git is the most commonly used version control system today and is quickly becoming the standard for version control. Git is a distributed version control system, so your local copy of code is a complete version control repository. These fully functional local repositories make it is easy to work offline or remotely. You commit your work locally, and then sync your copy of the repository with the copy on the server.

Git in Azure Repos is standard Git. You can use the clients and tools of your choice, such as Git for Windows, Mac, partners' Git services, and tools such as Visual Studio and Visual Studio Code.

Learn more: [Git and Azure Repos](#).

## Git workflow

Version control has a general workflow that most developers use when writing code and sharing it with the team. These steps are:

1. Get a local copy of code if they don't have one yet.
2. Make changes to code to fix bugs or add new features.
3. When the code is ready, make it available for your team to review.
4. After the code is reviewed, merge it into the team's shared codebase.

Git has a version of this workflow that uses terminology and commands such as repositories, branches, commits, and pull requests. These terms might sound familiar if you've used a version control system like Team Foundation Version Control (TFVC) or Subversion, but they behave differently in Git.

1. [Create a branch](#) for the changes you plan to make and give it a name, such as `users/jamal/fix-bug-3214` or `features/cool-feature`.
2. [Commit changes](#) to your branch. People often have multiple commits for a bug fix or feature.
3. [Push your branch](#) to the remote repository.
4. [Create a pull request](#) so other people can review your changes. To incorporate feedback, you might need to make more commits and push more changes. When the code is ready, complete the pull request and merge your code into the target branch, such as `master`.

Use this workflow if you're new to Git. As your team gets more experienced and confident with Git, extend it to suit your team's needs.

Learn more: [how we use Git at Microsoft](#).

## Notifications

With notifications, you receive an email when changes occur to work items, code reviews, pull requests, source control files, and builds. For example, you can get notified whenever a bug that you opened is resolved, or when a work item is assigned to you. You receive notifications based on rules or subscriptions made by you, for your teams, or for the project. Learn more: [About notifications](#).

## Projects

A project, which was previously known as a *team project*, provides a repository for source code. A project provides

a place where a group of people can plan, track progress, and collaborate on building software solutions. A project is defined for an Azure DevOps Services organization or within a TFS project collection. You can use it to focus on those objects defined within the project. To learn more, see [About projects and scaling your organization](#).

## Public projects

A project created within an Azure DevOps Services organization that is visible to the whole world. Everyone in the world can discover them and perform limited operations. Administrators can control who gets to fully contribute. Administrators can switch a project from private to public, and vice-versa, as described in [Change the project visibility](#).

## Pull request

Create pull requests to review and merge code in a [Git project](#). Pull requests let your team review code and give feedback on changes before you merge it into the master branch. Pull requests can come from either topic branches within the same repository or a branch in a [fork](#) of the original repository. Reviewers can step through the proposed changes, leave comments, and vote to approve or reject the code.

Learn more: [pull requests](#).

## Pull

A `pull` command updates the code in your local repository with the changes from other members of your team that are in the remote repository.

Learn more: [pull](#).

## Push

Share changes made in commits and branches by using the `push` command.

When you push, Git uploads the saved commits in your checked branch to the remote repository. If the branch exists on the remote repository, Git takes the [commits](#) and adds them to that branch on the remote repository. If that branch doesn't exist, Git creates a new branch with the same commits as your local branch.

Learn more: [push](#).

## Repository

A repository is a location for your code managed by version control. Azure Repos supports both [Git](#) and [TFVC](#).

## Teams

A team corresponds to a selected set of project members. With teams, organizations can subcategorize work to better focus on all the work they track within a project. Each team gets access to a suite of Agile tools. Teams can use these tools to work autonomously and collaborate with other teams across the enterprise. Each team can configure and customize each tool to meet their work requirements. To learn more, see [About teams and Agile tools](#).

## Team Foundation Version Control (TFVC)

A centralized version control system. With TFVC, devs have only one version of each file on their dev machines. Branches are path-based and created on the server. Historical data is maintained only on the server. Branches are path-based and created on the server. Learn more: [Use Team Foundation Version Control](#).

# Manage projects in Azure DevOps

1/31/2019 • 2 minutes to read • [Edit Online](#)

**Azure DevOps Services | Azure DevOps Server 2019 | TFS 2018 | TFS 2017 | TFS 2015 | TFS 2013**

Structure your projects by adding area paths, iteration paths, and teams.

## 5-minute quickstarts

- [Get started as an administrator](#)
- [Share your project vision](#)
- [Define area paths](#)
- [Define iteration paths or sprints](#)
- [Add a team](#)
- [Add users to a project or team](#)
- [Add administrators or set permissions at the project or collection level](#)

## Step-by-step tutorials

- [Change individual permissions, grant select access to specific functions](#)
- [Grant or restrict permissions to select tasks](#)
- [Customize a project \(Azure DevOps Services\)](#)

## Concepts

- [Customize a project \(TFS\)](#)
- [About areas and iterations](#)
- [About teams and Agile tools](#)
- [Resources granted to project members](#)

## How-to guides

- [Create a project](#)
- [Rename a project](#)
- [Delete a project :: moniker range="azure-devops"](#)
- [Restore a project :: moniker-end](#)
- [Change service visibility](#)
- [Connect to projects](#)

## Reference

- [Default permissions and access](#)
- [Permission lookup guide \(Security\)](#)
- [Azure DevOps data protection overview](#)

## Resources

- [Azure DevOps Services user guide](#)

- [Public Projects](#)
- [Security & identity](#)
- [Migrate from TFS to Azure DevOps Services](#)