Introduction

Completed

- 1 minute

This module describes different source control systems, such as Git and Team Foundation Version Control (TFVC), and helps with the initial steps for Git utilization.

Learning objectives

After completing this module, students and professionals can:

- Apply source control practices in your development process.
- Explain the differences between centralized and distributed version control.
- Understand Git and Team Foundation Version Control (TFVC).
- Develop using Git.

Prerequisites

- Understanding of what DevOps is and its concepts.
- Familiarity with version control principles is helpful but isn't necessary.
- Beneficial to have experience in an organization that delivers software.

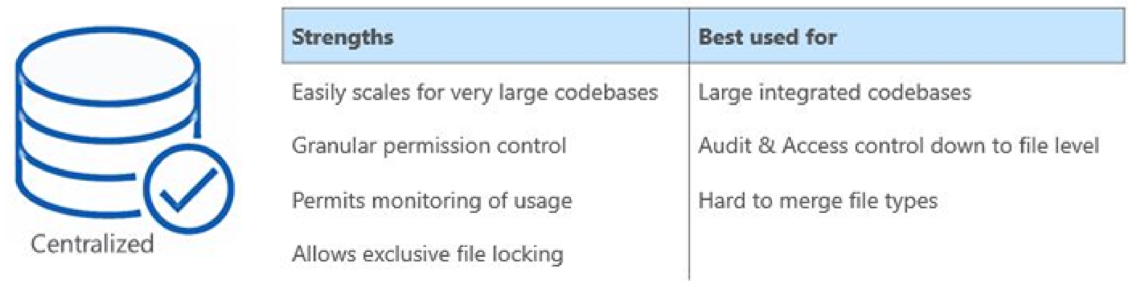
Continue

Need help? See our troubleshooting guide or provide specific feedback by reporting an issue .

Understand centralized source control

Completed

- 2 minutes


Centralized

Strengths	Best used for
Easily scales for very large codebases	Large integrated codebases
Granular permission control	Audit & Access control down to file level
Permits monitoring of usage	Hard to merge file types
Allows exclusive file locking	

Centralized source control systems are based on the idea that there's a single "central" copy of your project somewhere (probably on a server). Programmers will check in (or commit) their changes to this central copy.

"Committing" a change means to record the difference in the central system. Other programmers can then see this change.

Also, it's possible to pull down the change. The version control tool will automatically update the contents of any files that were changed.

Most modern version control systems deal with "changesets," which are a group of changes (possibly too many files) that should be treated as a cohesive whole.

Programmers no longer must keep many copies of files on their hard drives manually. The version control tool can talk to the central copy and retrieve any version they need on the fly.

Some of the most common-centralized version control systems you may have heard of or used are Team Foundation Version Control (TFVC), CVS, Subversion (or SVN), and Perforce.

A typical centralized source control workflow

If working with a centralized source control system, your workflow for adding a new feature or fixing a bug in your project will usually look something like this:

- Get the latest changes other people have made from the central server.
- Make your changes, and make sure they work correctly.

- Check in your changes to the main server so that other programmers can see them.

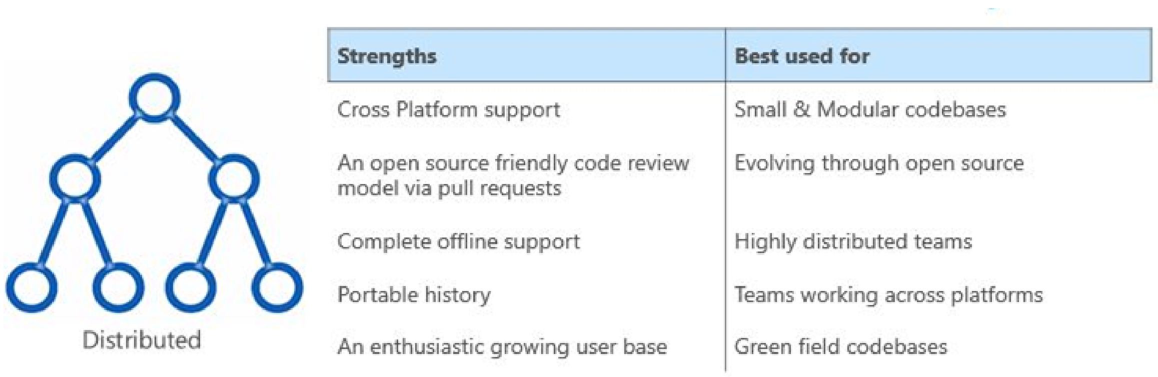
Continue

Need help? See our troubleshooting guide or provide specific feedback by reporting an issue .

Understand distributed source control

Completed

- 3 minutes


Distributed

Strengths	Best used for
Cross Platform support	Small & Modular codebases
An open source friendly code review model via pull requests	Evolving through open source
Complete offline support	Highly distributed teams
Portable history	Teams working across platforms
An enthusiastic growing user base	Green field codebases

Over time, so-called "distributed" source control or version control systems (DVCS for short) have become the most important.

The three most popular are Git, Mercurial, and Bazaar. These systems don't necessarily rely on a central server to store all the versions of a project's files. Instead, every developer "clones" a repository copy and has the project's complete history on their local storage. This copy (or "clone") has all the original metadata.

This method may sound wasteful but it isn't a problem in practice. Most programming projects consist primarily of plain text files (maybe a few images).

The disk space is so cheap that storing many copies of a file doesn't create a noticeable dent in a local storage free space. Modern systems also compress the files to use even less space; for example, objects (and deltas) are stored compressed, and text files used in programming compress well (around 60% of original size, or 40% reduction in size from compression).

Getting new changes from a repository is called "pulling." Moving your changes to a repository is called "pushing." You move changesets (changes to file groups as coherent wholes), not single-file diffs.

One common misconception about distributed version control systems is that there can't be a central project repository. It isn't true. Nothing stops you from saying, "this copy of the project is the authoritative one."

It means that instead of a central repository required by your tools, it's now optional.

Advantages over centralized source control

The act of cloning an entire repository gives distributed source control tools several advantages over centralized systems:

- Doing actions other than pushing and pulling changesets is fast because the tool only needs to access the local storage, not a remote server.
- Committing new changesets can be done locally without anyone else seeing them. Once you have a group of changesets ready, you can push all of them at once.
- Everything but pushing and pulling can be done without an internet connection. So, you can work on a plane, and you won't be forced to commit several bug fixes as one large changeset.
- Since each programmer has a full copy of the project repository, they can share changes with one, or two other people to get feedback before showing the changes to everyone.

Disadvantages compared to centralized source control

There are almost no disadvantages to using a distributed source control system over a centralized one.

Distributed systems don't prevent you from having a single "central" repository; they provide more options.

There are only two major inherent disadvantages to using a distributed system:

- If your project contains many large, binary files that can't be efficiently compressed, the space needed to store all versions of these files can accumulate quickly.
- If your project has a long history (50,000 changesets or more), downloading the entire history can take an impractical amount of time and disk space.

Continue

Need help? See our troubleshooting guide or provide specific feedback by reporting an issue .

Explore Git and Team Foundation Version Control

Completed

- 2 minutes

Git (distributed)

Git is a distributed version control system. Each developer has a copy of the source repository on their development system. Developers can commit each set of changes on their dev machine.

Branches are lightweight. When you need to switch contexts, you can create a private local branch. You can quickly switch from one branch to another to pivot among different variations of your codebase. Later, you can merge, publish, or dispose of the branch.

Team Foundation Version Control (TFVC-centralized)

Team Foundation Version Control (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.

TFVC has two workflow models:

- **Server workspaces -** Before making changes, team members publicly check out files. Most operations require developers to be connected to the server. This system helps lock workflows. Other software that works this way includes Visual Source Safe, Perforce, and CVS. You can scale up to huge codebases with millions of files per branch—also, large binary files with server workspaces.
- **Local workspaces -** Each team member copies the latest codebase version with them and works offline as needed. Developers check in their changes and resolve conflicts as necessary. Another system that works this way is Subversion.

Continue

Need help? See our troubleshooting guide or provide specific feedback by reporting an issue .

Examine and choose Git

Completed

- 4 minutes

Switching from a centralized version control system to Git changes the way your development team creates software.

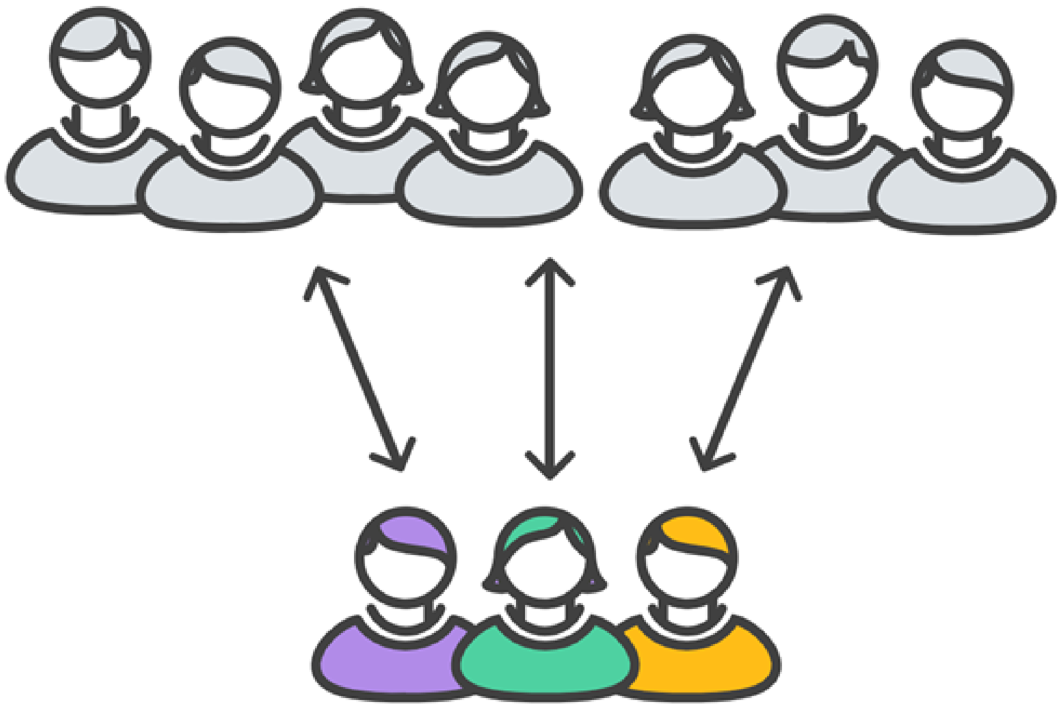
If you are a company that relies on its software for mission-critical applications, altering your development workflow impacts your entire business.

Developers would gain the following benefits by moving to Git.

Community

In many circles, Git has come to be the expected version control system for new projects.

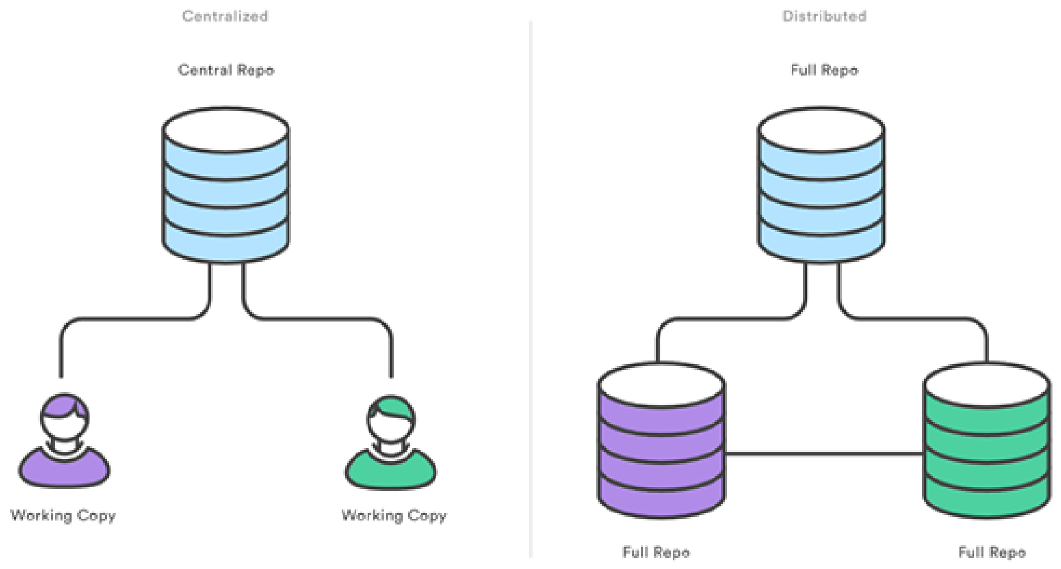
If your team is using Git, odds are you will not have to train new hires on your workflow because they will already be familiar with distributed development.



Also, Git is popular among open-source projects. It is easy to use 3rd-party libraries and encourage others to fork your open-source code.

Distributed development

In TFVC, each developer gets a working copy that points back to a single central repository. Git, however, is a distributed version control system. Instead of a working copy, each developer gets their local repository, complete with an entire history of commits.



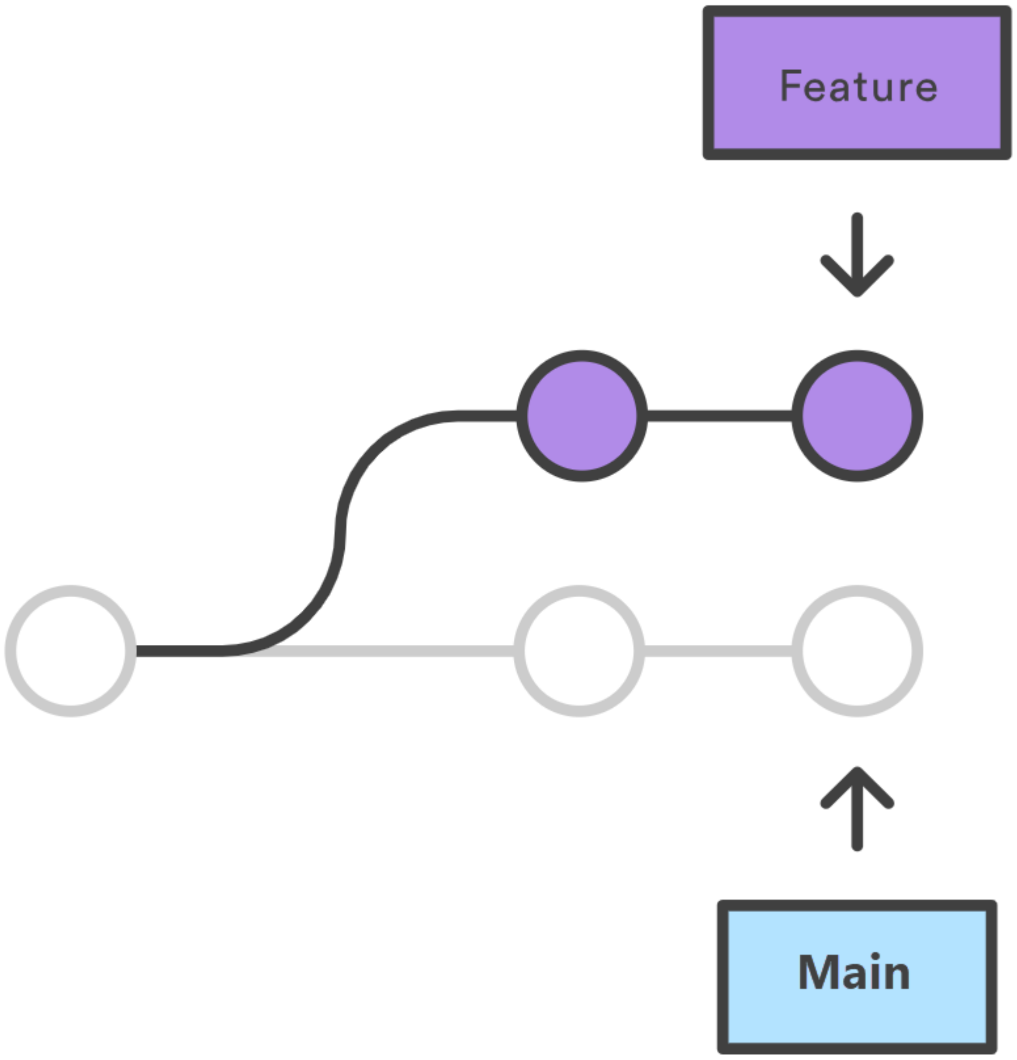
Having a complete local history makes Git fast since it means you do not need a network connection to create commits, inspect previous versions of a file, or do diffs between commits.

Distributed development also makes it easier to scale your engineering team. If someone breaks the production branch in SVN, other developers cannot check in their changes until it is fixed. With Git, this kind of blocking does not exist. Everybody can continue going about their business in their local repositories.

And, like feature branches, distributed development creates a more reliable environment. Even if developers obliterate their repository, they can clone from someone else and start afresh.

Trunk-based development

One of the most significant advantages of Git is its branching capabilities. Unlike centralized version control systems, Git branches are cheap and easy to merge.



Trunk-based development provides an isolated environment for every change to your codebase. When developers want to start working on something—no matter how large or small—they create a new branch. It ensures that the main branch always contains production-quality code.

Using trunk-based development is more reliable than directly-editing production code, but it also provides organizational benefits.

They let you represent development work at the same granularity as your agile backlog.

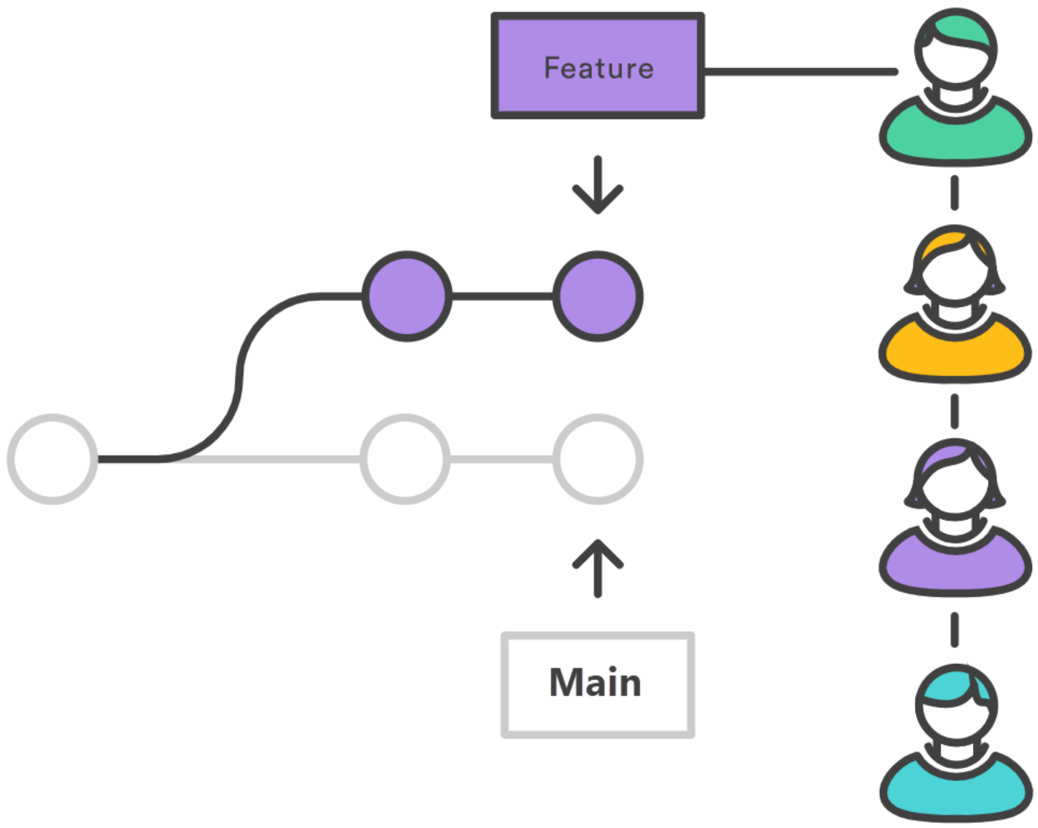
For example, you might implement a policy where each work item is addressed in its feature branch.

Pull requests

Many source code management tools such as Azure Repos enhance core Git functionality with pull requests.

A pull request is a way to ask another developer to merge one of your branches into their repository.

It makes it easier for project leads to keep track of changes and lets developers start discussions around their work before integrating it with the rest of the codebase.



Since they are essentially a comment thread attached to a feature branch, pull requests are incredibly versatile.

When a developer gets stuck with a complex problem, they can open a pull request to ask for help from the rest of the team.

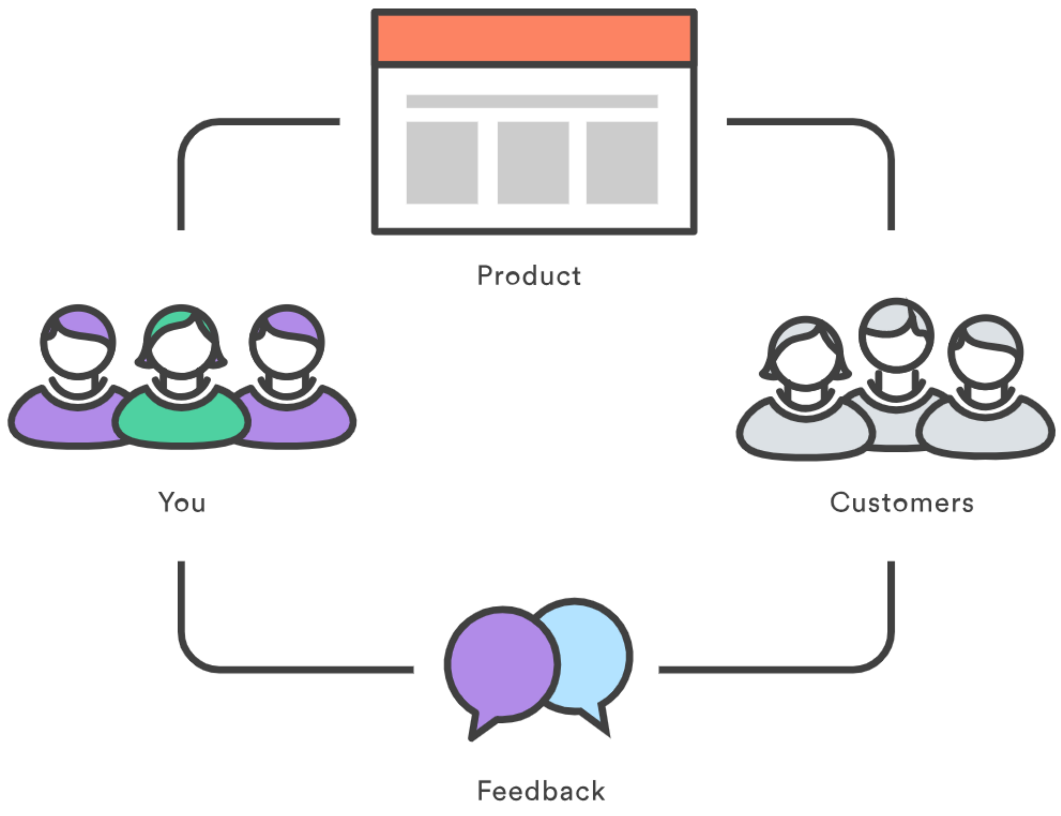
Instead, junior developers can be confident that they are not destroying the entire project by treating pull requests as a formal code review.

Faster release cycle

A faster release cycle is the ultimate result of feature branches, distributed development, pull requests, and a stable community.

These capabilities promote an agile workflow where developers are encouraged to share more minor changes more frequently.

In turn, changes can get pushed down the deployment pipeline faster than the standard of the monolithic releases with centralized version control systems.



As you might expect, Git works well with continuous integration and continuous delivery environments.

Git hooks allow you to run scripts when certain events occur inside a repository, which lets you automate deployment to your heart’s content.

You can even build or deploy code from specific branches to different servers.

For example, you might want to configure Git to deploy the most recent commit from the develop branch to a test server whenever anyone merges a pull request into it.

Combining this kind of build automation with peer review means you have the highest possible confidence in your code as it moves from development to staging to production.

Continue

Need help? See our troubleshooting guide or provide specific feedback by reporting an issue .

Understand objections to using Git

Completed

- 2 minutes

There are three common objections I often hear to migrating to Git:

- I can overwrite history.
- I have large files.
- There is a steep learning curve.

Overwriting history

Git technically does allow you to overwrite history - but like any helpful feature, if misused can cause conflicts.

If your teams are careful, they should never have to overwrite history.

If you are synchronizing to Azure Repos, you can also add a security rule that prevents developers from overwriting history by using the explicit "Force Push" permissions.

Every source control system works best when developers understand how it works and which conventions work.

While you cannot overwrite history with Team Foundation Version Control (TFVC), you can still overwrite code and do other painful things.

Large files

Git works best with repos that are small and do not contain large files (or binaries).

Every time you (or your build machines) clone the repo, they get the entire repo with its history from the first commit.

It is great for most situations but can be frustrating if you have large files.

Binary files are even worse because Git cannot optimize how they are stored.

That is why Git LFS was created.

It lets you separate large files of your repos and still has all the benefits of versioning and comparing.

Also, if you are used to storing compiled binaries in your source repos, stop!

Use Azure Artifacts or some other package management tool to store binaries for which you have source code.

However, teams with large files (like 3D models or other assets) can use Git LFS to keep the code repo slim and trimmed.

Learning curve

There is a learning curve. If you have never used source control before, you are probably better off when learning Git. I have found that users of centralized source control (TFVC or SubVersion) battle initially to make the mental shift, especially around branches and synchronizing.

Once developers understand how Git branches work and get over the fact that they must commit and then push, they have all the basics they need to succeed in Git.

Continue

Need help? See our troubleshooting guide or provide specific feedback by reporting an issue .

Describe working with Git locally

Completed

- 7 minutes

Git and Continuous Delivery is one of those delicious chocolate and peanut butter combinations. We occasionally find two great tastes that taste great together in the software world!

Continuous Delivery of software demands a significant level of automation. It is hard to deliver continuously if you do not have a quality codebase.

Git provides you with the building blocks to take charge of quality in your codebase. It allows you to automate most of the checks in your codebase.

Also, it works before committing the code into your repository.

To fully appreciate the effectiveness of Git, you must first understand how to carry out basic operations on Git. For example, clone, commit, push, and pull.

The natural question is, how do we get started with Git?

One option is to go native with the command line or look for a code editor that supports Git natively.

Visual Studio Code is a cross-platform, open-source code editor that provides powerful developer tooling for hundreds of languages.

To work in open-source, you need to embrace open-source tools.

This recipe will start by:

- Setting up the development environment with Visual Studio Code.
- Creating a new Git repository.
- Committing code changes locally.
- Pushing changes to a remote repository on Azure DevOps.

Getting ready

This tutorial will teach us how to initialize a Git repository locally.

Then we will use the ASP.NET Core MVC project template to create a new project and version it in the local Git repository.

We will then use Visual Studio Code to interact with the Git repository to do basic commit, pull, and push operations.

You will need to set up your working environment with the following:

- .NET Core 3.1 SDK or later: Download .NET .
- Visual Studio Code: Download Visual Studio Code .
- C# Visual Studio Code extension: C# programming with Visual Studio Code .
- Git: Git - Downloads
- Git for Windows (if you are using Windows): Git for Windows

The Visual Studio Marketplace features several extensions for Visual Studio Code that you can install to enhance your experience of using Git:

- Git Lens : This extension brings visualization for code history by using Git blame annotations and code lens. The extension enables you to seamlessly navigate and explore the history of a file or branch. Also, the extension allows you to gain valuable insights via powerful comparison commands and much more.
- Git History : Brings visualization and interaction capabilities to view the Git log, file history and compare branches or commits.

How to do it

1. Open the Command Prompt and create a new-working folder:

   ```
   mkdir myWebApp
   cd myWebApp
   ```

2. In myWebApp, initialize a new Git repository:

   ```
   git init
   ```

3. Configure global settings for the name and email address to be used when committing in this Git repository:

   ```
   git config --global user.name "John Doe"
   git config --global user.email "john.doe@contoso.com"
   ```

   If you are working behind an enterprise proxy, you can make your Git repository proxy-aware by adding the proxy details in the Git global configuration file.

   Different variations of this command will allow you to set up an HTTP/HTTPS proxy (with username/password) and optionally bypass SSL verification.

   Run the below command to configure a proxy in your global git config.

   ```
   git config --global http.proxy
   http://proxyUsername:proxyPassword@proxy.server.com:port
   ```

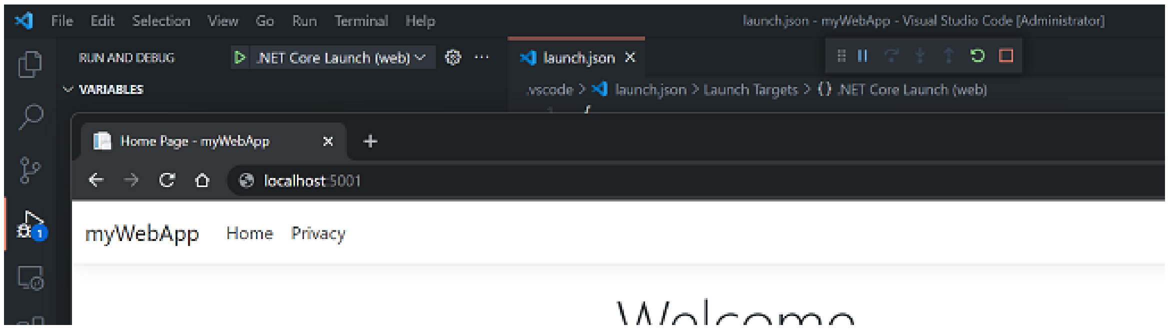
4. Create a new ASP.NET core application. The new command offers a collection of switches that can be used for language, authentication, and framework selection. More details can be found on Microsoft docs .

   ```
   dotnet new mvc
   ```

   Launch Visual Studio Code in the context of the current-working folder:

```
code .
```

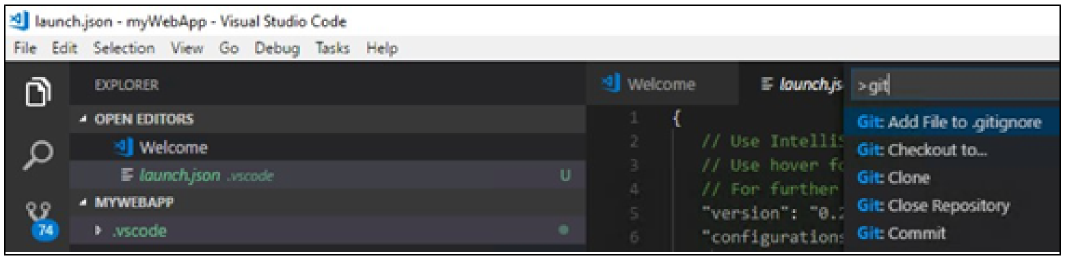
5. When the project opens in Visual Studio Code, select **Yes** for the **Required assets to build and debug are missing from 'myWebApp.' Add them?** Warning message. Select **Restore** for the **There are unresolved dependencies** info message. Hit **F5** to debug the application, then myWebApp will load in the browser, as shown in the following screenshot:


File Edit Selection View Go Run Terminal Help
launch.json - myWebApp - Visual Studio Code [Administrator]
RUN AND DEBUG
.NET Core Launch (web)
launch.json
VARIABLES
.vscode > launch.json > Launch Targets > {} .NET Core Launch (web)
Home Page - myWebApp
localhost:5001
myWebApp
Home
Privacy
Welcome

If you prefer to use the command line, you can run the following commands in the context of the git repository to run the web application.

```
dotnet build
dotnet run
```

You will notice the ".vscode" folder is added to your working folder. To avoid committing this folder to your Git repository, you can include it in the .gitignore file. Select a file from the ".vscode" folder, hit F1 to launch the command window in Visual Studio Code, type gitIgnore, and accept the option to include the selected file in the new .gitIgnore file.

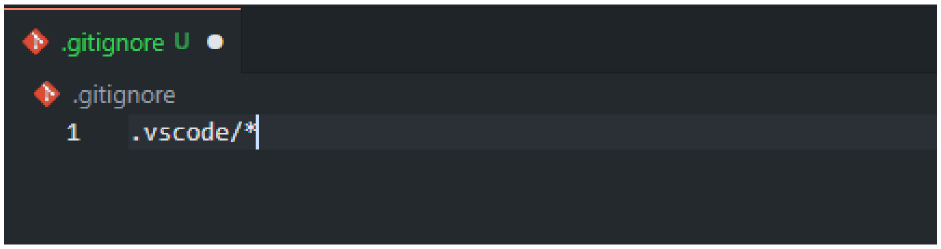


```
{
    // Use IntelliS
    // Use hover fo
    // For further
    "version": "0.2
    "configurations
```

Note

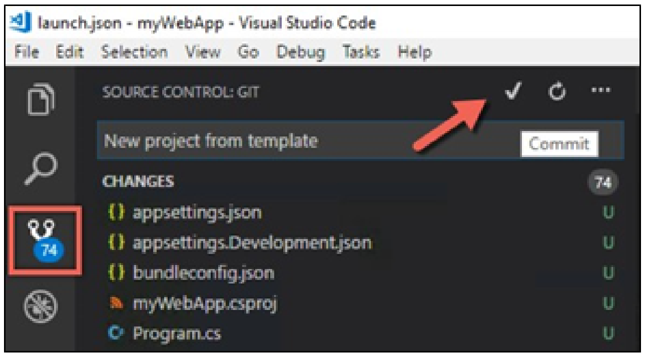
To ignore an entire directory, you need to include the name of the directory with the slash / at the end.

Open your .gitignore, remove the file name from the path, and leave the folder with a slash, for example, *.vscode/** .



```
.vscode/*
```

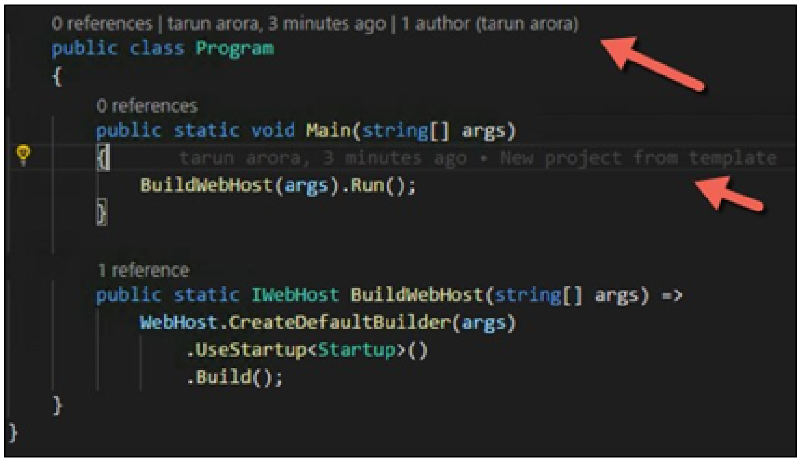
6. To stage and commit the newly created myWebApp project to your Git repository from Visual Studio Code, navigate the Git icon from the left panel. Add a commit comment and commit the changes by clicking the checkmark icon. It will stage and commit the changes in one operation:


launch.json - myWebApp - Visual Studio Code
File Edit Selection View Go Debug Tasks Help
SOURCE CONTROL: GIT
Commit
CHANGES 74
74

New project from template

appsettings.json	U
appsettings.Development.json	U
bundleconfig.json	U
myWebApp.csproj	U
Program.cs	U

Open Program.cs, you will notice Git lens decorates the classes and functions with the commit history and brings this information in line to every line of code:



```
0 references | tarun arora, 3 minutes ago | 1 author (tarun arora)
public class Program
{
    0 references
    public static void Main(string[] args)
    {        tarun arora, 3 minutes ago • New project from template
        BuildWebHost(args).Run();
    }

    1 reference
    public static IWebHost BuildWebHost(string[] args) =>
        WebHost.CreateDefaultBuilder(args)
            .UseStartup<Startup>()
            .Build();
}
}
```

7. Now launch cmd in the context of the git repository and run `git branch --list` . It will show you that only the `main` branch currently exists in this repository. Now run the following command to create a new branch called `feature-devops-home-page` .

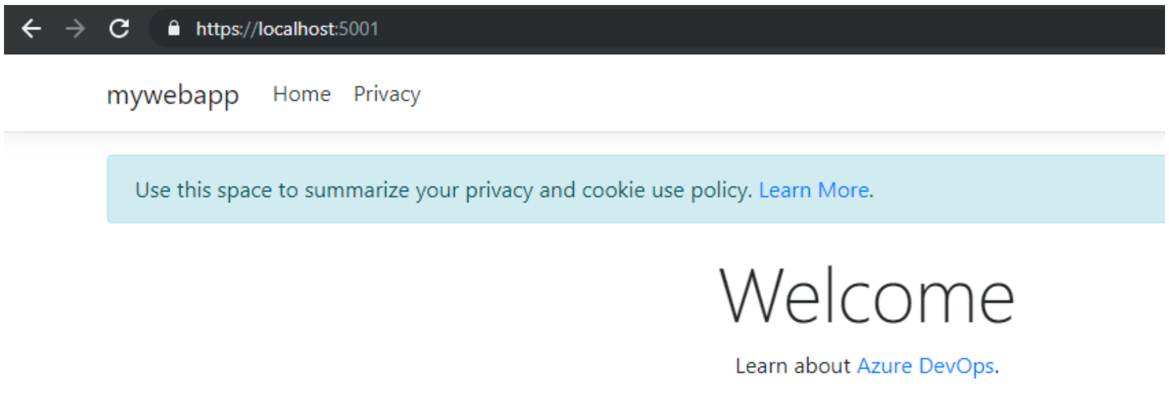
```
git branch feature-devops-home-page
git checkout feature-devops-home-page
git branch --list
```

You have created a new branch with these commands and checked it out. The `--list` keyword shows you a list of all branches in your repository. The green color represents the branch that is currently checked out.

8. Now navigate to the file `~\Views\Home\Index.cshtml` and replace the contents with the text below.

```
@{
    ViewData["Title"] = "Home Page";
}

<div class="text-center">
    <h1 class="display-4">Welcome</h1>
    <p>Learn about <a
href="https://azure.microsoft.com/services/devops/">Azure
DevOps</a>.</p>
</div>
```

9. Refresh the web app in the browser to see the changes.



Use this space to summarize your privacy and cookie use policy. Learn More.

Welcome

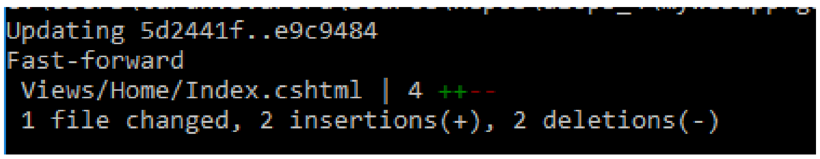
Learn about Azure DevOps.

10. In the context of the git repository, execute the following commands. These commands will stage the changes in the branch and then commit them.

```
git status

git add .

git commit -m "updated welcome page."

git status
```

11. To merge the changes from the feature-devops-home-page into the main, run the following commands in the context of the git repository.

```
git checkout main

git merge feature-devops-home-page
```



```
Updating 5d2441f..e9c9484
Fast-forward
 Views/Home/Index.cshtml | 4 ++--
 1 file changed, 2 insertions(+), 2 deletions(-)
```

12. Run the below command to delete the feature branch.

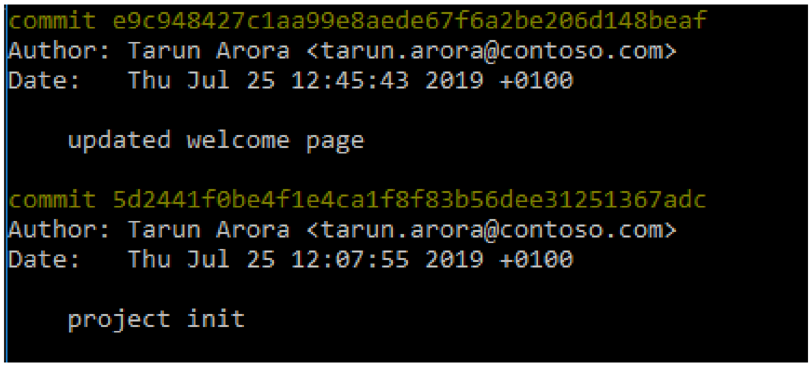
```
git branch --delete feature-devops-home-page
```

How it works

The easiest way to understand the outcome of the steps done earlier is to check the history of the operation. Let us have a look at how to do it.

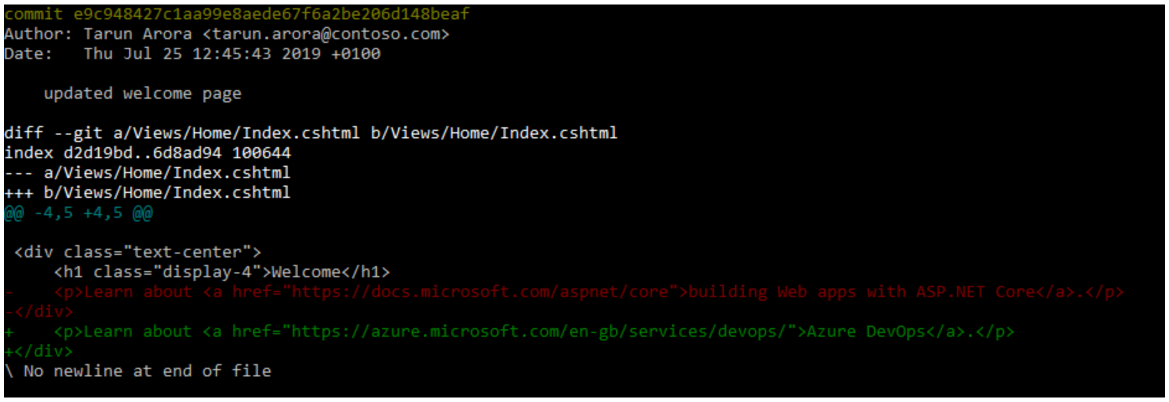
1. In Git, committing changes to a repository is a two-step process. Running: `add .` The changes are staged but not committed. Finally, running the commit promotes the staged changes in the repository.

2. To see the history of changes in the main branch, run the command `git log -v`



```
commit e9c948427c1aa99e8aede67f6a2be206d148beaf
Author: Tarun Arora <tarun.arora@contoso.com>
Date:   Thu Jul 25 12:45:43 2019 +0100

    updated welcome page

commit 5d2441f0be4f1e4ca1f8f83b56dee31251367adc
Author: Tarun Arora <tarun.arora@contoso.com>
Date:   Thu Jul 25 12:07:55 2019 +0100

    project init
```

3. To investigate the actual changes in the commit, you can run the command `git log -p`



```
commit e9c948427c1aa99e8aede67f6a2be206d148beaf
Author: Tarun Arora <tarun.arora@contoso.com>
Date:   Thu Jul 25 12:45:43 2019 +0100

    updated welcome page

diff --git a/Views/Home/Index.cshtml b/Views/Home/Index.cshtml
index d2d19bd..6d8ad94 100644
--- a/Views/Home/Index.cshtml
+++ b/Views/Home/Index.cshtml
@@ -4,5 +4,5 @@

 <div class="text-center">
     <h1 class="display-4">Welcome</h1>
-    <p>Learn about <a href="https://docs.microsoft.com/aspnet/core">building Web apps with ASP.NET Core</a>.</p>
-</div>
+    <p>Learn about <a href="https://azure.microsoft.com/en-gb/services/devops/">Azure DevOps</a>.</p>
+</div>
\ No newline at end of file
```

There is more

Git makes it easy to back out changes. Following our example, if you want to take out the changes made to the welcome page.

You can do It hard resetting the main branch to a previous version of the commit using the following command.

```
git reset --hard 5d2441f0be4f1e4ca1f8f83b56dee31251367adc
```

Running the above command would reset the branch to the project init change.

If you run `git log -v,` you will see that the changes done to the welcome page are removed from the repository.

Continue

Need help? See our troubleshooting guide or provide specific feedback by reporting an issue .

Knowledge check

Completed

- 4 minutes

Choose the best response for each question. Then select **Check your answers** .

Check your knowledge

1.

Which of the following choices correctly describes what is source control?

○

Source control is the practice of controlling what is deployed to test and production environments.

○

Source control is the practice of controlling security through code files.

○

Source control is the practice of tracking and managing changes to code.

2.

Which of the following choices isn't a benefit of using distributed version control?

○

Complete offline support

○

Allows exclusive file locking.

○

Portable history.

3.

Which of the following choices isn't a benefit of using centralized version control?

○

Easily scales for large codebases.

○

An open-source friendly code review model via pull requests.

○

Allows exclusive file locking.

Check your answers

You must answer all questions before checking your work.

Continue

Summary

Completed

- 1 minute

This module described different source control systems such Git and Team Foundation Version Control (TFVC) and helped with the initial steps for Git utilization.

You learned how to describe the benefits and usage of:

- Apply source control practices in your development process.
- Explain the differences between centralized and distributed version control.
- Understand Git and Team Foundation Version Control (TFVC).
- Develop using Git.

Learn more

- What is Team Foundation Version Control - Azure Repos | Microsoft Docs .
- Migrate from TFVC to Git - Azure DevOps | Microsoft Docs .
- Git and TFVC version control - Azure Repos | Microsoft Docs .
- Get started with Git and Visual Studio - Azure Repos | Microsoft Docs .

Continue