- The main branch should always be in a ready-to-release state.
- Protect the main branch with branch policies.
- Any changes to the main branch flow through pull requests only.
- Tag all releases in the main branch with Git tags.

- The feature branch:
  - Use feature branches for all new features and bug fixes.
  - Use feature flags to manage long-running feature branches.
  - Changes from feature branches to the main only flow through pull requests.
  - Name your feature to reflect its purpose.

  List of branches:

  CMD

  ```
  bugfix/description
  features/feature-name
  features/feature-area/feature-name
  hotfix/description
  users/username/description
  users/username/workitem
  ```

- Pull requests:
  - Review and merge code with pull requests.
  - Automate what you inspect and validate as part of pull requests.
  - Tracks pull request completion duration and set goals to reduce the time it takes.

We'll be using the myWebApp created in the previous exercises. See Describe working with Git locally.

In this recipe, we'll be using three trendy extensions from the marketplace:

- Azure CLI: is a command-line interface for Azure.
- Azure DevOps CLI: It's an extension for the Azure CLI for working with Azure DevOps and Azure DevOps Server. It's designed to seamlessly integrate with Git, CI pipelines, and Agile tools. With the Azure DevOps CLI, you can contribute to your projects without leaving the command line. CLI runs on Windows, Linux, and Mac.
- Git Pull Request Merge Conflict: This open-source extension created by Microsoft DevLabs allows you to review and resolve pull request merge conflicts on the web. Before a Git pull request can complete, any conflicts with the target branch must be resolved. With this extension, you can resolve these conflicts on the web as part of the pull request merge instead of doing the merge and resolving conflicts in a local clone.

The Azure DevOps CLI supports returning the query results in JSON, JSONC, YAML, YAMLC

The Azure DevOps CLI supports returning the query results in JSON, JSONC, YAML, YAMLC, table, TSV, and none. You can configure your preference by using the configure command.

How to do it

1. After you've cloned the main branch into a local repository, create a new feature branch, myFeature-1:

   myWebApp **>**

   ```
   CMD
   ```

   ```
   git checkout -b feature/myFeature-1
   ```

   Output:

   *Switched to a new branch '*feature/myFeature-1*'.*

2. Run the Git branch command to see all the branches. The branch showing up with an asterisk is the "currently-checked-out" branch:

   myWebApp **>**

   ```
   CMD
   ```

   ```
   git branch
   ```

   Output:

   feature/myFeature-1

   main

3. Make a change to the Program.cs file in the feature/myFeature-1 branch:

   myWebApp **>**

   ```
   CMD
   ```

   ```
   notepad Program.cs
   ```

4. Stage your changes and commit locally, then publish your branch to remote:

   myWebApp **>**

   ```
   CMD
   ```

```
git status
```

Output:

On branch feature/myFeature-1 Changes not staged for commit: (use "git add <file>..." to update what will be committed) (use "git checkout -- <file>..." to discard changes in working directory) modified: Program.cs.

myWebApp **>**

CMD

```
git add .
git commit -m "Feature 1 added to Program.cs"
```

Output:

[feature/myFeature-1 70f67b2] feature 1 added to program.cs 1 file changed, 1 insertion(+).

myWebApp **>**

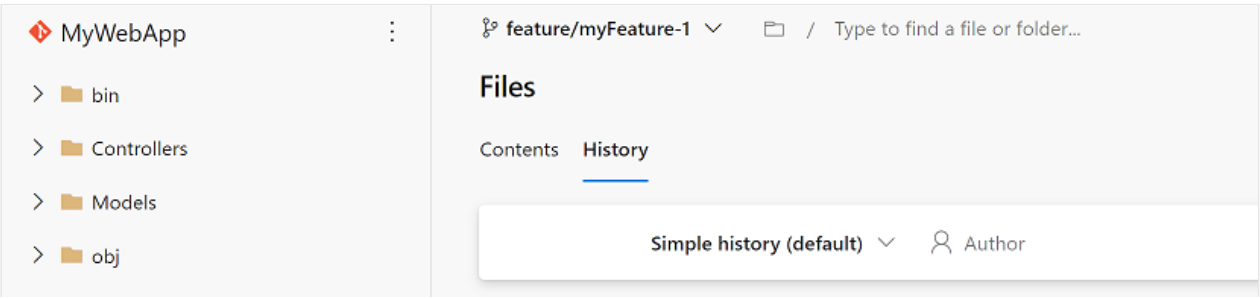
CMD

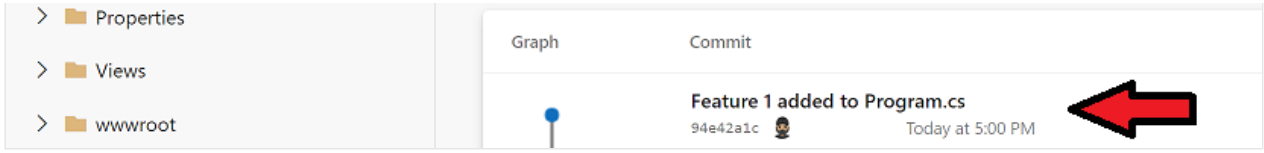
```
git push -u origin feature/myFeature-1
```

Output:

*Delta compression using up to 8 threads. Compressing objects: 100% (3/3), done. Writing objects: 100% (3/3), 348 bytes | 348.00 KiB/s, done. Total 3 (delta 2), reused 0 (delta 0) remote: Analyzing objects... (3/3) (10 ms) remote: Storing packfile... done (44 ms) remote: Storing index... done (62 ms) To https://dev.azure.com/Geeks/PartsUnlimited/_git/MyWebApp * [new branch] feature/myFeature-1 -> feature/myFeature-1 Branch feature/myFeature-1 set up to track remote branch feature/myFeature-1 from origin.*

The remote shows the history of the changes:





5. Configure Azure DevOps CLI for your organization and project. Replace **organization** and **project name**:

```
CMD

az devops configure --defaults
organization=https://dev.azure.com/organization project="project name"
```

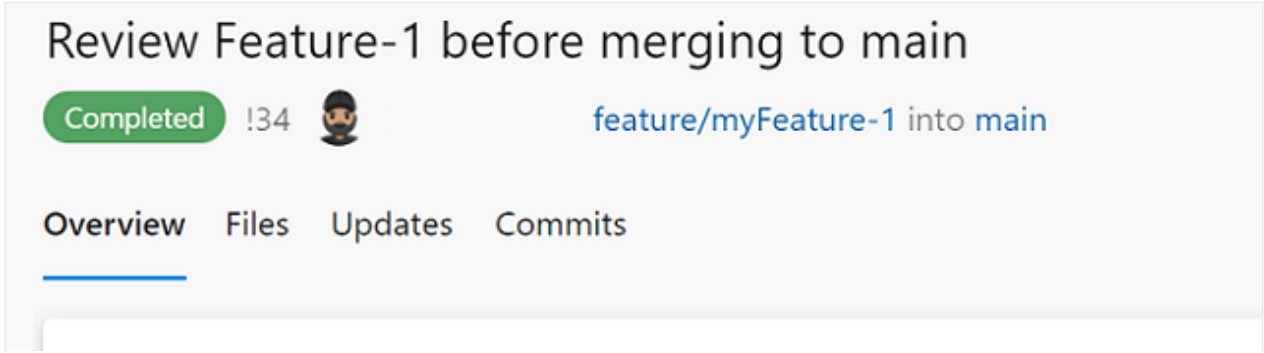
6. Create a new pull request (using the Azure DevOps CLI) to review the changes in the feature-1 branch:

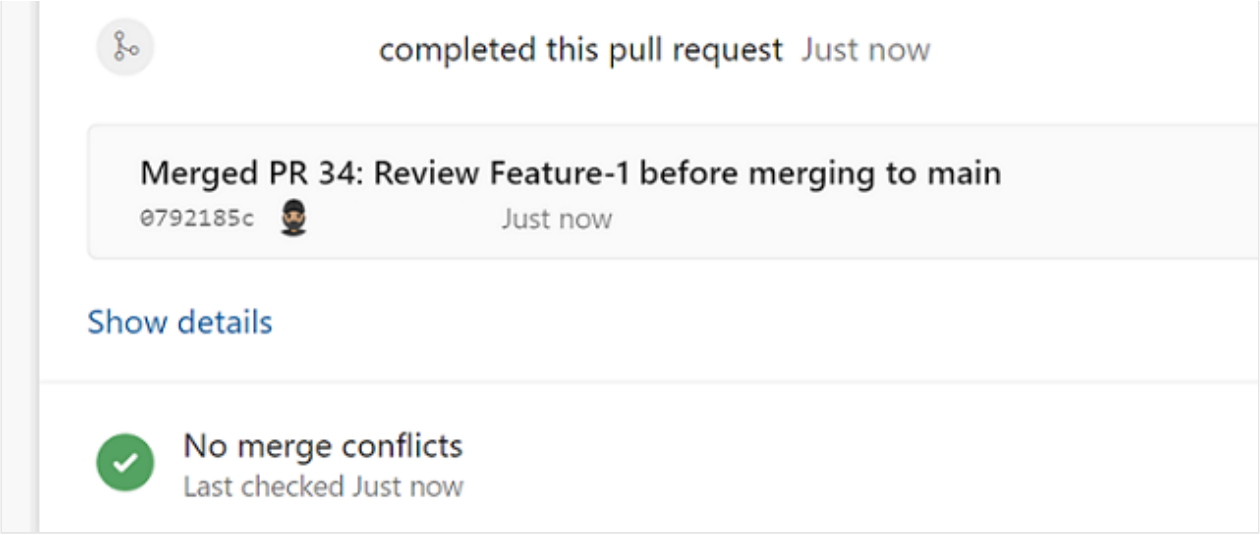
```
CMD

az repos pr create --title "Review Feature-1 before merging to main" --work-
items 38 39 `
--description "#Merge feature-1 to main" `
--source-branch feature/myFeature-1 --target-branch main `
--repository myWebApp --open
```

Use the --open switch when raising the pull request to open the pull request in a web browser after it has been created. The --deletesource-branch switch can be used to delete the branch after the pull request is complete. Also, consider using --auto-complete to complete automatically when all policies have passed, and the source branch can be merged into the target branch.

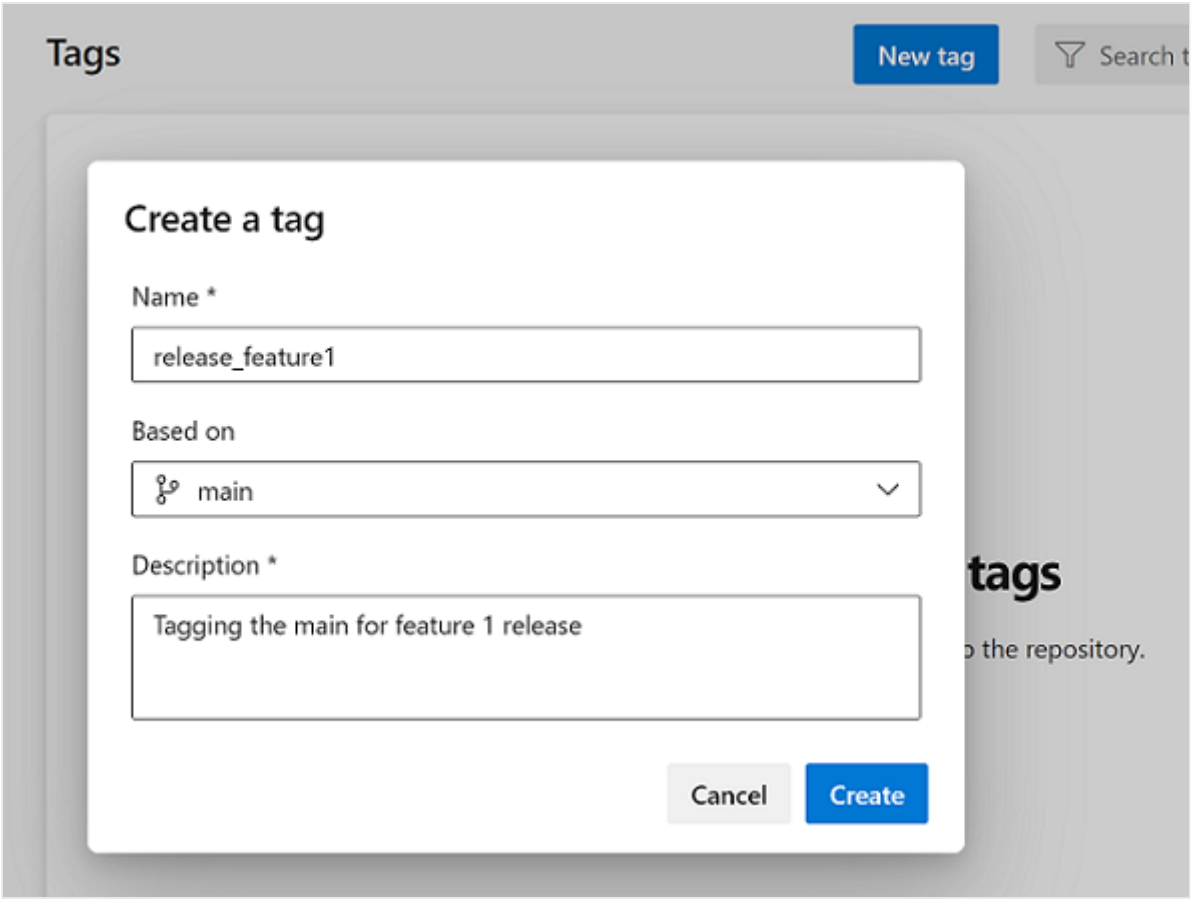
> **Note**
>
> For more information about **az repos pr create** parameter, see **Create a pull request to review and merge code**.

The team jointly reviews the code changes and approves the pull request:





The main is ready to release. Team tags main branch with the release number:



7. Start work on Feature 2. Create a branch on remote from the main branch and do the checkout locally:

myWebApp >

CMD

```
git push origin origin:refs/heads/feature/myFeature-2
```

Output:

*Total 0 (delta 0), reused 0 (delta 0) To https://dev.azure.com/Geeks/PartsUnlimited/_git/MyWebApp * [new branch] origin/HEAD -> refs/heads/feature/myFeature-2.*

myWebApp >

```
CMD
```

```
git checkout feature/myFeature-2
```

Output:

Switched to a new branch 'feature/myFeature-2' Branch feature/myFeature-2 set up to track remote branch feature/myFeature-2 from origin.

8. Modify Program.cs by changing the same comment line in the code changed in feature-1.

```

```

```
public class Program
{
    // Editing the same line (file from feature-2 branch)
    public static void Main(string[] args)
    {
        BuildWebHost(args).Run();
    }

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

9. Commit the changes locally, push them to the remote repository, and then raise a pull request to merge the changes from feature/myFeature-2 to the main branch:

```
CMD
```

```
az repos pr create --title "Review Feature-2 before merging to main" --work-
items 40 42 `
--description "#Merge feature-2 to main" `
--source-branch feature/myFeature-2 --target-branch main `
--repository myWebApp --open
```

A critical bug is reported in production against the feature-1 release with the pull request in flight. To investigate the issue, you need to debug against the version of code currently deployed in production. To investigate the issue, create a new fof branch using the release_feature1 tag:

myWebApp **>**

```
CMD
```

```
git checkout -b fof/bug-1 release_feature1
```

Output:

Switched to a new branch 'fof/bug-1'.

10. Modify Program.cs by changing the same line of code that was changed in the feature-1 release:

```
public class Program
{
    // Editing the same line (file from feature-FOF branch)
    public static void Main(string[] args)
    {
        BuildWebHost(args).Run();
    }

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

11. Stage and commit the changes locally, then push changes to the remote repository:

myWebApp **>**

```
CMD
```

```
git add .
git commit -m "Adding FOF changes."
git push -u origin fof/bug-1
```

Output:

*To https://dev.azure.com/Geeks/PartsUnlimited/_git/MyWebApp * [new branch] fof/bug-1 -> fof/bug-1 Branch fof/bug-1 set up to track remote branch fof/bug-1 from*

fof/bug-1 -> fof/bug-1 Branch fof/bug-1 set up to track remote branch fof/bug-1 from origin.

12. Immediately after the changes have been rolled out to production, tag the fof\bug-1 branch with the release_bug-1 tag, then raise a pull request to merge the changes from fof/bug-1 back into the main:

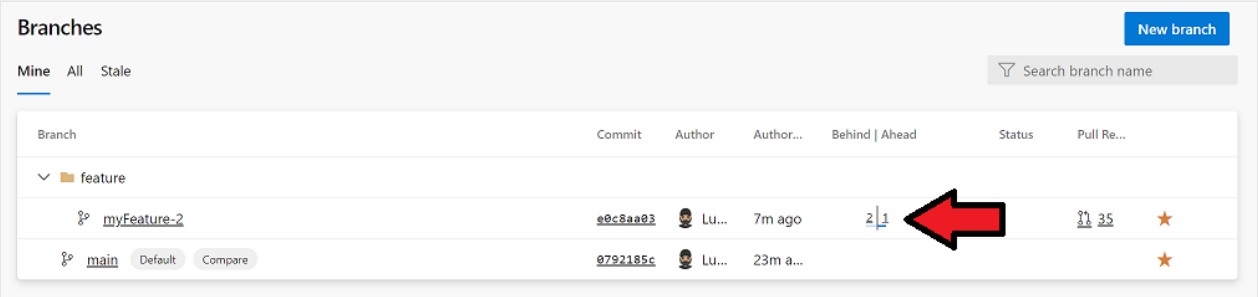
CMD

```
az repos pr create --title "Review Bug-1 before merging to main" --work-items
100 `
--description "#Merge Bug-1 to main" `
--source-branch fof/Bug-1 --target-branch main `
--repository myWebApp --open
```

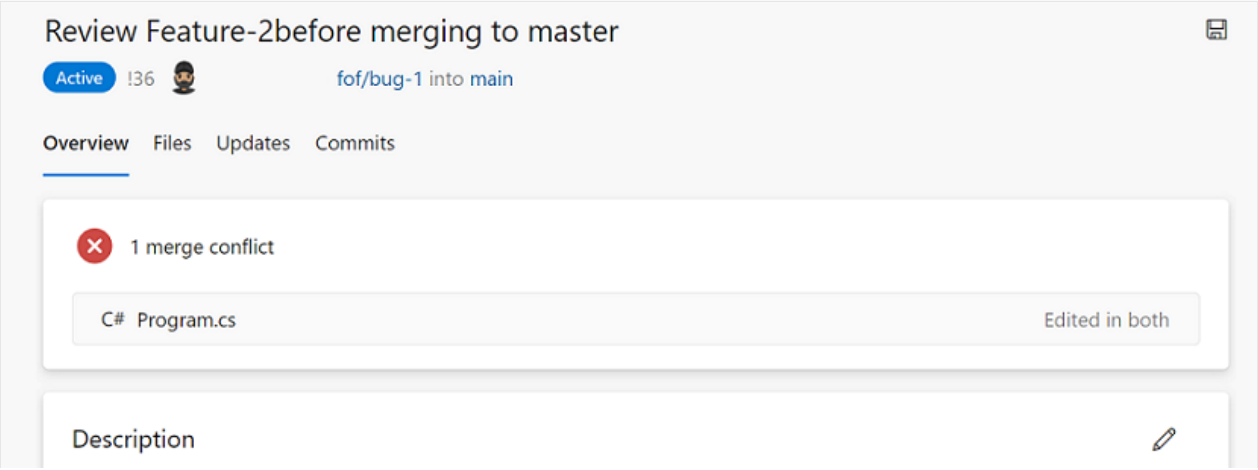
As part of the pull request, the branch is deleted. However, you can still reference the entire history using the tag.

With the critical bug fix out of the way, let's go back to the review of the feature-2 pull request.

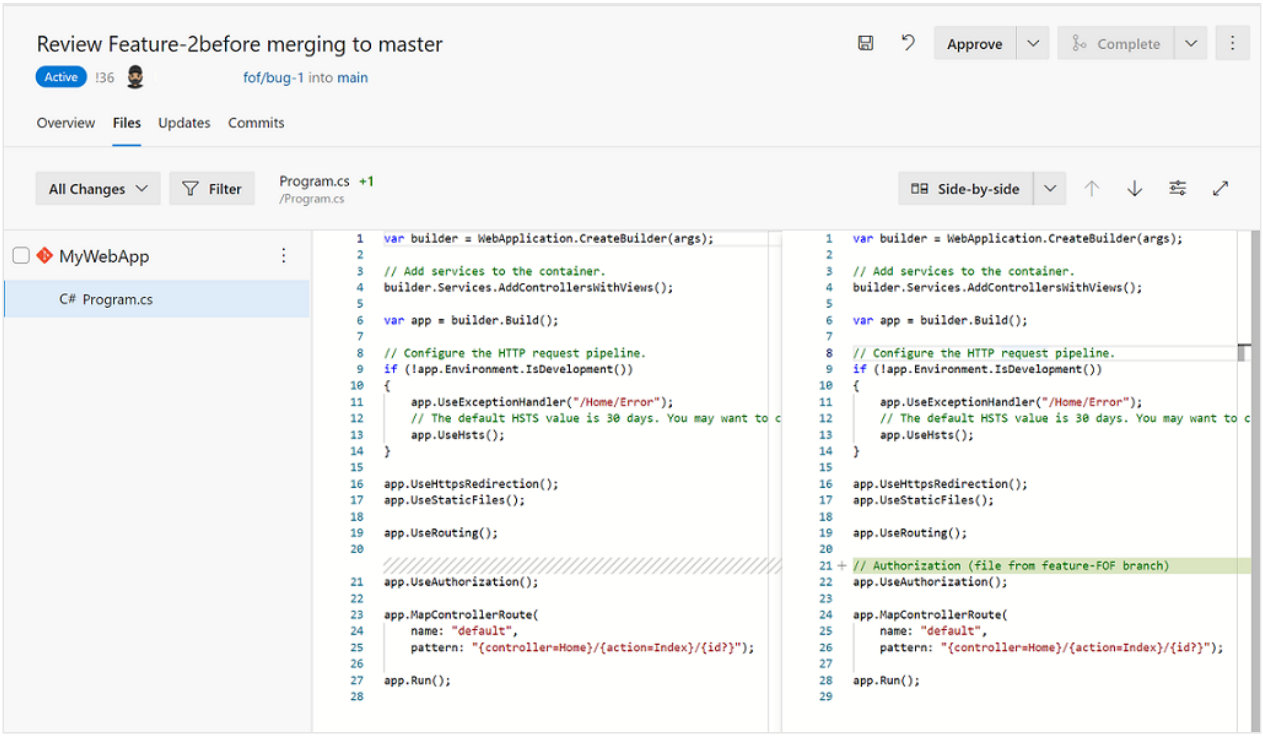
The branches page makes it clear that the feature/myFeature-2 branch is one change ahead of the main and two changes behind the main:



If you tried to approve the pull request, you'd see an error message informing you of a merge conflict:



13. The Git Pull Request Merge Conflict resolution extension makes it possible to resolve merge conflicts right in the browser. Navigate to the conflicts tab and click on Program.cs to resolve the merge conflicts:



The user interface allows you to take the source, target, add custom changes, review, and submit the merge. With the changes merged, the pull request is completed.

How it works

We learned how the Git branching model gives you the flexibility to work on features in parallel by creating a branch for each feature.

The pull request workflow allows you to review code changes using the branch policies.

Git tags are a great way to record milestones, such as the version of code released; tags give you a way to create branches from tags

you a way to create branches from tags.

We created a branch from a previous release tag to fix a critical bug in production.

The branches view in the web portal makes it easy to identify branches ahead of the main. Also, it forces a merge conflict if any ongoing pull requests try to merge to the main without resolving the merge conflicts.

A lean branching model allows you to create short-lived branches and push quality changes to production faster.

Next unit: Explore GitHub flow

Continue >

How are we doing? ☆ ☆ ☆ ☆ ☆