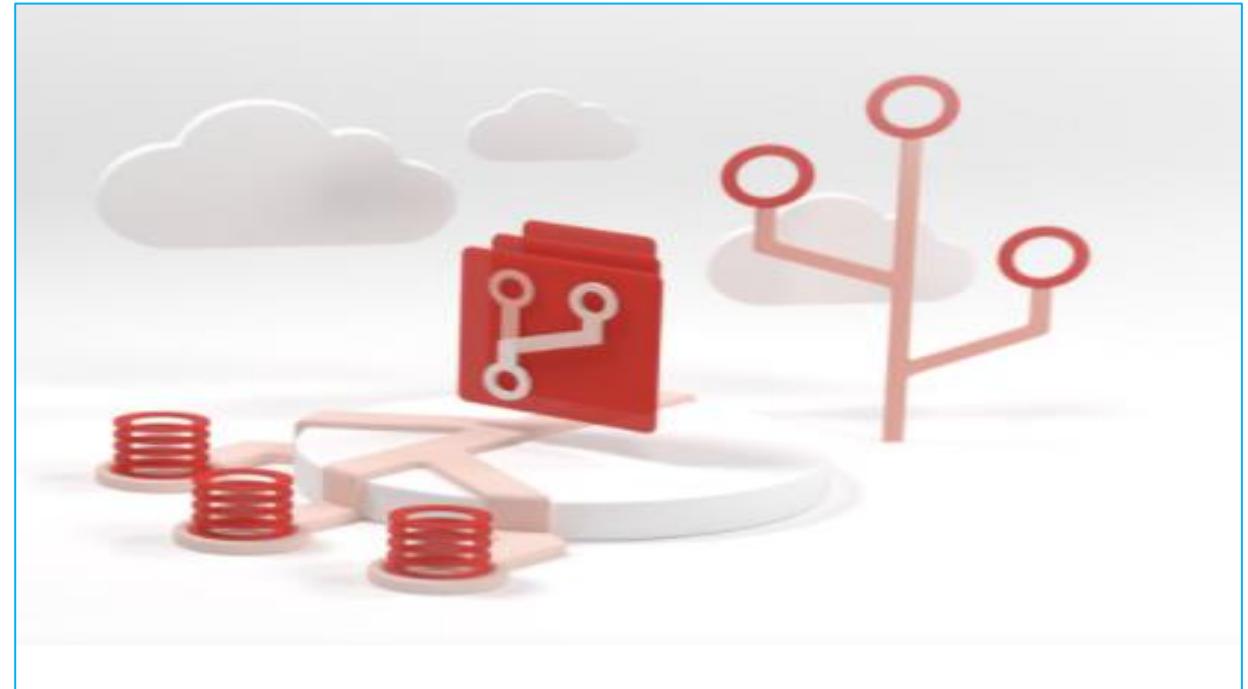


Azure Repos

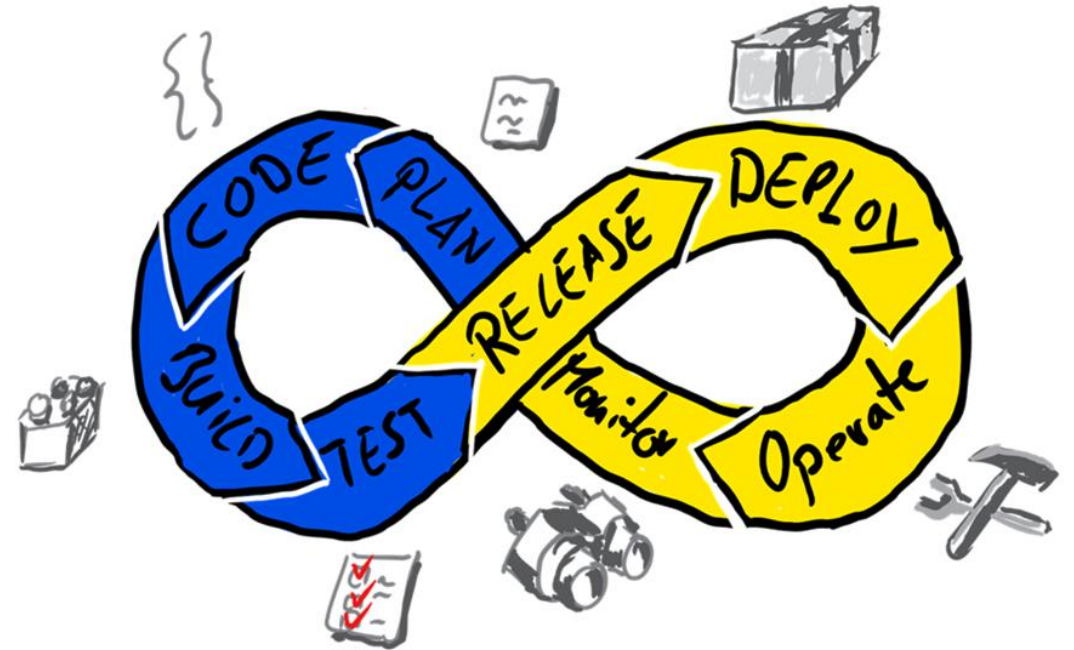
Azure Repos

Get unlimited, cloud-hosted private Git repos for your project



Introduction to Source Control

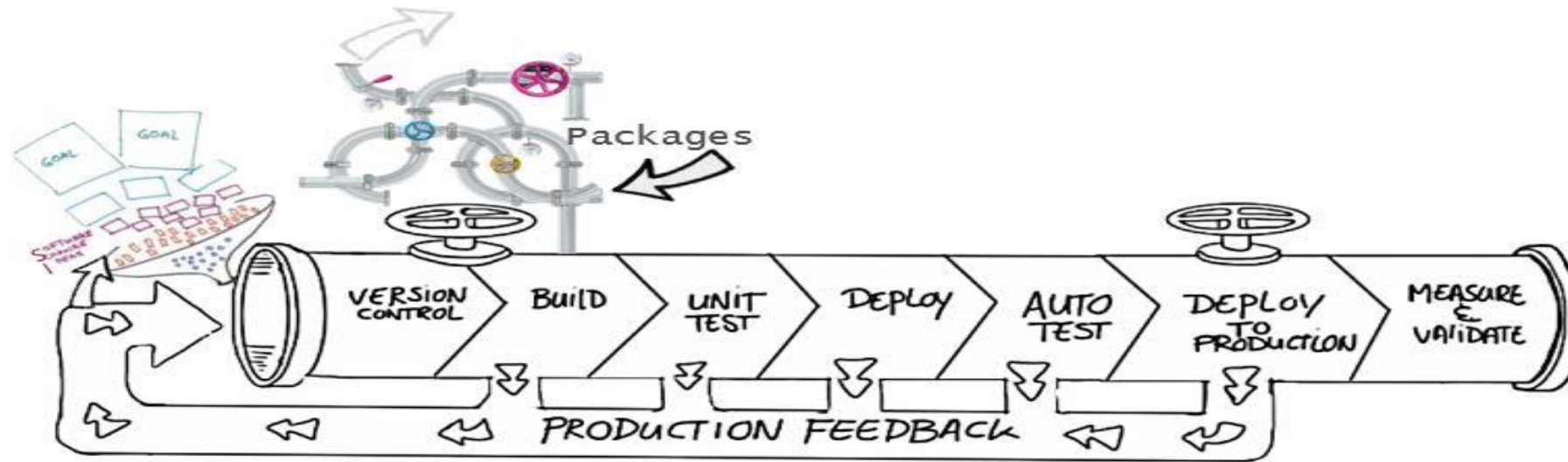
- DevOps is a revolutionary way to release software quickly and efficiently while maintaining a high level of security
- Source control (version control) is a critical part of DevOps



What is Source Control?

- Source control is the practice of tracking and managing changes to code
- Source control management (SCM) systems provide a running history of code development and help to resolve conflicts when merging contributions from multiple sources
- Source control protects source code from both catastrophe and the casual degradation of human error and unintended consequences
- Benefits include: reusability, traceability, manageability, efficiency, collaboration, and learning.

Benefits of Source Control



- Create workflows
- Work with versions
- Collaboration
- Maintains history of changes
- Automate tasks

Best Practices for Source Control

- Make small changes
- Don't commit personal files
- Update often and right before pushing to avoid merge conflicts
- Verify your code change before pushing it to a repository; ensure it compiles and tests are passing.
- Pay close attention to commit messages as these will tell you why a change was made
- Link code changes to work items
- No matter your background or preferences, be a team player and follow agreed conventions and workflows

Centralized Source Control



Strengths

- Easily scales for very large codebases
- Granular permission control
- Permits monitoring of usage
- Allows exclusive file locking

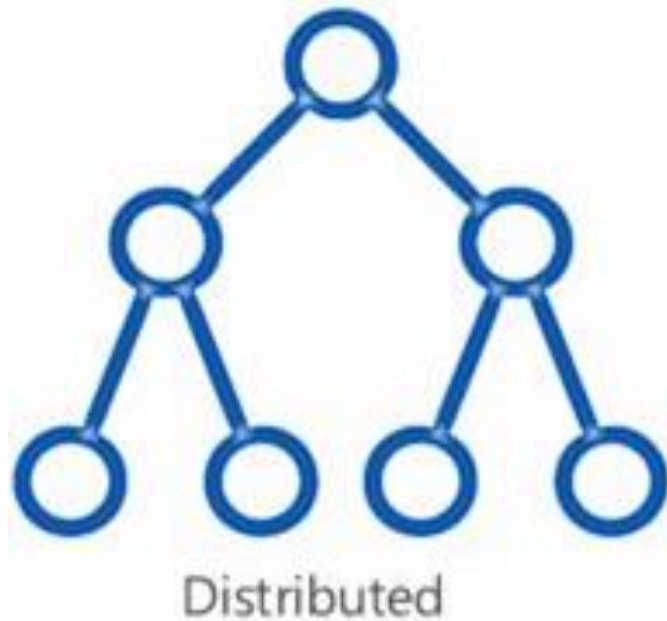
Best Used for

- Large integrated codebases
- Audit and access control down to the file level
- Hard to merge file types

- There is a single central copy of your project and programmers commit their changes to this central copy
- Common centralized version control systems are TFVC, CVS, Subversion (or SVN) and Perforce

Distributed Source Control

- Every developer clones a copy of a repository and has the full history of the project
- Common distributed source control systems are Mercurial, Git and Bazaar.



Strengths

- Cross platform support
- An open source friendly code review model via pull requests
- Complete offline support
- Portable history
- An enthusiastic growing user based

Best Used for

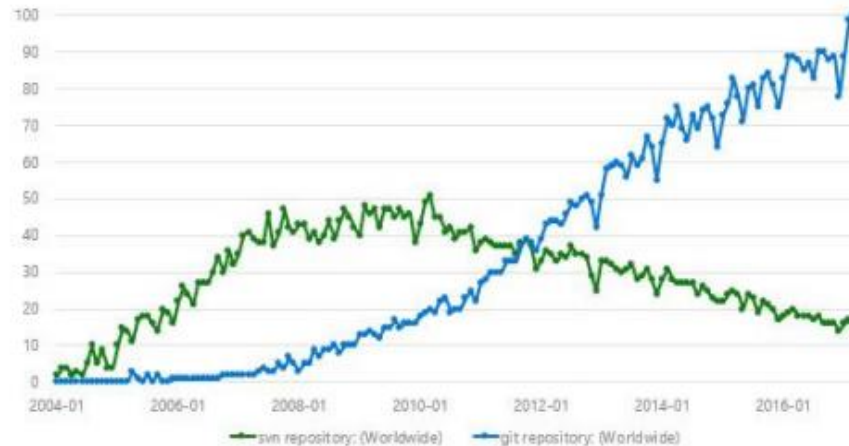
- Small and modular codebases
- Evolving through open source
- Highly distributed teams
- Teams working across platforms
- Greenfield codebases

GIT and TFVC

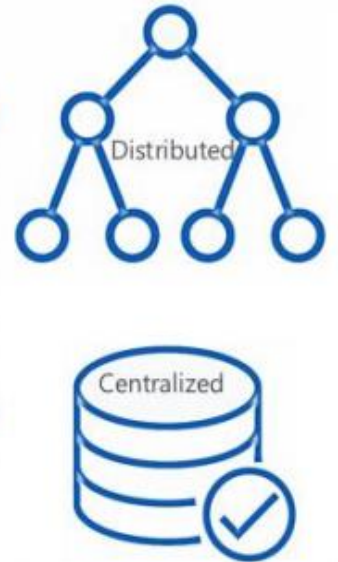
- Git
 - Distributed source control system
 - Each developer has a copy of the source repository on their dev machine
- TFVC
 - Centralized source control system
 - Team members have only one version of each file on their dev machines
 - In the *Server workspaces* model, before making changes, team members publicly check out files
 - In the *Local workspaces* model, each team member takes a copy of the latest version of the codebase with them and works offline as needed

Git and TFVC

Google Trend: svn (centralized) vs git (distributed)

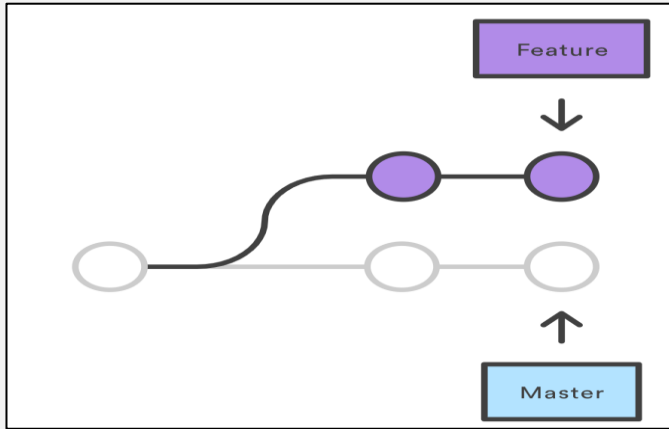


<https://trends.google.com/trends/explore?cat=5&date=all&q=svn%20repository,git%20repository>

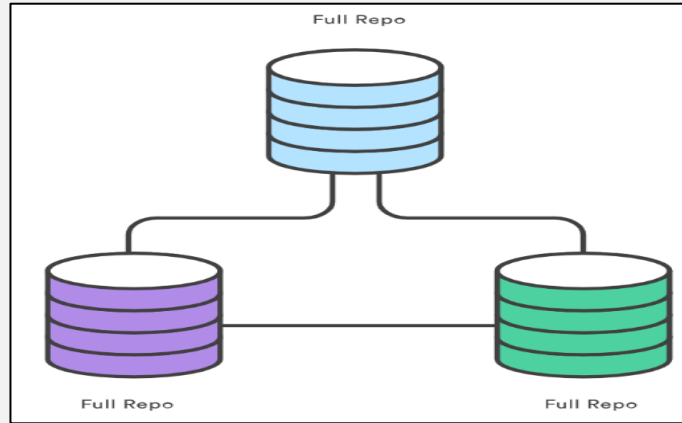


Why Git?

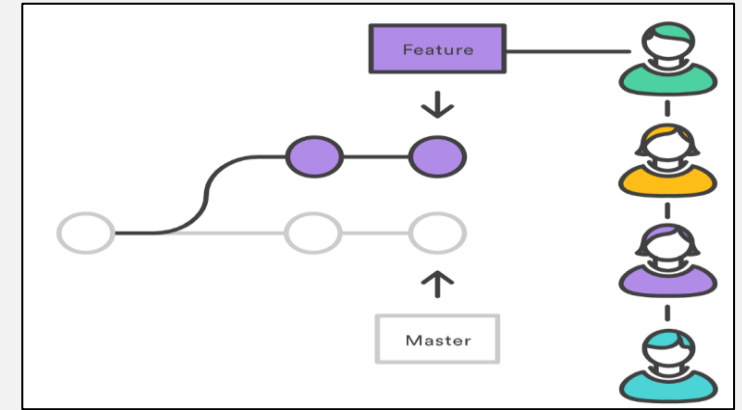
Feature branches



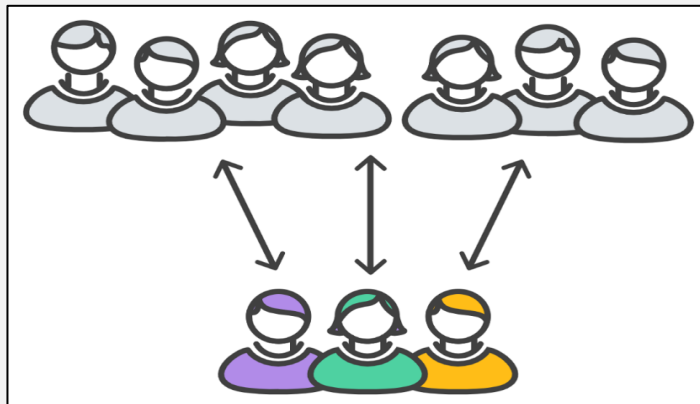
Distributed development



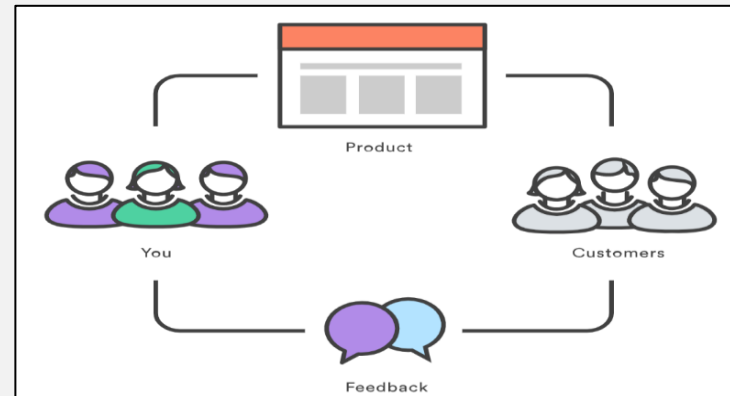
Pull requests



Community



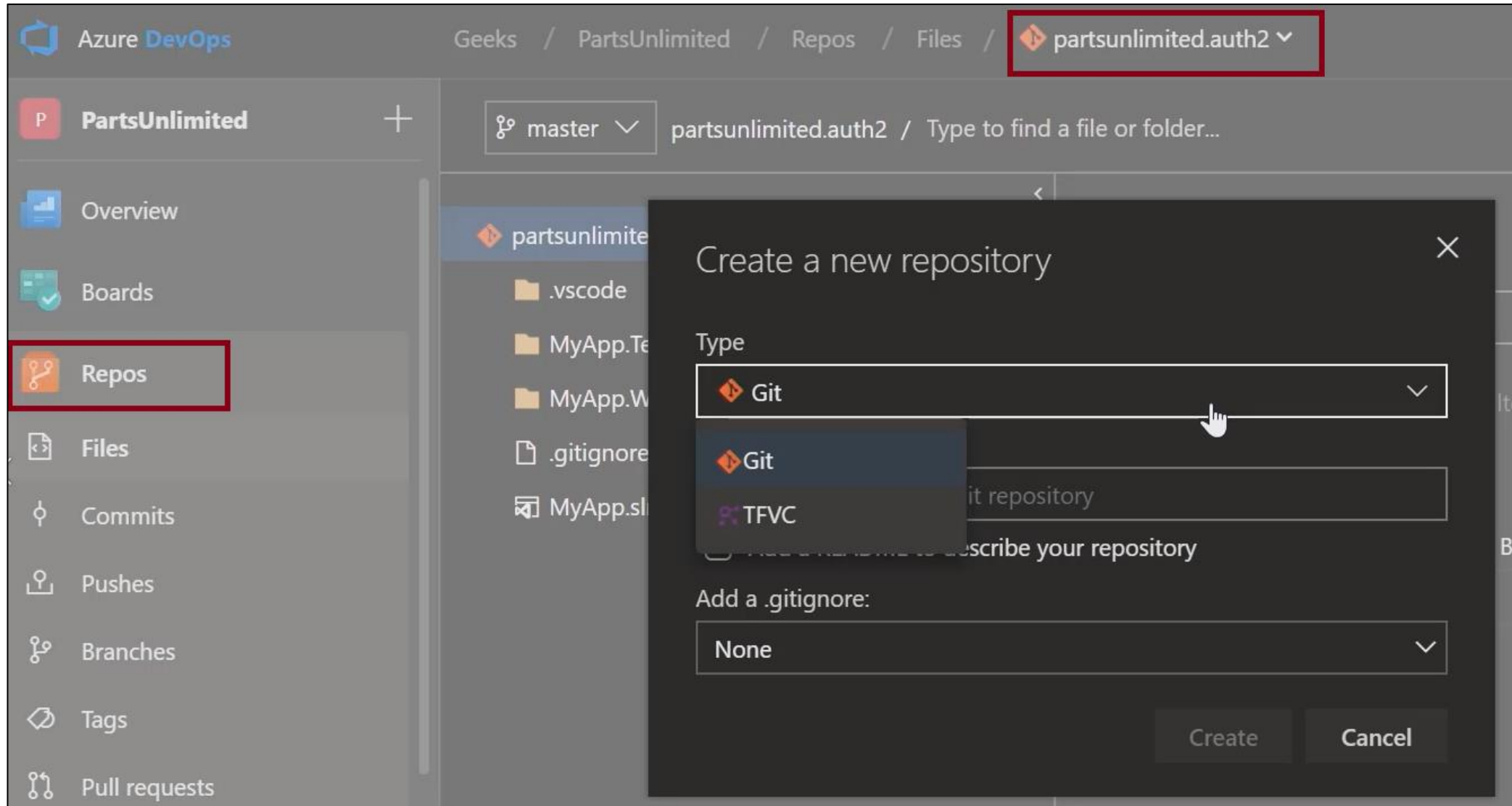
Release cycles



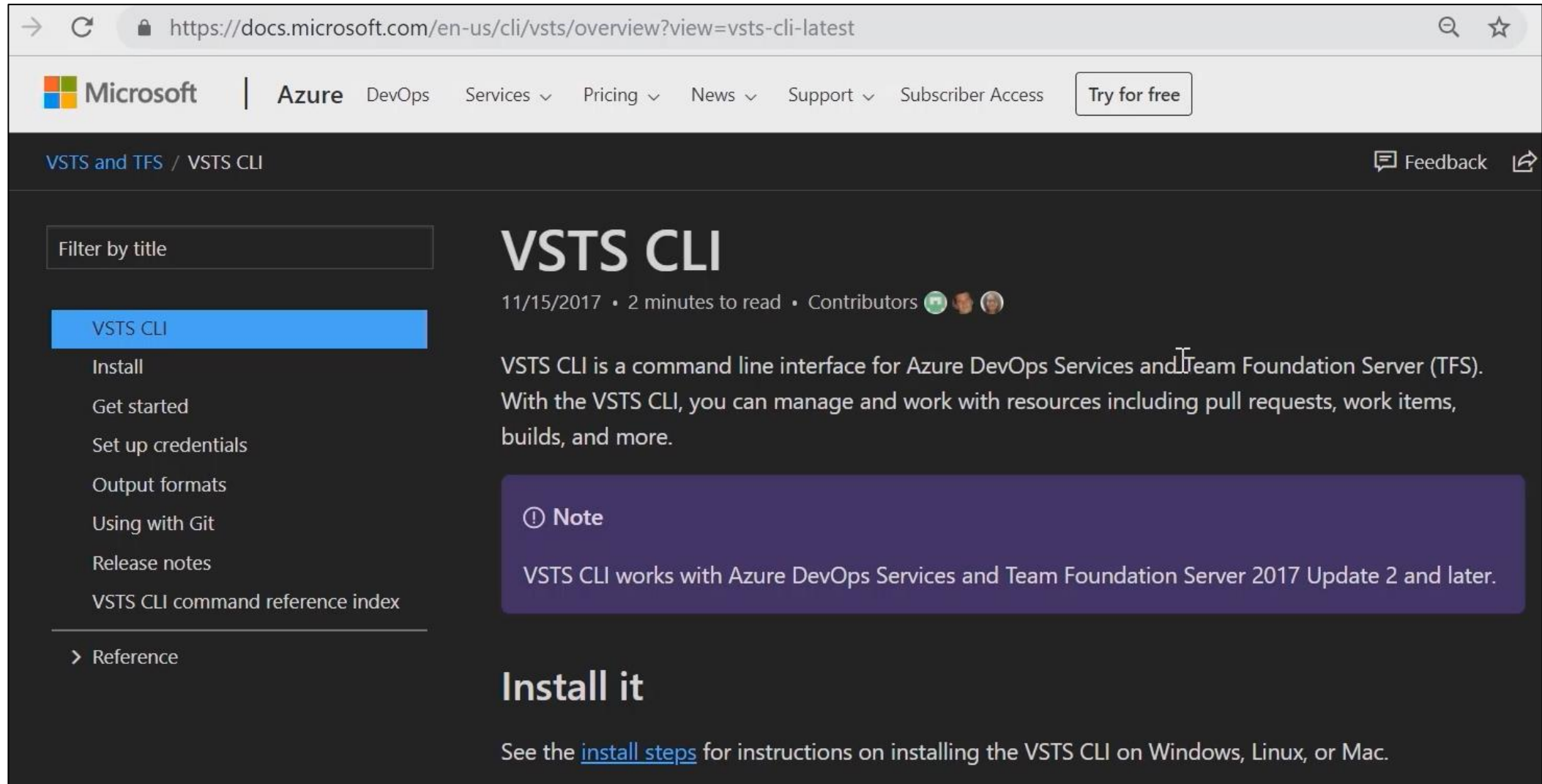
Common Objections to Using Git

- **Overwriting History** – TFVC does not allow this, but with both you still overwrite the code
- **Large Files** - Git works best with repos that are small and that do not contain large files (or binaries); **consider using Git LFS**
- **Large Repos** – This is no longer a blocker, **Virtual File System (VFS) for Git**
- **Git? GitHub?** – There is confusion about the difference
- **Learning Curve** – Some training and instruction will be needed
- **Discussion:** What objections do you have?

Video: Introduction to Azure Repositories




Video: Azure Repositories with VSTS CLI



The screenshot shows a web browser window with the URL <https://docs.microsoft.com/en-us/cli/vsts/overview?view=vsts-cli-latest>. The page features the Microsoft logo and navigation links for Azure, DevOps, Services, Pricing, News, Support, and Subscriber Access, along with a 'Try for free' button. The breadcrumb trail indicates the location: VSTS and TFS / VSTS CLI. A feedback link is also present. On the left, a sidebar with a 'Filter by title' search box lists various topics, with 'VSTS CLI' selected. The main content area is titled 'VSTS CLI' and includes a metadata line: '11/15/2017 • 2 minutes to read • Contributors'. The introductory text states that VSTS CLI is a command line interface for Azure DevOps Services and Team Foundation Server (TFS), used for managing pull requests, work items, builds, and more. A purple 'Note' box specifies that VSTS CLI works with Azure DevOps Services and Team Foundation Server 2017 Update 2 and later. Below this, the 'Install it' section begins with a link to the 'install steps' for Windows, Linux, or Mac.

→ ↻ 🔒 <https://docs.microsoft.com/en-us/cli/vsts/overview?view=vsts-cli-latest> 🔍 ☆

 **Microsoft** | **Azure** DevOps Services ▾ Pricing ▾ News ▾ Support ▾ Subscriber Access [Try for free](#)


[VSTS and TFS](#) / [VSTS CLI](#) [Feedback](#) [Share](#)

Filter by title

- VSTS CLI**
- Install
- Get started
- Set up credentials
- Output formats
- Using with Git
- Release notes
- VSTS CLI command reference index

> Reference

VSTS CLI

11/15/2017 • 2 minutes to read • Contributors 

VSTS CLI is a command line interface for Azure DevOps Services and Team Foundation Server (TFS). With the VSTS CLI, you can manage and work with resources including pull requests, work items, builds, and more.

Note

VSTS CLI works with Azure DevOps Services and Team Foundation Server 2017 Update 2 and later.

Install it

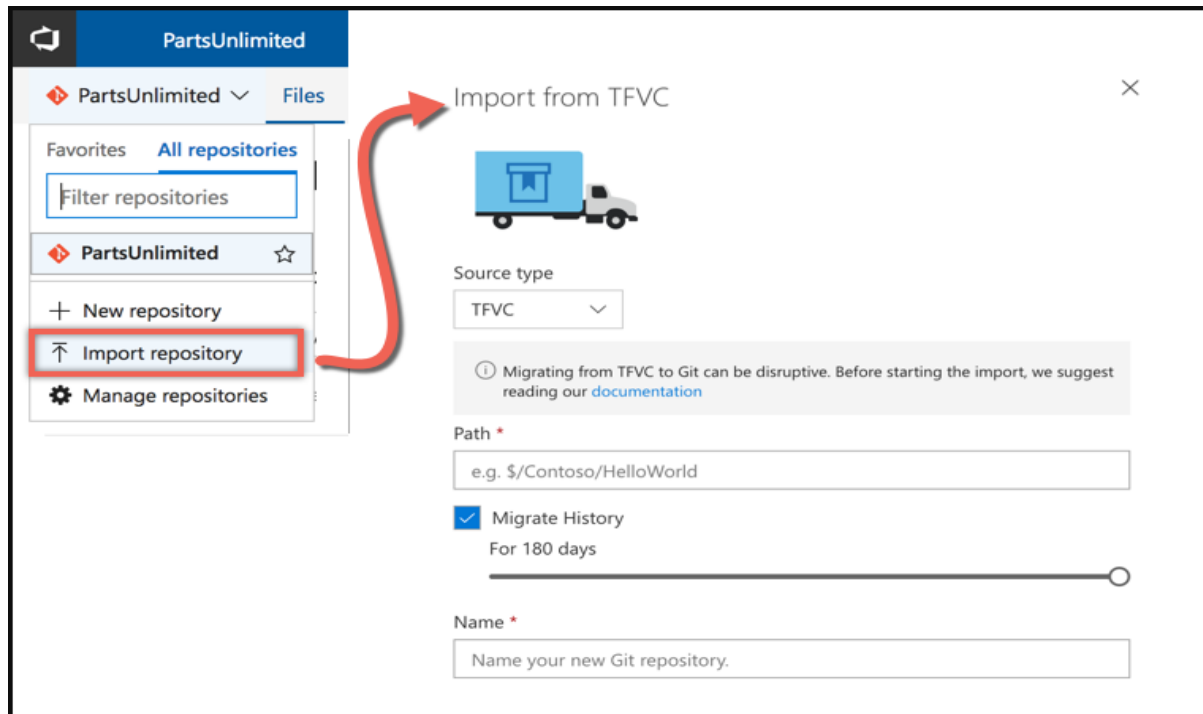
See the [install steps](#) for instructions on installing the VSTS CLI on Windows, Linux, or Mac.

Lab: Version Controlling with Git in Azure Repos

- In this lab, [Version Controlling with Git in Azure Repos](#), you will establish and work with a local Git repository.
 - You will learn how to:
 - Exercise 1: Cloning an existing repository
 - Exercise 2: Save work with commits
 - Exercise 3: Review history
 - Exercise 4: Manage branches from Visual Studio
 - Exercise 5: Managing branches from Azure DevOps
- ✓ Note that you must have already completed the prerequisite labs in the Welcome section.

Migrating from TFVC to Git

- Single branch import



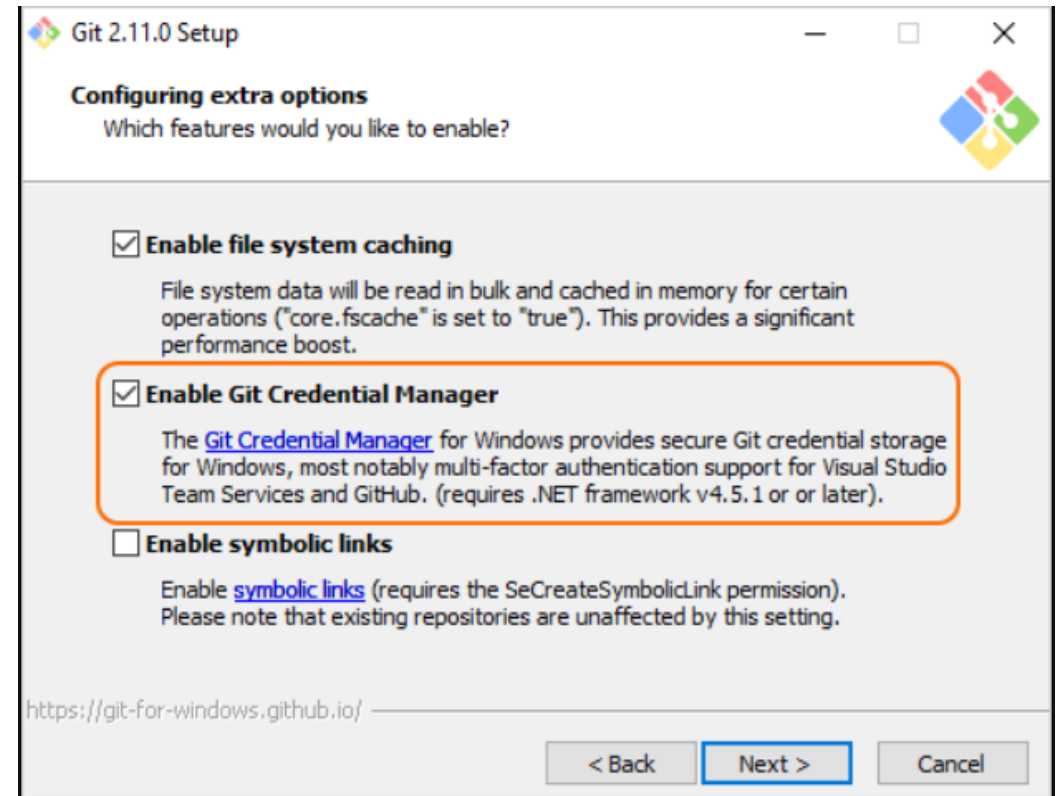
- Full synchronization (Git-tfs)

Video: Migrating from TFVC to Git

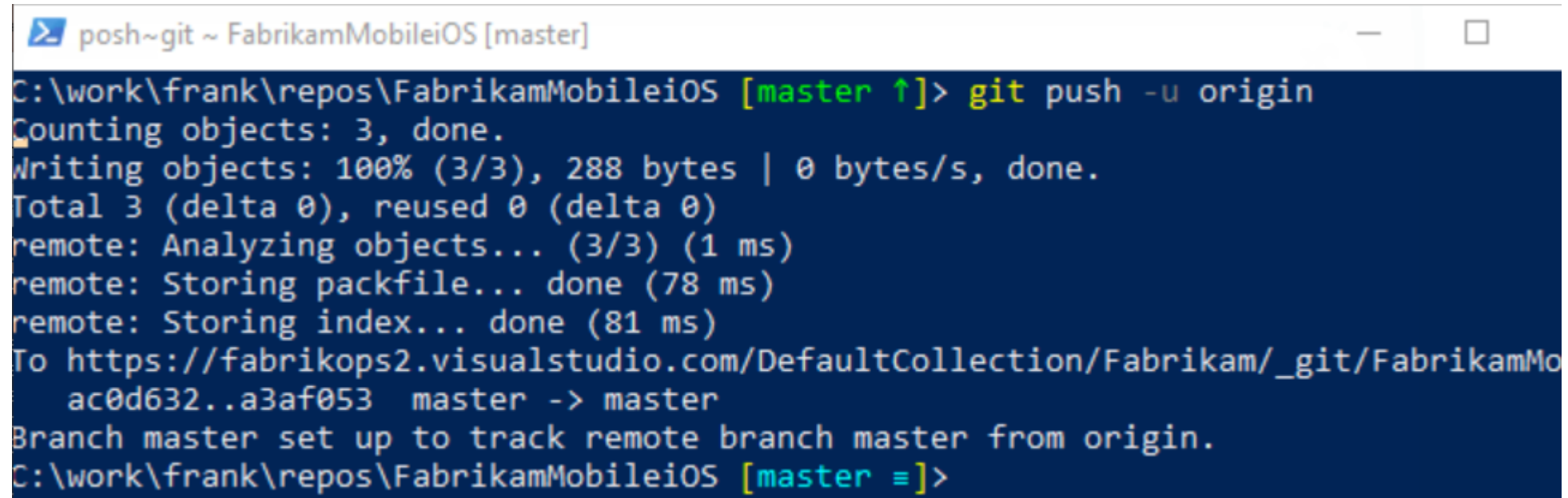
- Migrating the tip
 - Only the latest version of the code
 - History remains on the old server
- Migrating with history
 - Tries to mimic the history in git
- Recommend to migrate the tip, because:
 - History is stored differently – TFVC stores change sets, Git stores snapshots of the repository
 - Branches are stored differently – TFVC branches folders, Git branches the entire repository

Authenticating to Your Git Repos

- Git Credential Manager simplifies authentication with your Azure DevOps Services/TFS Git repos
- Supports MFA and two-factor authentication with GitHub repositories
- You can also configure your IDE with a Personal Access Token or SSH to connect with your repos



Demonstration: Git Credential Manager

A terminal window with a dark blue background and white text. The title bar shows 'posh~git ~ FabrikamMobileiOS [master]'. The command prompt is 'C:\work\frank\repos\FabrikamMobileiOS [master ↑]>'. The user enters 'git push -u origin'. The output shows 'Counting objects: 3, done.', 'Writing objects: 100% (3/3), 288 bytes | 0 bytes/s, done.', 'Total 3 (delta 0), reused 0 (delta 0)', 'remote: Analyzing objects... (3/3) (1 ms)', 'remote: Storing packfile... done (78 ms)', 'remote: Storing index... done (81 ms)', 'To https://fabrikops2.visualstudio.com/DefaultCollection/Fabrikam/_git/FabrikamMo', 'ac0d632..a3af053 master -> master', 'Branch master set up to track remote branch master from origin.', and the prompt 'C:\work\frank\repos\FabrikamMobileiOS [master =]>'.

```
posh~git ~ FabrikamMobileiOS [master]
C:\work\frank\repos\FabrikamMobileiOS [master ↑]> git push -u origin
Counting objects: 3, done.
Writing objects: 100% (3/3), 288 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
remote: Analyzing objects... (3/3) (1 ms)
remote: Storing packfile... done (78 ms)
remote: Storing index... done (81 ms)
To https://fabrikops2.visualstudio.com/DefaultCollection/Fabrikam/_git/FabrikamMo
    ac0d632..a3af053  master -> master
Branch master set up to track remote branch master from origin.
C:\work\frank\repos\FabrikamMobileiOS [master =]>
```

- When you first connect to a Git repo the credential manager prompts for your Microsoft Account or Azure Active Directory credentials
- Once authenticated, the credential manager creates and caches a personal access token for future connections to the repo

How to Structure Git Repos - Mono vs Multi Repos

A repository is a place where the history of your work is stored

Advantages

Mono-repo - source code is kept in a single repository

- Clear ownership
- Better scale
- Narrow clones

Multiple-repo – each project has its own repository

- Better developer testing
- Reduced code complexity
- Effective code reviews
- Sharing of common components
- Easy refactoring

Video - Git Branching Workflows

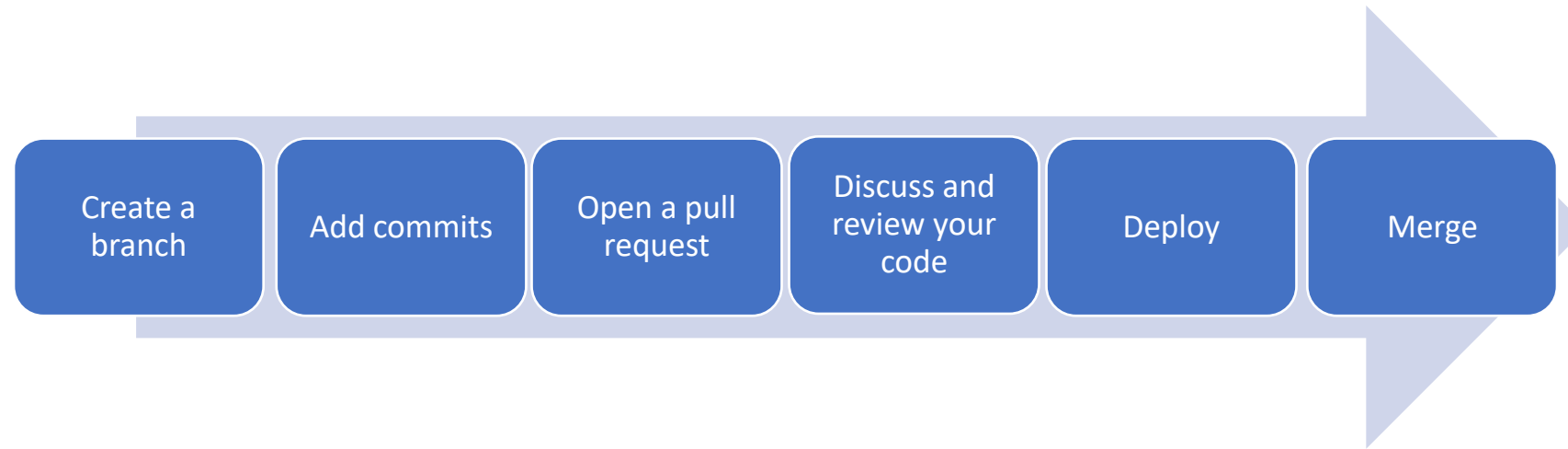
- **Scenario 1** - Branching will help you do more
- **Scenario 2** - Branching will help with communication and trust issues
- **Scenario 3** - Branching can help with shifting priorities

✓ Don't let flexibility in Git lead you to technical debt

Branching Workflow Types

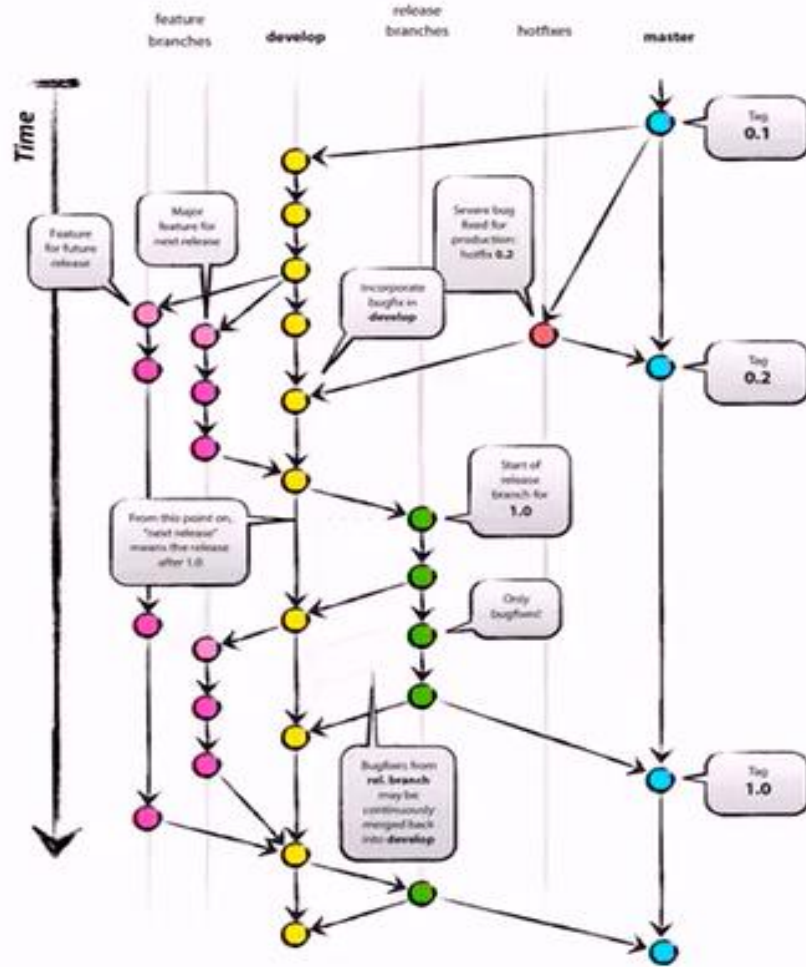
- **Feature branching** - all feature development should take place in a dedicated branch instead of the master branch
- **GitFlow branching** - a strict branching model designed around the project release
- **Forking Workflow** - every developer uses a server-side repository
- Evaluate the workflow
 - Does this workflow scale with team size?
 - Is it easy to undo mistakes and errors with this workflow?
 - Does this workflow impose any new unnecessary cognitive overhead to the team?

Feature Branch Workflow



- All feature development should take place in a dedicated branch instead of the master branch
- Encapsulating feature development leverages pull requests, which are a way to initiate discussions around a branch
- Share a feature with others without touching any official code

Video - GitFlow Improving the Flow of Code



Git Flow...

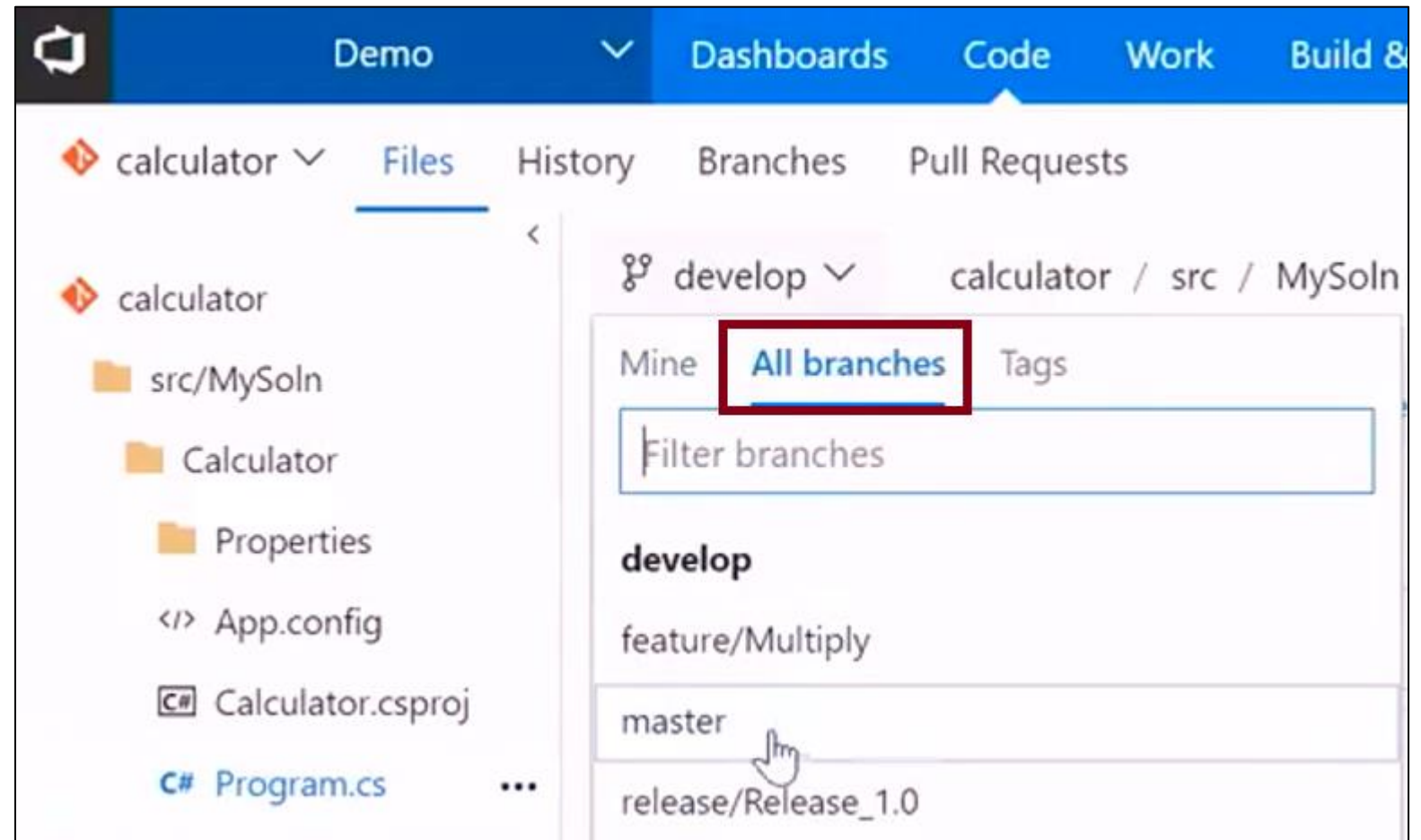
- Introduced by Vincent Driessen in 2010
- Prescriptive process that defines a structure & approach to branching & merging for,
 - Adding new features
 - Fixing bugs
 - Preparing a release
 - Deploying to production
 - Applying hotfixes
- The git-flow process is designed largely around the "release."
- It is best suited for organizations that need multiple branches to mature code quality.



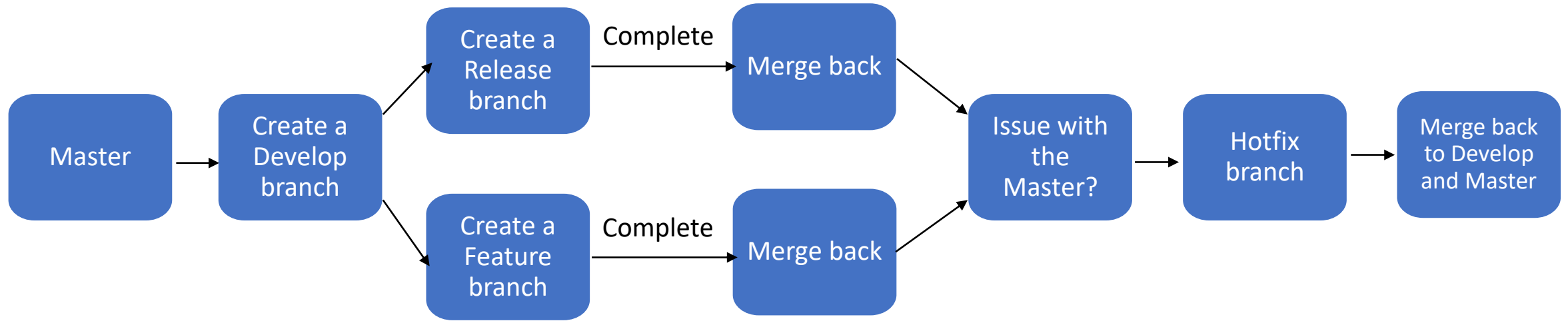
<http://nvie.com/posts/a-successful-git-branching-model/>

Video – Implementing GitFlow

- Allows you to work on multiple features in parallel
- You can release just the work that is completed



GitFlow Branch Workflow

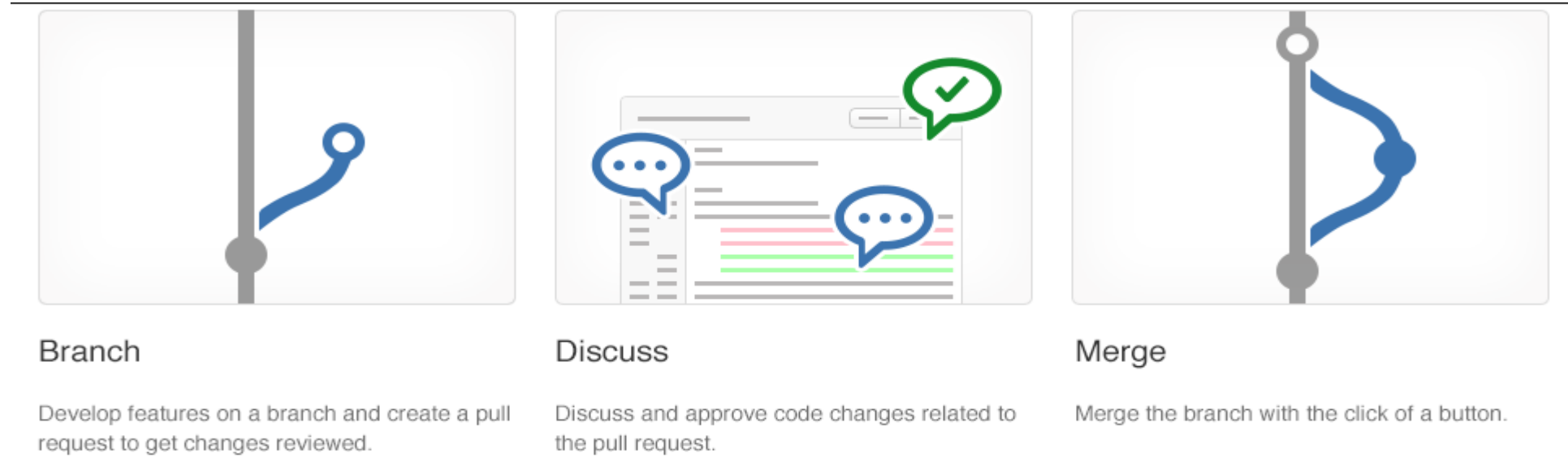


- GitFlow is great for a release-based software workflow
- GitFlow offers a dedicated channel for hotfixes to production

Forking Branch Workflow

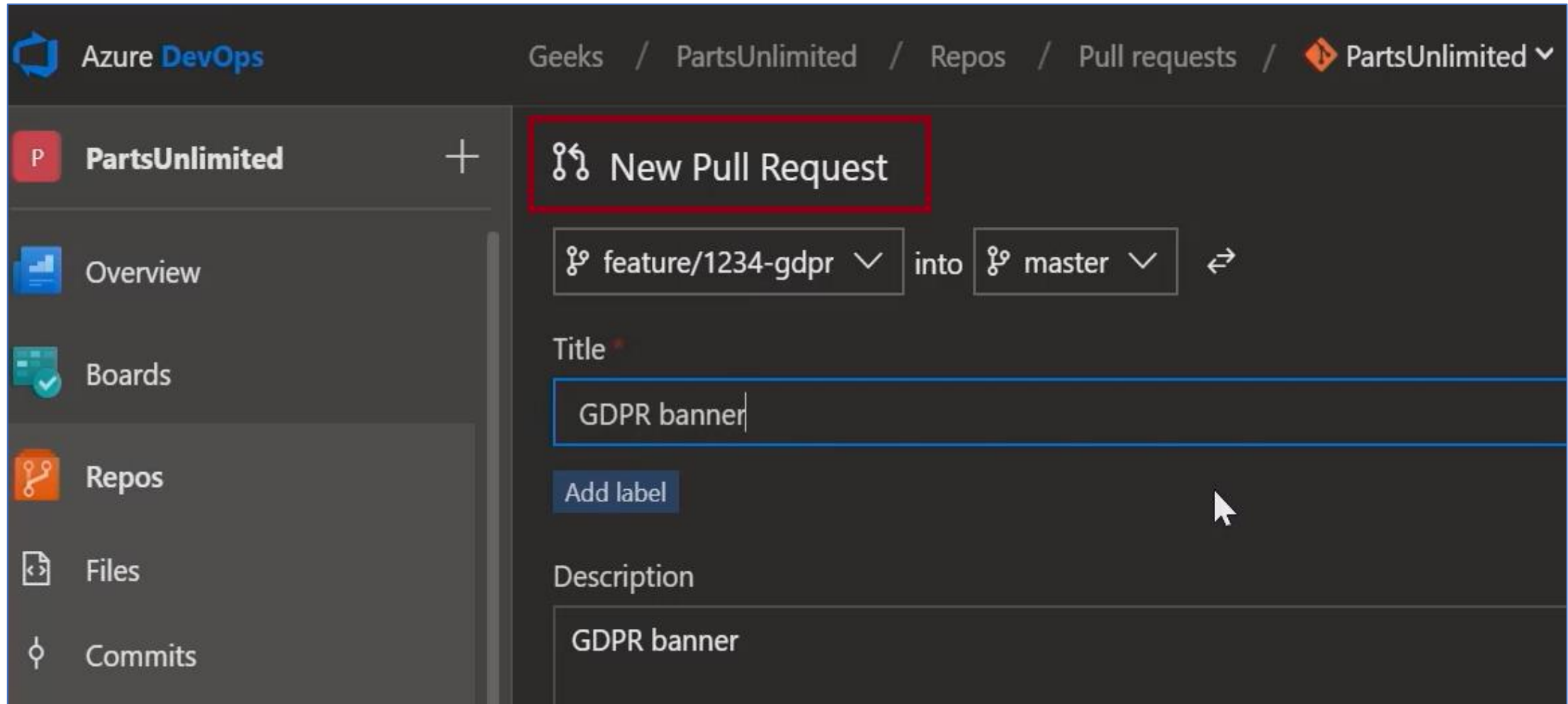
- Forking Branch workflow gives every developer their own server-side repository
 - Each contributor has not one, but two Git repositories: a private local one and a public server-side one
 - Most often seen in public open source projects
 - Contributions can be integrated without the need for everybody to push to a single central repository
 - Typically follows a branching model based on the GitFlow Workflow
- ✓ Forked repositories use the standard git **clone** command

Collaborating with Pull Requests



- Pull requests let you tell others about changes
- Collaboration using the Shared Repository Model
- Review and merge your code in a single collaborative process
- Be sure to provide good feedback and protect branches with policies

Collaborating with Pull Requests

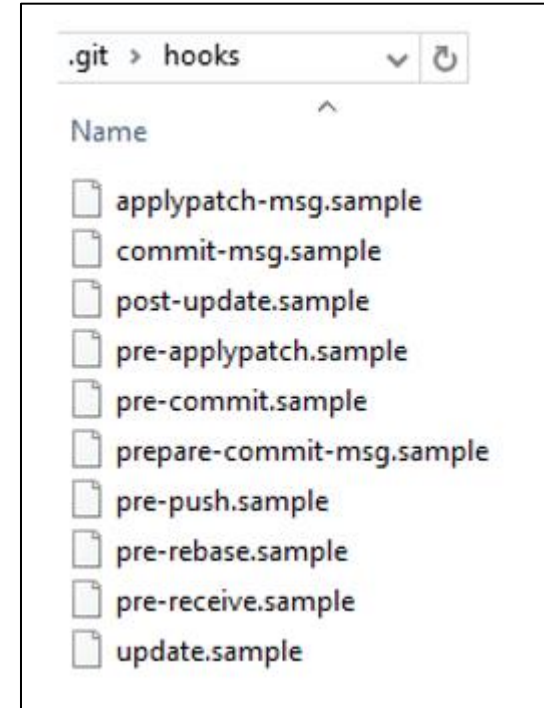


Lab: Code Review with Pull Requests

- In this lab, [Version Controlling with Git in Azure Repos](#), you will work branching and merging. You will learn how to:
 - Exercise 6: Working with pull requests
 - Exercise 7: Managing repositories
- ✓ Note that you must have already completed the prerequisite labs in the Welcome section.

Why Care about GitHooks?

- A mechanism that allows arbitrary code to be run before, or after, certain Git lifecycle events occur
- Use GitHooks to enforce policies, ensure consistency, and control your environment
- Can be either client-side or server-side



- ✓ Using GitHooks is like having little robot minions to carry out your every wish

Video: GitHooks in Action

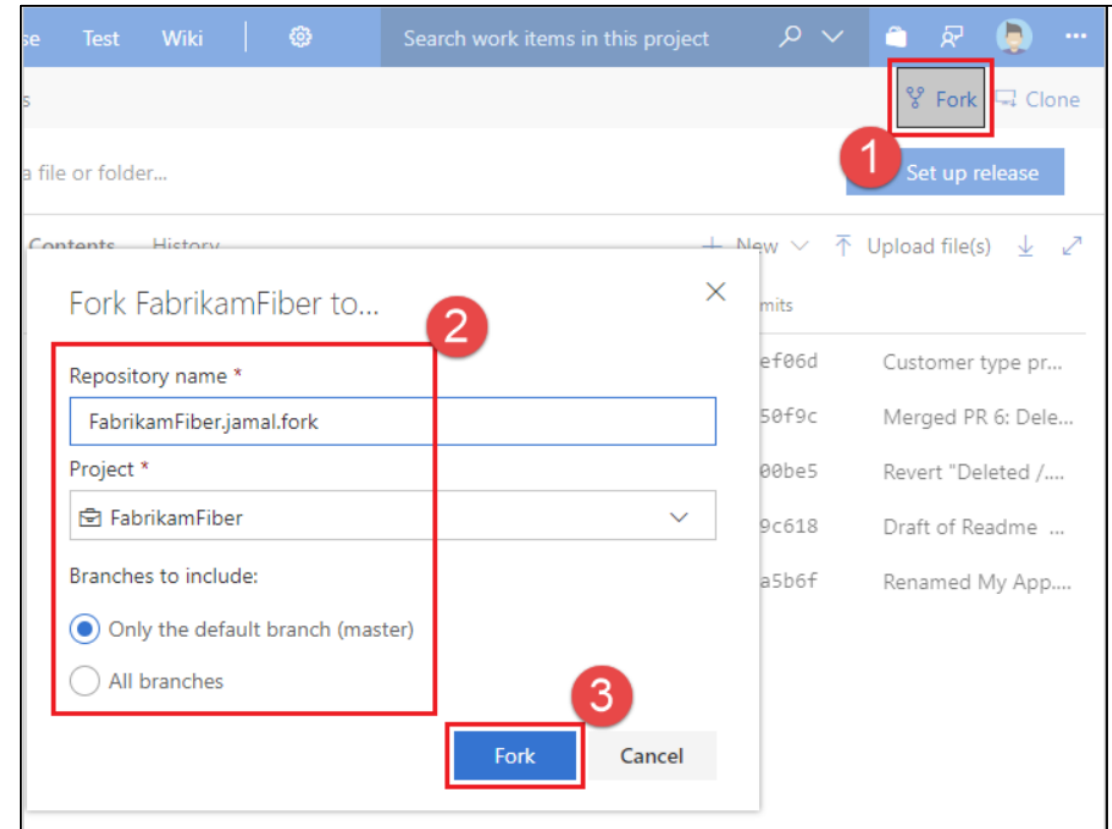
- Will my code:
 - Break other code?
 - Introduce code quality issues?
 - Drop the code coverage?
 - Take on a new dependency?
- Will the incoming code:
 - Break my code?
 - Introduce code quality issues?
 - Drop the code coverage?
 - Take on a new dependency?

Fostering Internal Open Source

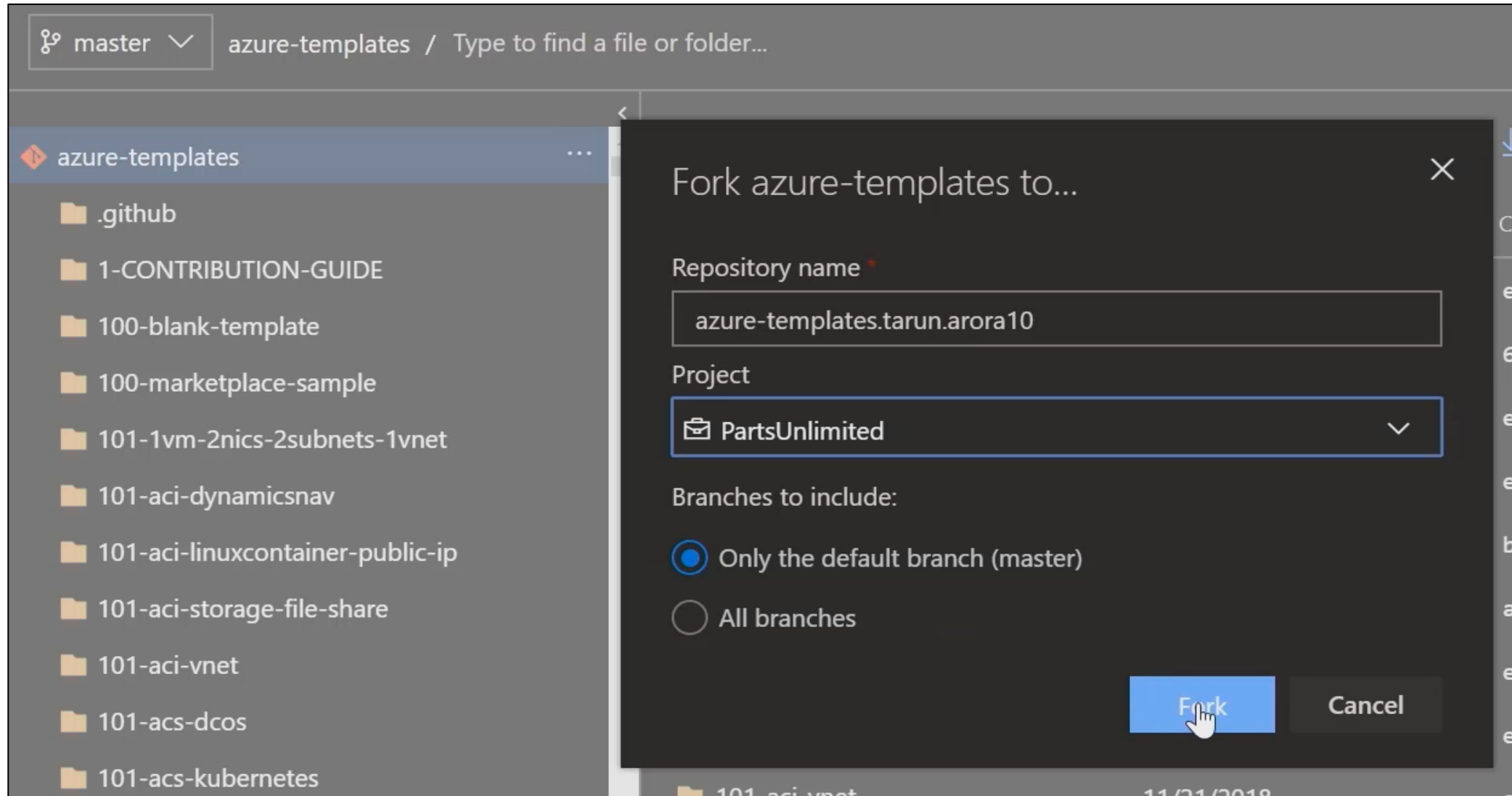
- Fork-based pull request workflows allows anybody to contribute
- **Inner Source** brings all the benefits of open source software development inside your firewall
- We recommend the forking workflow for large numbers of casual or occasional committers

Implementing the Fork Workflow

- What's in a fork?
- Sharing code between forks
- Choosing between branches and forks
- The forking workflow
 - Create a fork
 - Clone it locally
 - Make your changes locally and push them to a branch
 - Create and complete a PR to upstream
 - Sync your fork to the latest from upstream



Video: Inner Source with Forks



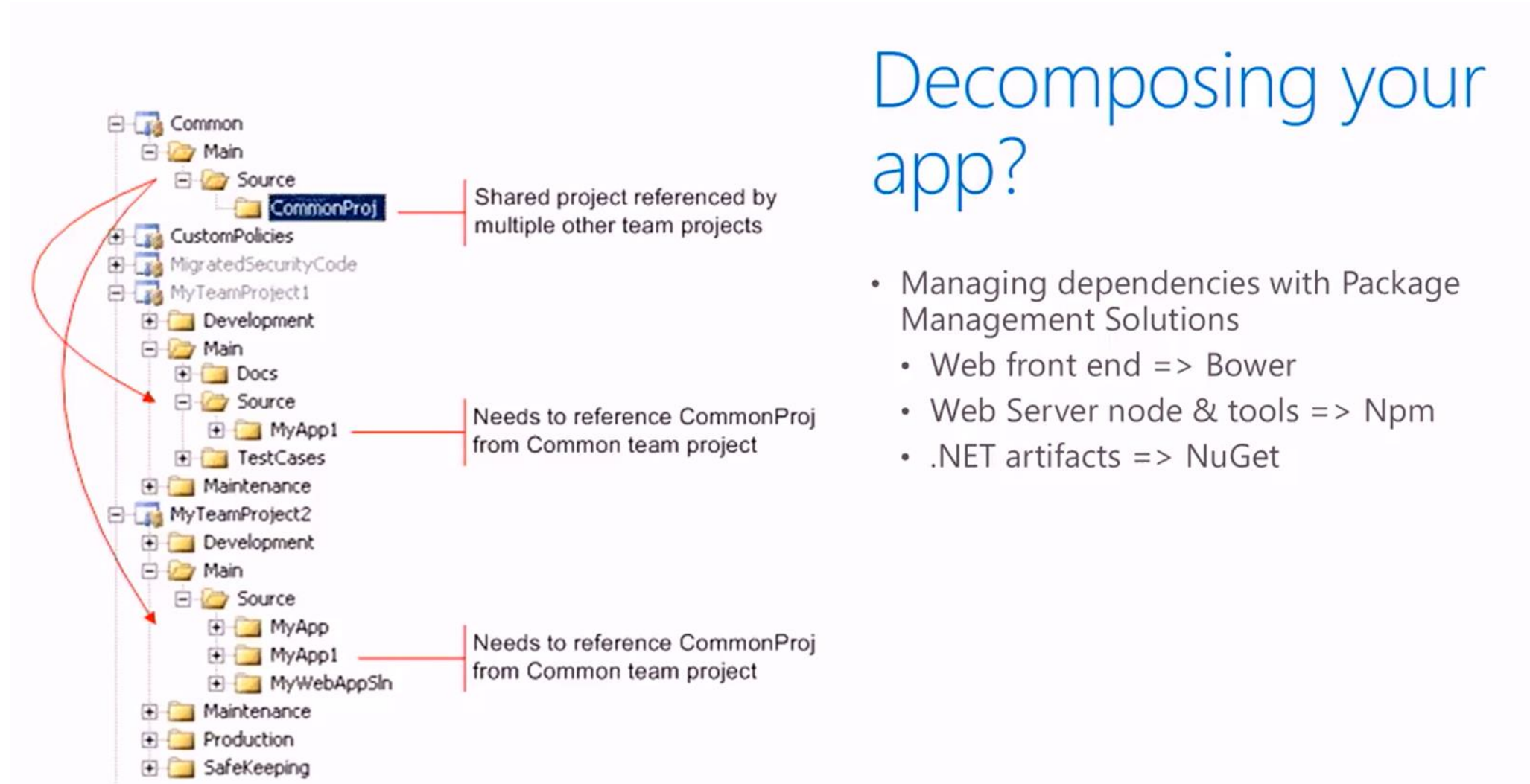
Git Versioning

- Semantic Versioning is all about releases, not builds
- GitVersion is a tool to help you achieve Semantic Versioning
- GitVersion calculates the version based on:
 - the nearest tag
 - the commit messages between this commit and the nearest tag
 - the name of the branch

Create version numbers using the approach **MAJOR.MINOR.PATCH**

- **MAJOR** – Increment when you make incompatible API changes
- **MINOR** – Increment when you add new functionality that is backward compatible
- **PATCH** – Increment when you add bug fixes that are backward compatible

Video: GitVersion



Decomposing your app?

- Managing dependencies with Package Management Solutions
 - Web front end => Bower
 - Web Server node & tools => Npm
 - .NET artifacts => NuGet

Public Projects

- Azure DevOps Team Projects can be public – no Microsoft account required
- Public projects enables anonymous users to:
 - Browse the code base, download code, view commits, branches, and pull requests
 - View and filter work items
 - View a project page or dashboard
 - View the project Wiki
 - Perform semantic search of the code or work items
- Private projects require users to be granted access to the project and signed in to access the services

Video: Public Projects

Pricing and setup

Free

Free for public and private repositories

\$0

Add parallel jobs

Add parallel jobs for private repositories

\$40

per parallel job
/ month

Azure Pipelines

Free

Free for public and private repositories

- ✓ Linux, macOS, and Windows
- ✓ 10 free parallel jobs for public repositories
- ✓ Unlimited minutes for public repositories
- ✓ 1 free parallel job for private repositories (1,800 minutes per month)

Account: [tarunaroraonline](#) ▼

Install it for free

Next: Confirm your installation location.

Storing Files in Git

- Use a package management system for DLLs, library files, and other dependent files
- Don't commit the binaries, logs, tracing output or diagnostic data from your builds
- Don't commit large, frequently updated binary assets
- Use diffable plain text formats, such as JSON, for configuration information

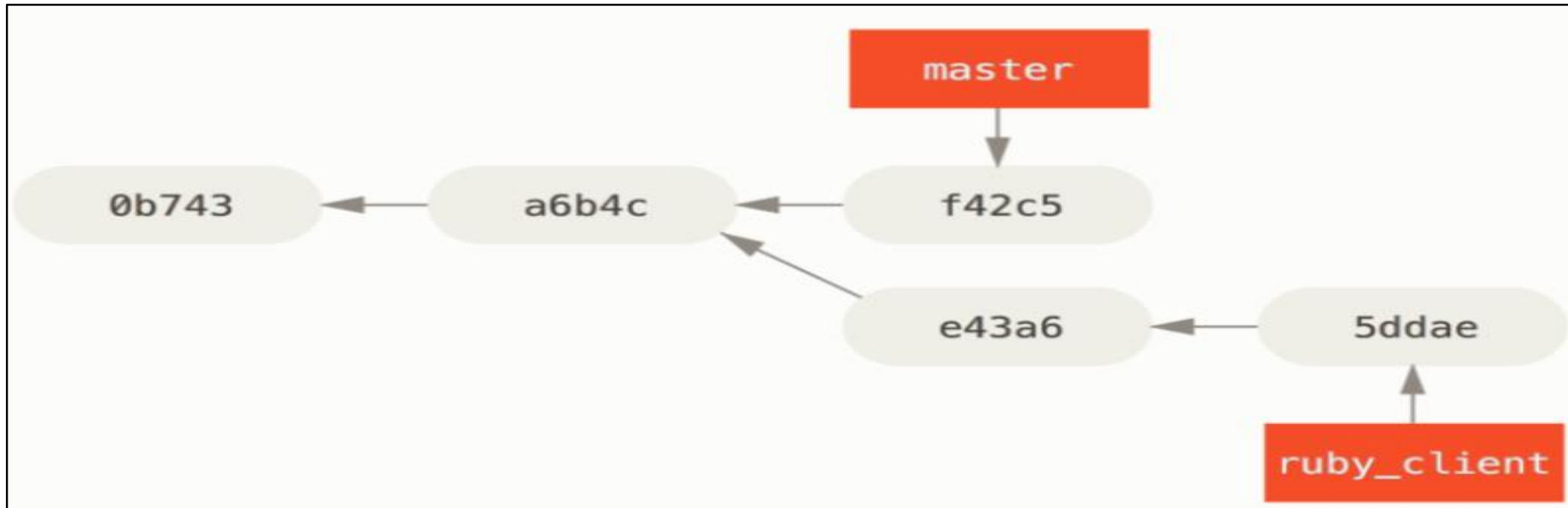
Git Large File Storage (LFS)

- Use Git LFS for source files with large differences between versions and frequent updates, you to manage these file types
- Benefits
 - Familiar end to end Git workflow
 - LFS files can be as big as you need them to be
 - Supports file locking for assets like videos, sounds, and game maps
 - Git LFS is fully supported and free in Azure DevOps Services
- Some limitations
 - Everyone must install the Git LFS client and understand its tracking configuration
 - Git cannot merge the changes from two different versions of a binary file even if both versions have a common parent
 - Currently does not support using SSH in repos with Git LFS tracked files

Git Cherry-Pick

A cherry-pick in Git is like [a rebase for a single commit](#). It takes the patch that was introduced in a commit and tries to reapply it on the branch you're currently on.

- you have several commits on a topic branch and you want to integrate only one of them
- you only have one commit on a topic branch and you'd prefer to cherry-pick it rather than run rebase. For example, suppose you have a project that looks like this:



Git Cherry-Pick

If you want to pull commit e43a6 into your master branch, you can run:

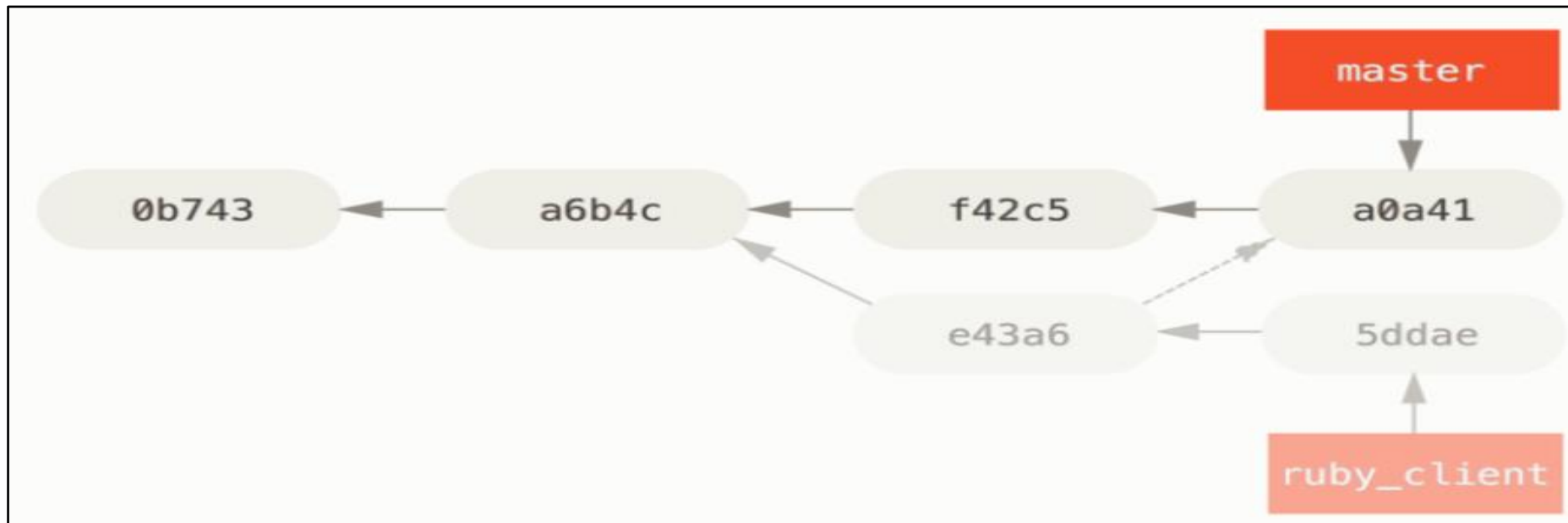
```
$ git cherry-pick e43a6
```

Finished one cherry-pick.

[master]: created a0a41a9: "More friendly message when locking the index fails."

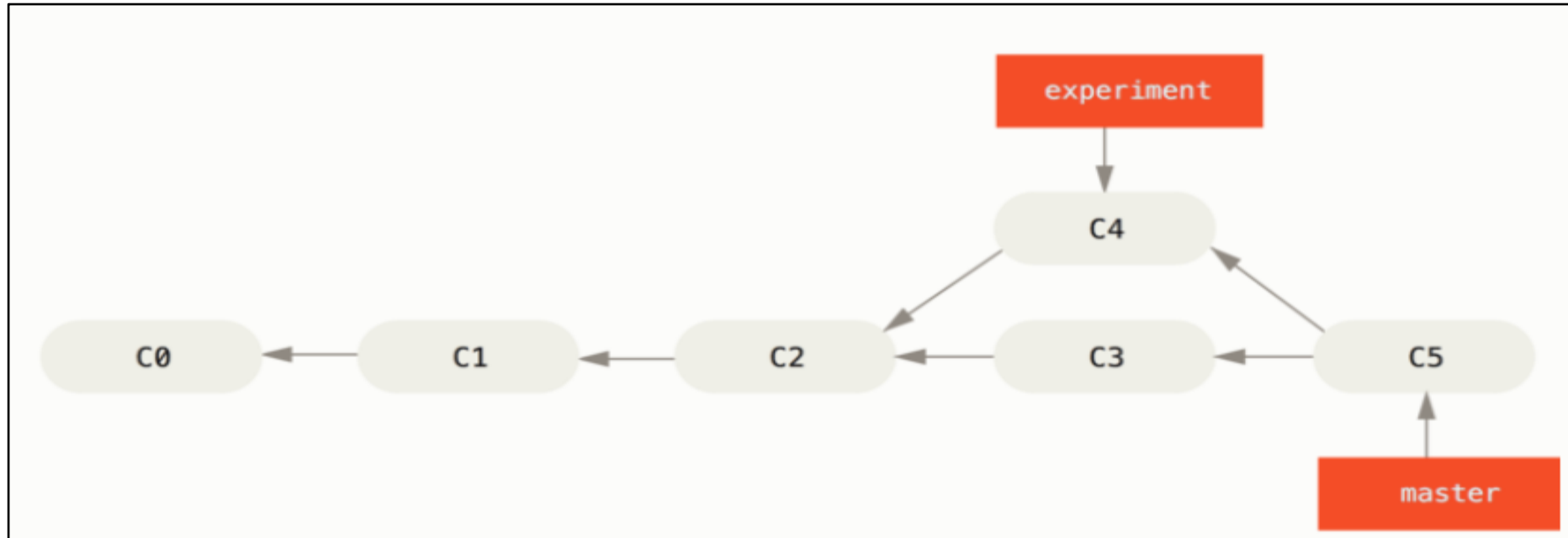
3 files changed, 17 insertions(+), 3 deletions(-)

This pulls the same change introduced in e43a6, but you get a new commit SHA-1 value, because the date applied is different. Now your history looks like this:



Git Rebase

In Git Rebase you consider the patch that was introduced in a commit and try to reapply it on the branch you're currently on.



However, there is another way: you can take the patch of the change that was introduced in C4 and reapply it on top of C3. In Git, this is called rebasing.

Git Rebase

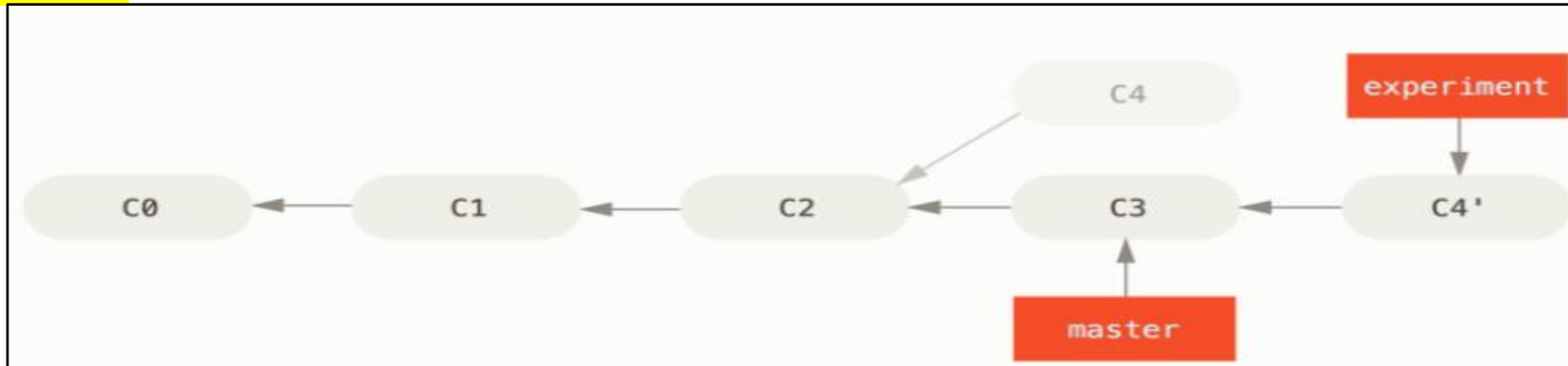
For this example, you would check out the experiment branch, and then rebase it onto the master branch as follows:

```
$ git checkout experiment
```

```
$ git rebase master
```

First, rewinding head to replay your work on top of it...

Applying: added staged command



```
$ git checkout master
```

```
$ git merge experiment
```

