Microsoft

Module 2: Scaling Git for Enterprise DevOps

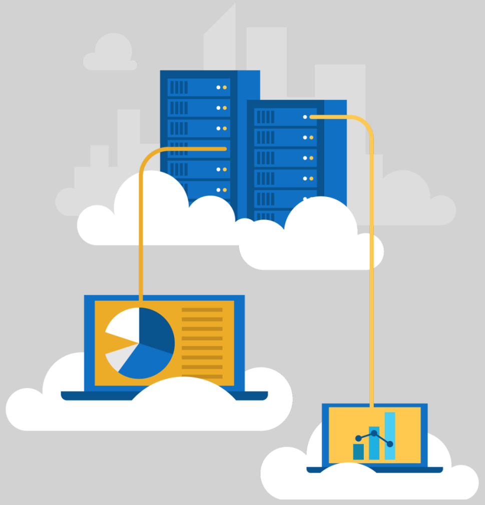
Lesson 01: How to Structure Your Git Repo

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

Lesson 02: Git Branching Workflows



Git Branching Workflows

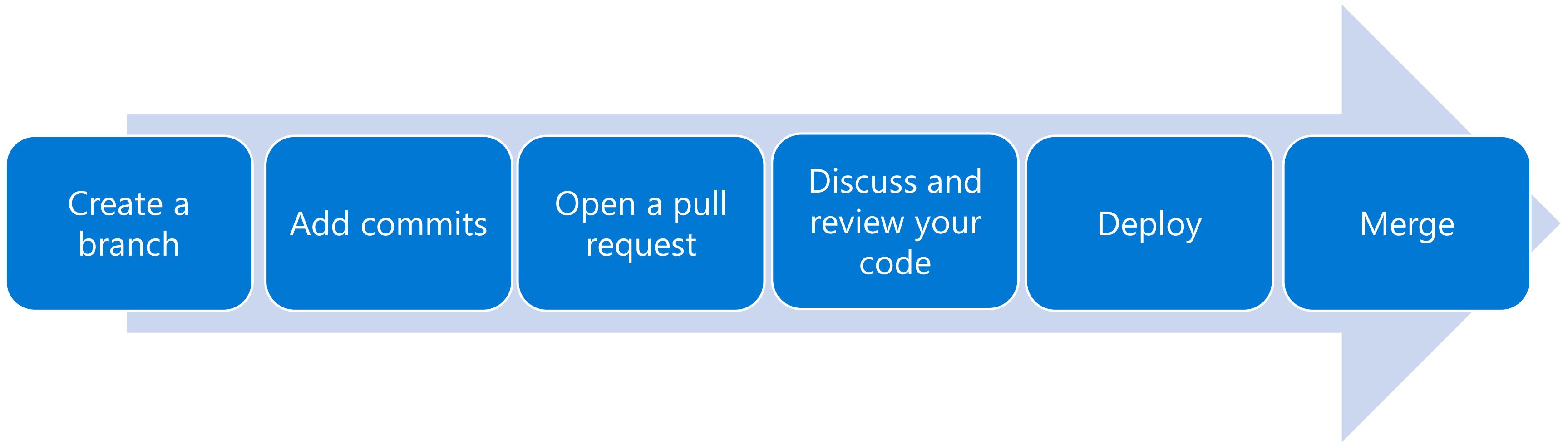
- Scenario 1 - Branching will help you do more
- Scenario 2 - Branching will help with communication and trust issues
- Scenario 3 - Branching can help with shifting priorities

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

Branching Workflow Types

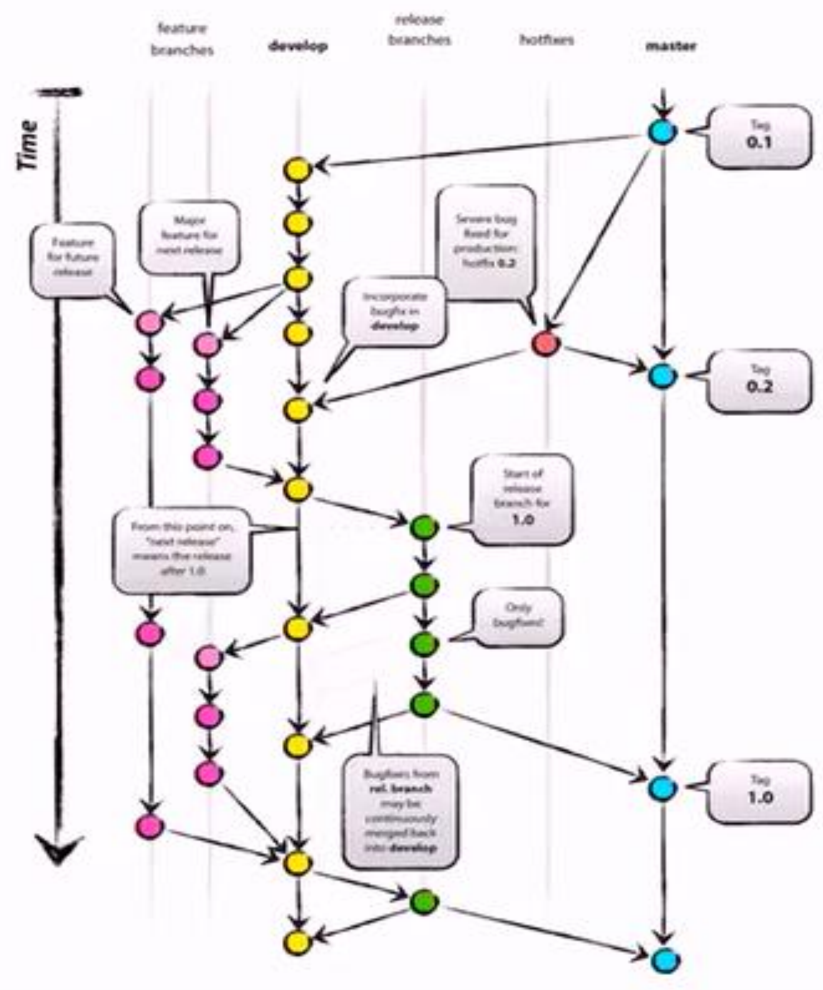
- Feature branching - all feature development should take place in a dedicated branch instead of the master branch
- Gitflow branching - a strict branching model designed around the project release
- Forking Workflow - every developer uses a server-side repository
- Evaluate the workflow
    - Does this 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

GitFlow Improving the Flow of Code


feature branches
develop
release branches
hotfixes
master
Time
Tag 0.1
Feature for future release
Major feature for next release
Severe bug fixed for production: hotfix 0.2
Incorporate bugfix in develop
Tag 0.2
Start of release branch for 1.0
From this point on, "next release" means the release after 1.0
Only bugfixes!
Bugfixes from rel. branch may be continuously merged back into develop
Tag 1.0

Git Flow…

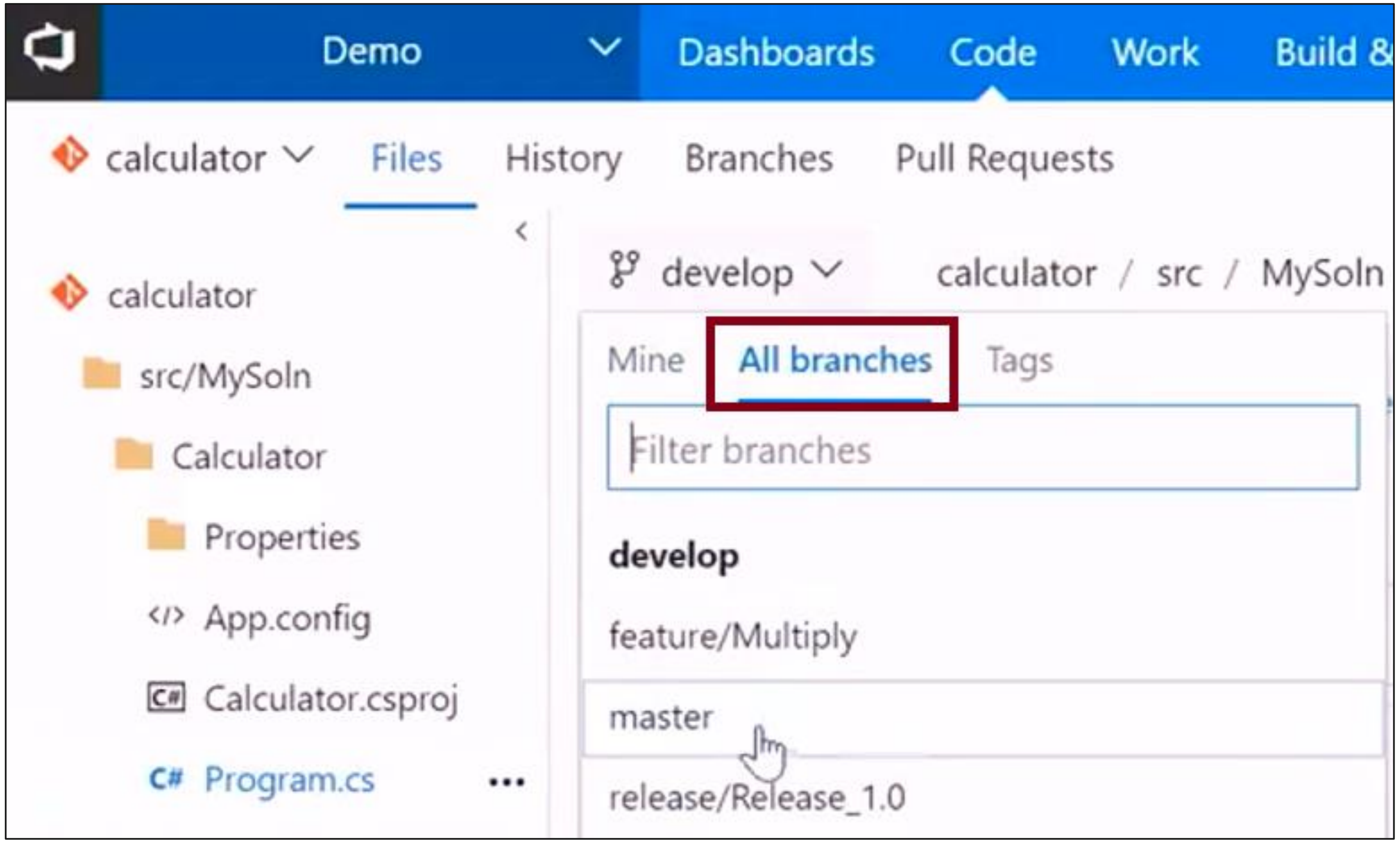
- Introduced by Vincent Driessen in 2010
- Prescriptive process that defines a structure & approach to branching & merging for,
  - Adding new features
  - Fixing bugs
  - Preparing a release
  - Deploying to production
  - Applying hotfixes
- The git-flow process is designed largely around the "release."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

Microsoft

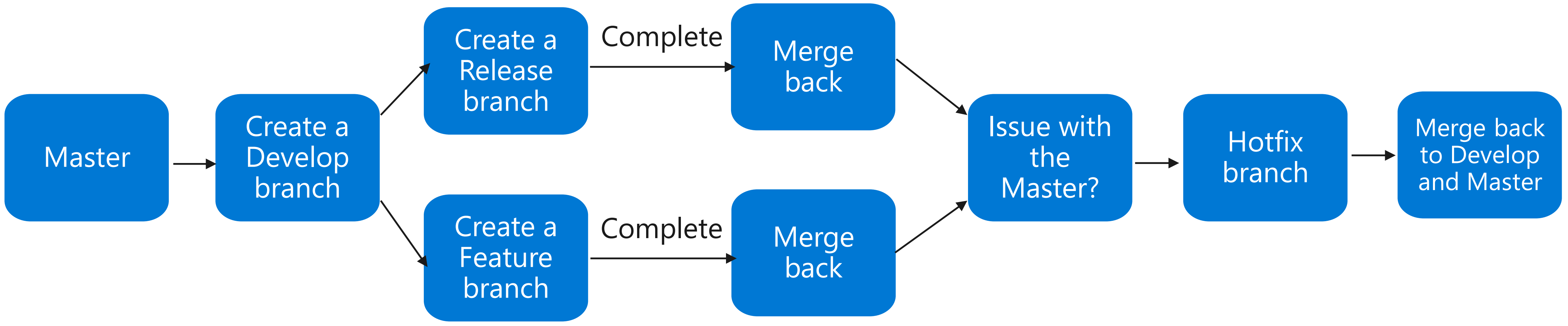
http://nvie.com/posts/a-successful-git-branching-model/

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

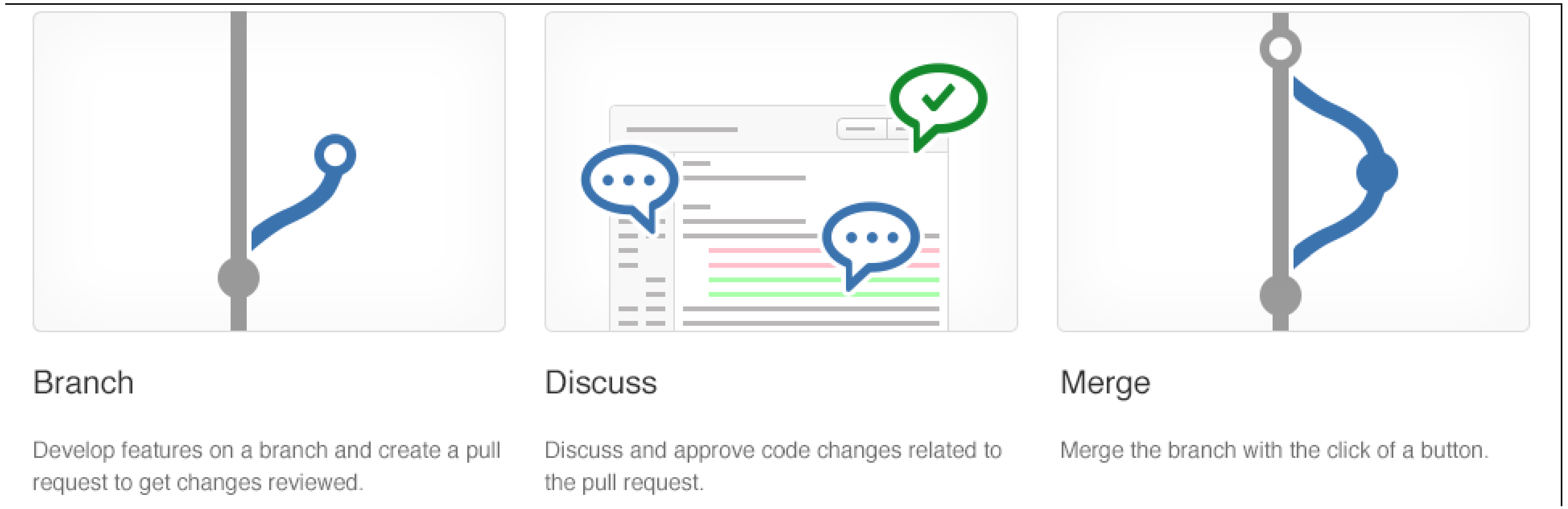
Forking Branch Workflow

- 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

Lesson 03: Collaborating with Pull Requests

Collaborating with Pull Requests



Branch

Develop features on a branch and create a pull request to get changes reviewed.

Discuss

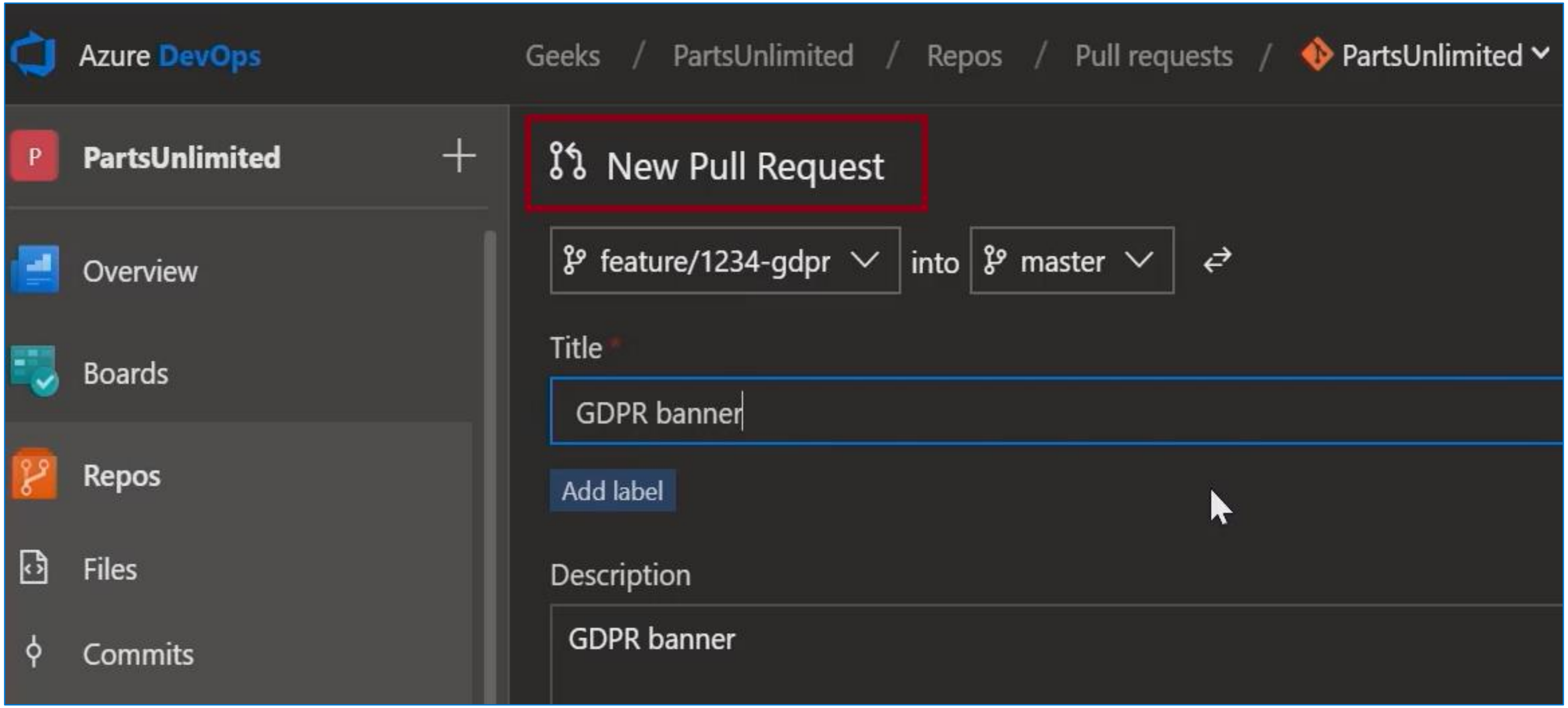
Discuss and approve code changes related to the pull request.

Merge

Merge the branch with the click of a button.

- 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



Title

GDPR banner

Description

GDPR banner

Lab: Code Review with Pull Requests

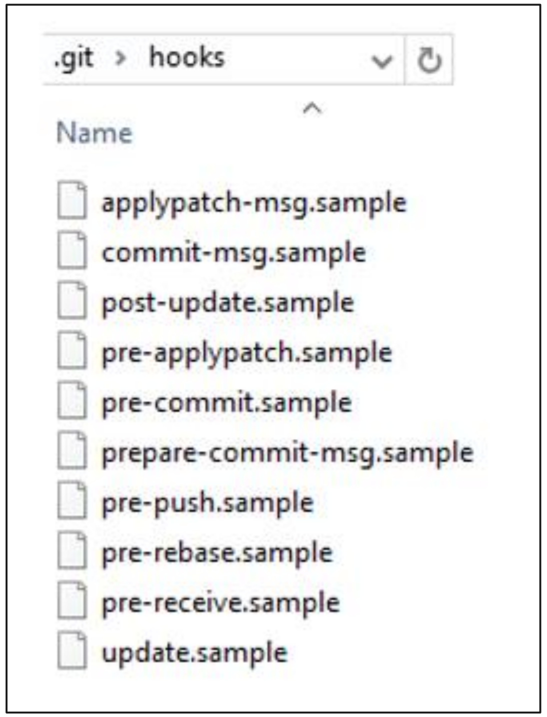
- In this lab, Version Controlling with Git in Azure Repos, you will work branching and merging. You will learn how to:
  - Exercise 6: Working with pull requests
  - Exercise 7: Managing repositories

✓ Note that you must have already completed the prerequisite labs in the Welcome section.

Lesson 04: Why Care about GitHooks

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



applypatch-msg.sample
commit-msg.sample
post-update.sample
pre-applypatch.sample
pre-commit.sample
prepare-commit-msg.sample
pre-push.sample
pre-rebase.sample
pre-receive.sample
update.sample

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

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?

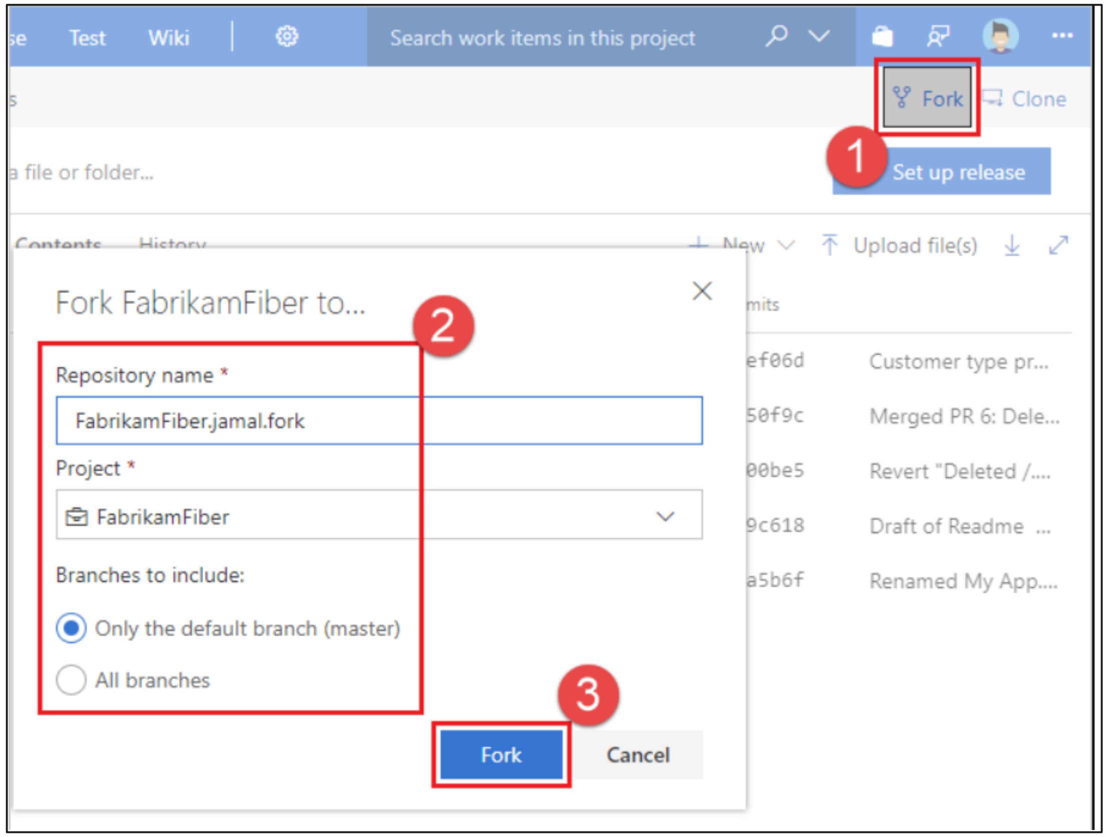
Lesson 05: Fostering Internal Open Source

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



Fork FabrikamFiber to...

Repository name *: FabrikamFiber.jamal.fork

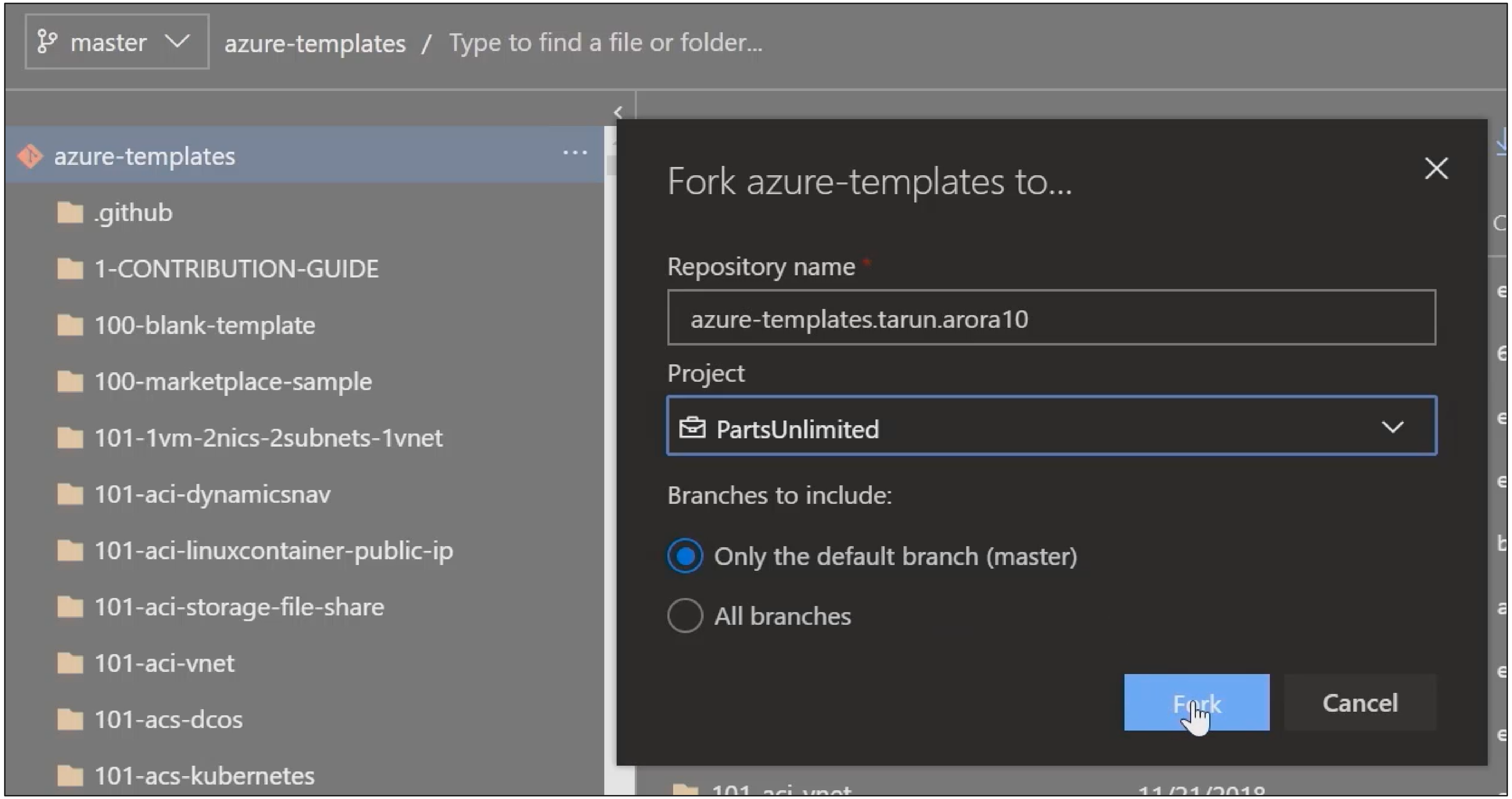
Project *: FabrikamFiber

Branches to include:

- [✓ Only the default branch (master)]
- All branches

ef06d	Customer type pr...
50f9c	Merged PR 6: Dele...
00be5	Revert "Deleted /....
9c618	Draft of Readme ...
a5b6f	Renamed My App....

Inner Source with Forks



Fork azure-templates to...

Repository name *

azure-templates.tarun.arora10

Project

PartsUnlimited

Branches to include:

[✓ Only the default branch (master)]

All branches

Lesson 06: Git Version

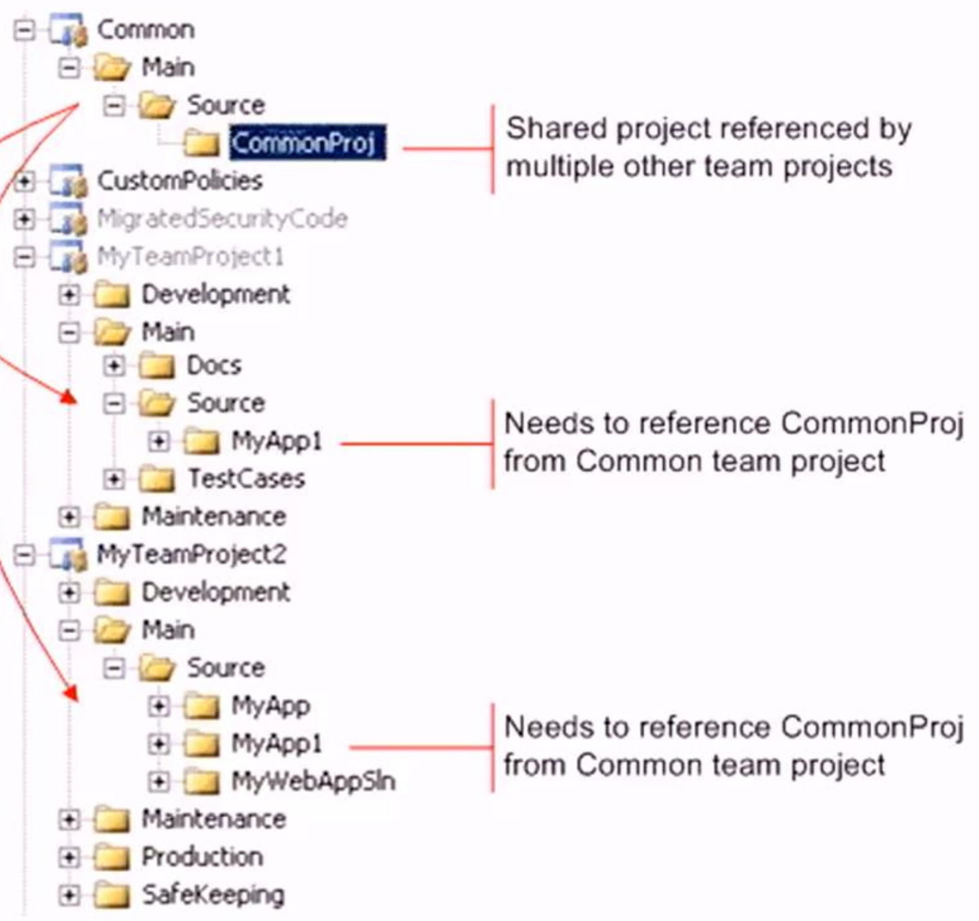
Git Versioning

- 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?

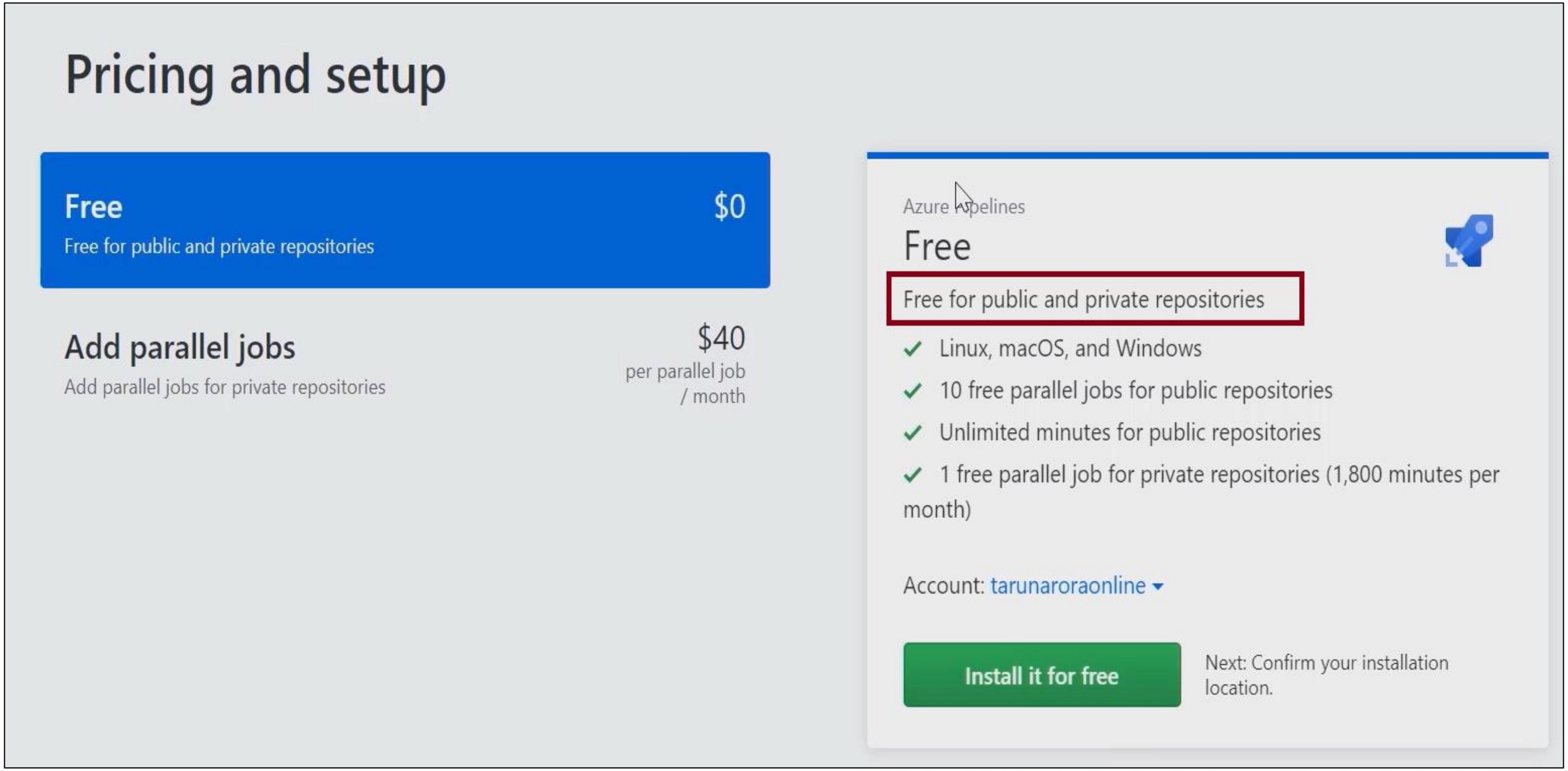
- Managing dependencies with Package Management Solutions
  - Web front end => Bower
  - Web Server node & tools => Npm
  - .NET artifacts => NuGet

Lesson 07: Public Projects

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

Public Projects



Pricing and setup

Free $0
Free for public and private repositories

Add parallel jobs $40 per parallel job / month
Add parallel jobs for private repositories

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.

Lesson 08: Files in Git

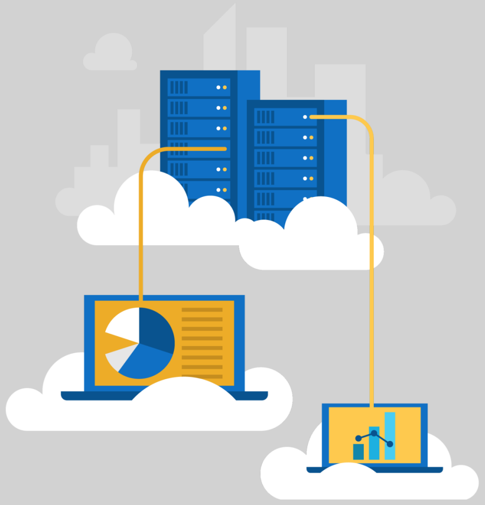
Storing Files in Git

- 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 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

Lesson 09: Module 2 Review Questions



Module 2: Review Questions

1. What are the two main types of repositories and what are advantages of each?
2. Can you name and describe the three types of branching?
3. What are GitHooks?
4. What are some best practices when working with files in Git? What do you suggest for working with large files?