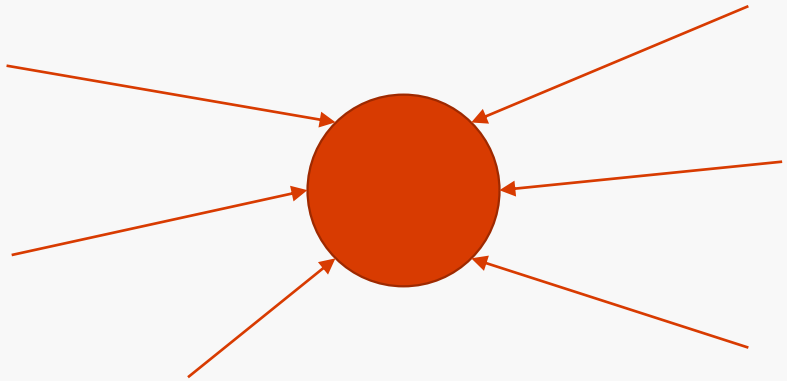
Microsoft

Git

Crystal Tenn
Crystal.Tenn@microsoft.com

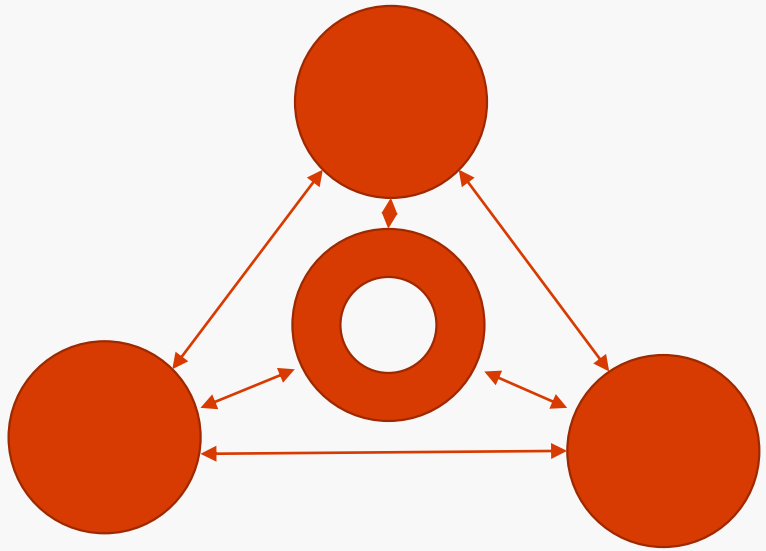
Centralized Version Control System

- Most version control systems are centralized version control systems (CVCS). In a CVCS, a single, shared repository is used to maintain an authoritative copy of the source code, and all changes are made against the central copy. Individuals have local copies of the files which they modify and send back to the server to share with others.
- Team Foundation Version Control (TFVC) is a centralized, server-based version control system.
  - TFVC enables two modes of operating against the mode uses Server Workspaces, where the server must be contacted before any developer can add, edit, rename, delete, or even diff their changes against the original.
  - The second (default) mode uses Local Workspaces, where the additional metadata on the developer's machine tracks changes to files, but the server must still be contacted to update files and submit changes. Local workspaces offer greater flexibility for working with source control while offline, and without good communication can lead to additional merging before changes can be checked in. NOTE: Metadata does NOT include complete file history.
- In addition to TFVC, there numerous other CVCSs – including systems such as Visual SourceSafe5, CVS6 and Subversion7.



Distributed Version Control System

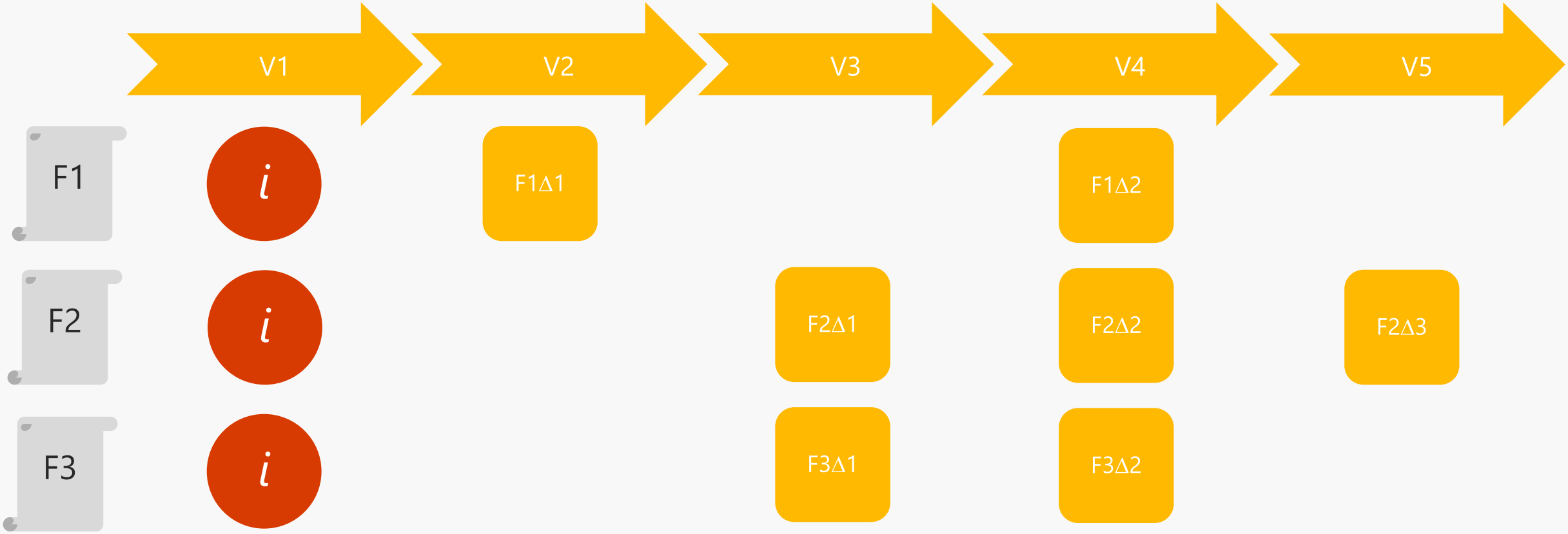
- Git is an open source distributed version control system. Git enables developers to work in two modes
  - By setting up a central copy and working in a hub and spoke model
  - Or, by creating a true peer-to-peer model.
- The first mode uses a central repository. Once the repo is cloned by developers, the central repo need not be contacted to be able to commit and view full history from their local repo. Developers can work in complete isolation and disconnected from the central repo until they are ready to merge their changes with the central repo.
- In addition to Git, there are numerous other DVCSs – including Monotone and Mercurial.



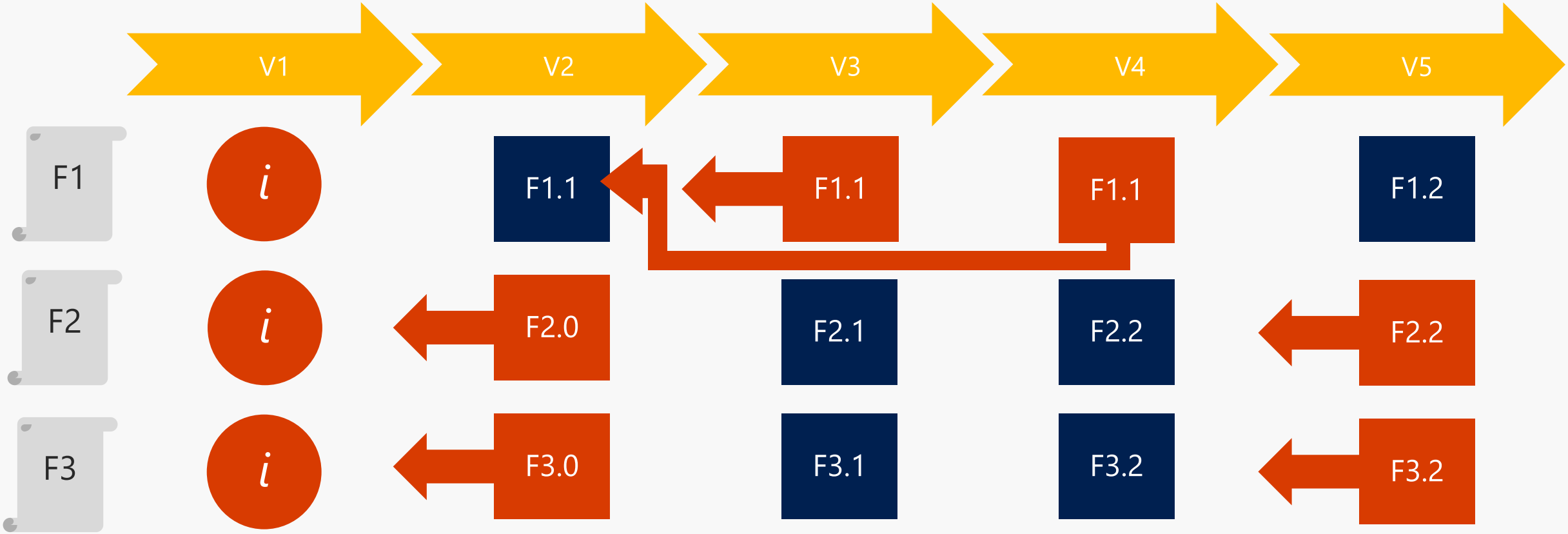
Modern source-control approaches

		Strengths	Best for
Centralized Version Control	Check-in Check-out	• Fine level permission control • Allows usage monitoring	• Large integrated codebases • Control and auditability over source code down to the file level
	Edit Commit	• Offline editing support • Easy to edit files outside Visual Studio or Eclipse	• Medium-sized integrated codebases • A balance of fine-grained control with reduced friction
Distributed Version Control (DVCS)		• Fast offline experience • Complete repository with portable history • Flexible advanced branching model	• Modular codebases • Integrating with open source • Highly distributed teams

Versioning in TFVC (and others)



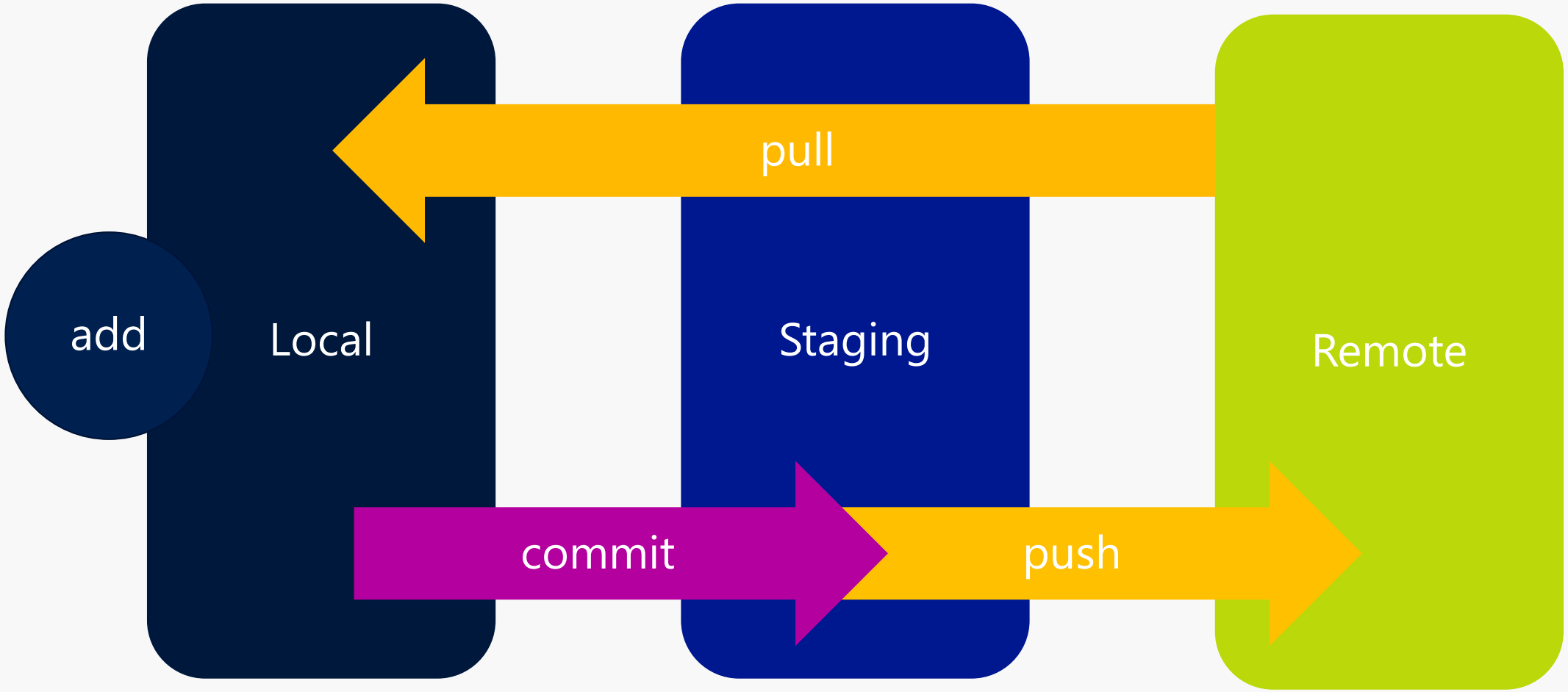
Versioning in Git



Git operations are performed locally

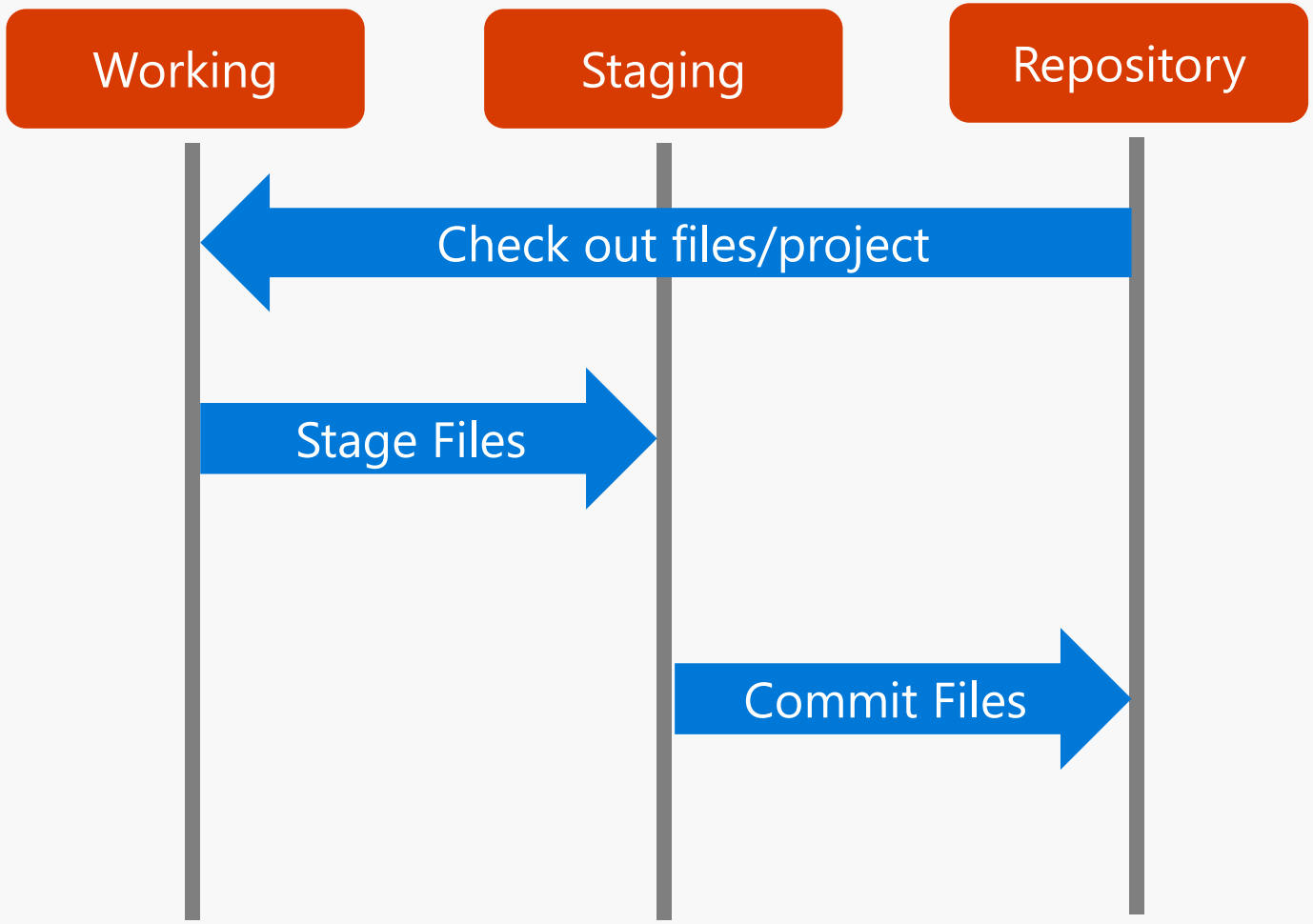
Git only adds data

Basic Git Workflow



Local Operations

- Three directories
  - Working
  - Staging
  - Repository

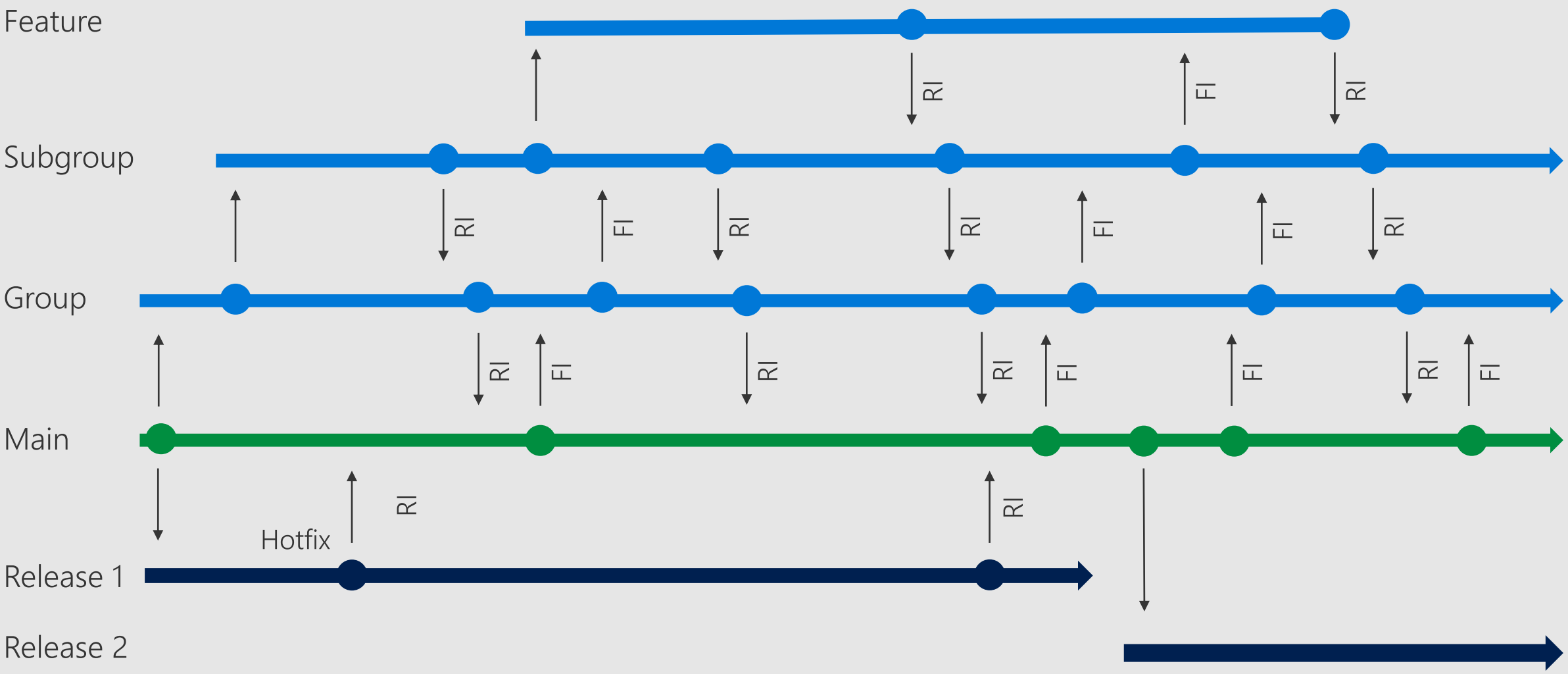


Branching Strategies

BRANCHES

Sometimes they appear out of nowhere.

Typical TFVC Branching Structure



"Organizations which design systems… are constrained to produce designs which are copies of the communication structures of these organizations…"

Conway's Law

Organizations tend to produce branching structures that copy the organization chart.

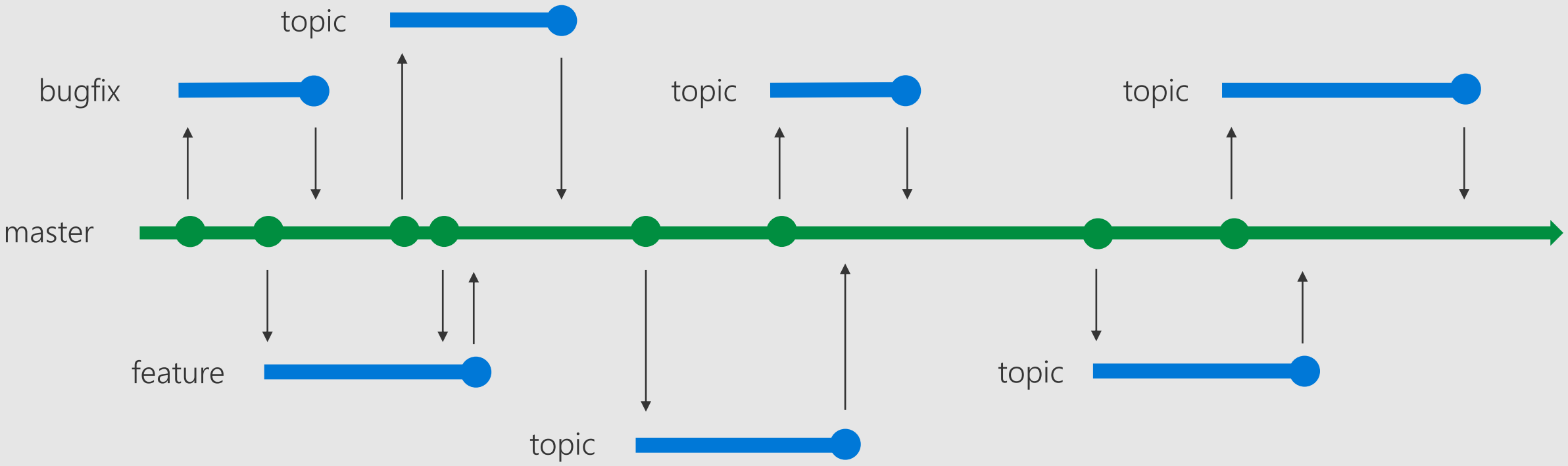
Simplified Branching Structure

Code close to master

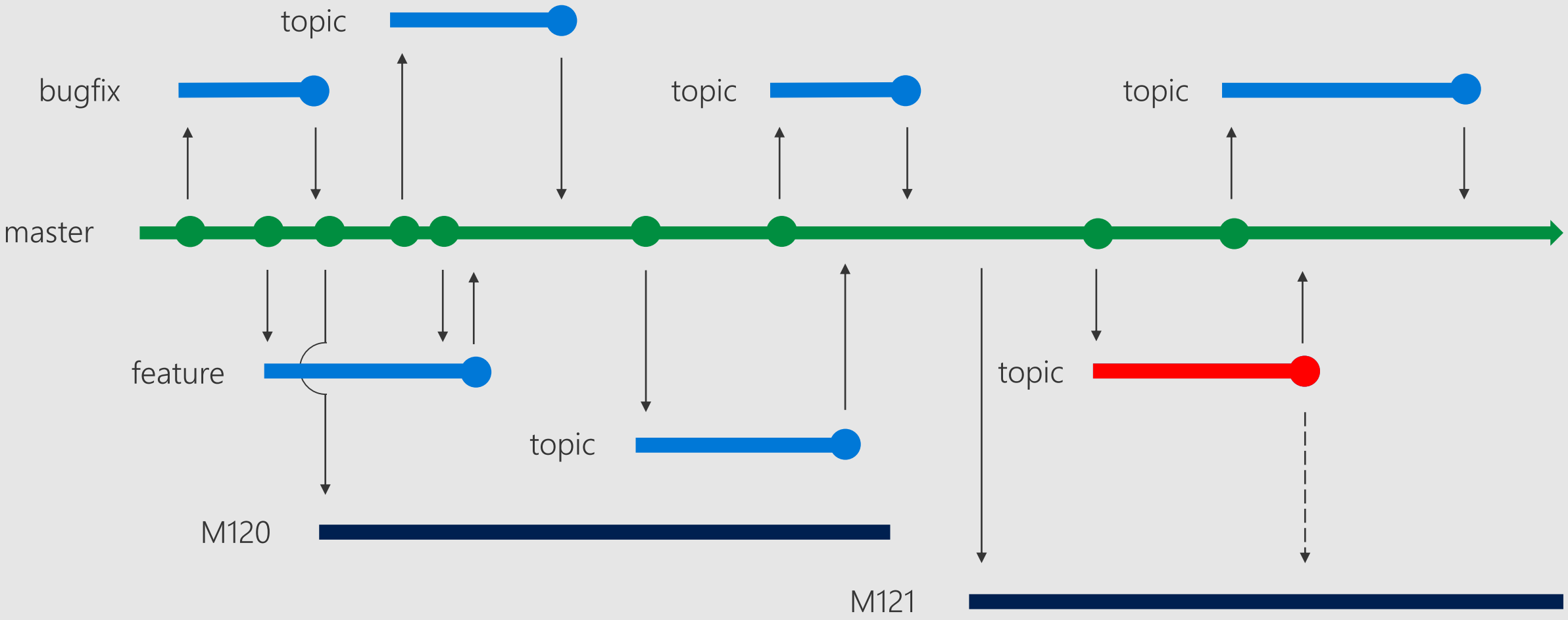
Small, simple changes

- Fewer merge conflicts
- Easy to code review
- Encourages pull requests
- Simpler to ship; faster velocity

GitHub Flow Branching Structure



Release Flow Branching Structure



Common Git Commands and Mapping to TFVS

Common Operations

Operation	Command
Create (initialize) a local repository	git init
Clone a remote repository	git clone
Fetch and then pull changes from a remote repository	git fetch, git pull
Manage the set of repositories ("remotes") whose branches you track.	git remote
Stage and then Commit changes	git add, git commit
Undo a committed change	git revert
Branch and merge/rebase	git branch, git merge or git rebase
Push changes to a remote repository	git push

command mapping

Get

Check In

Pending Changes

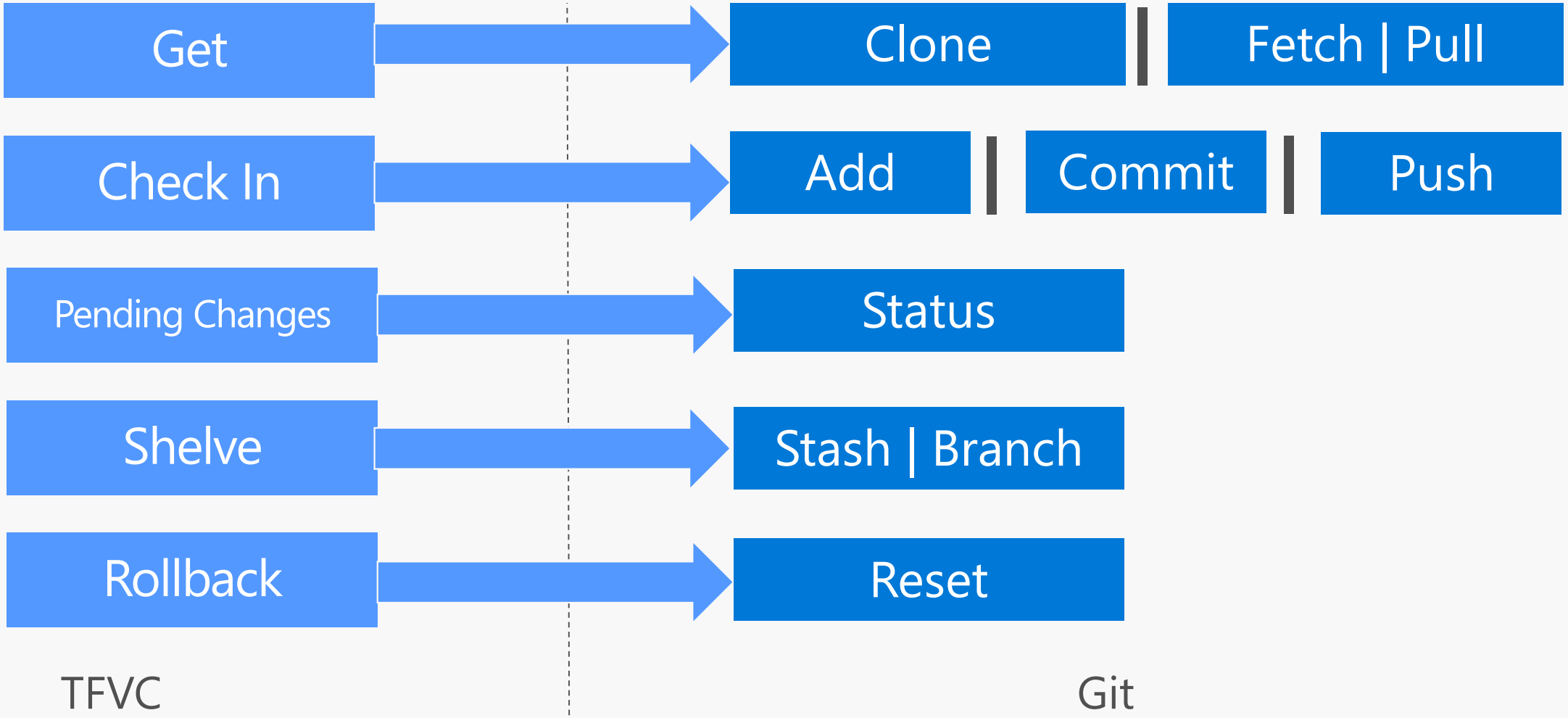
Shelve

Rollback

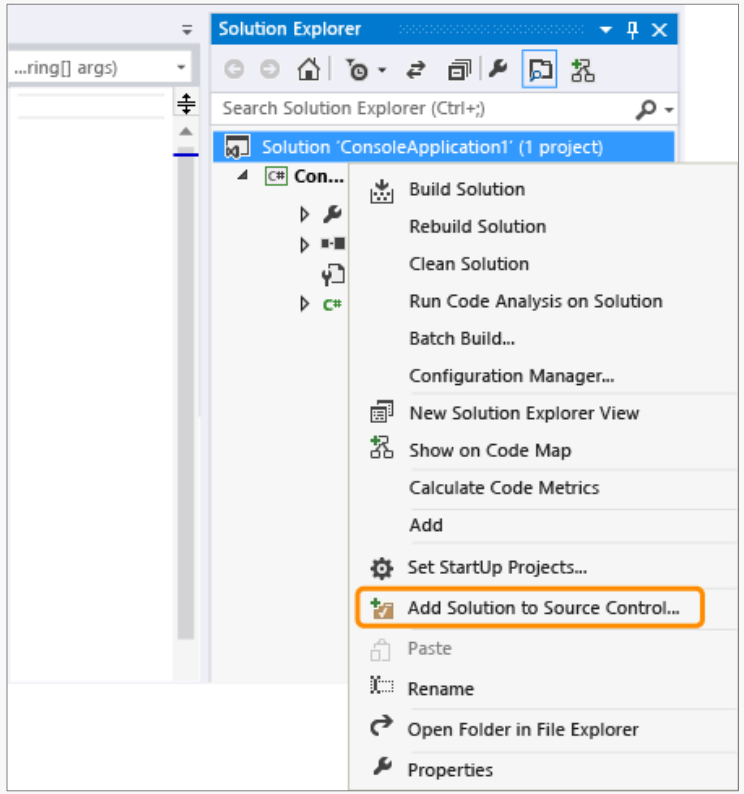
TFVC

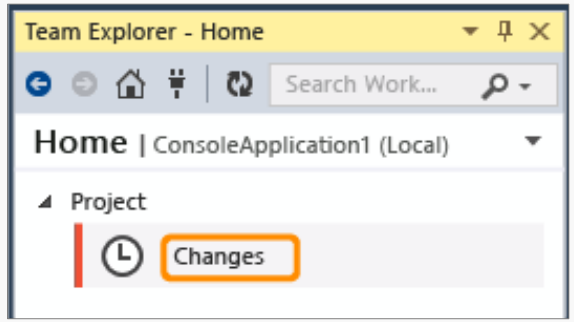
Git

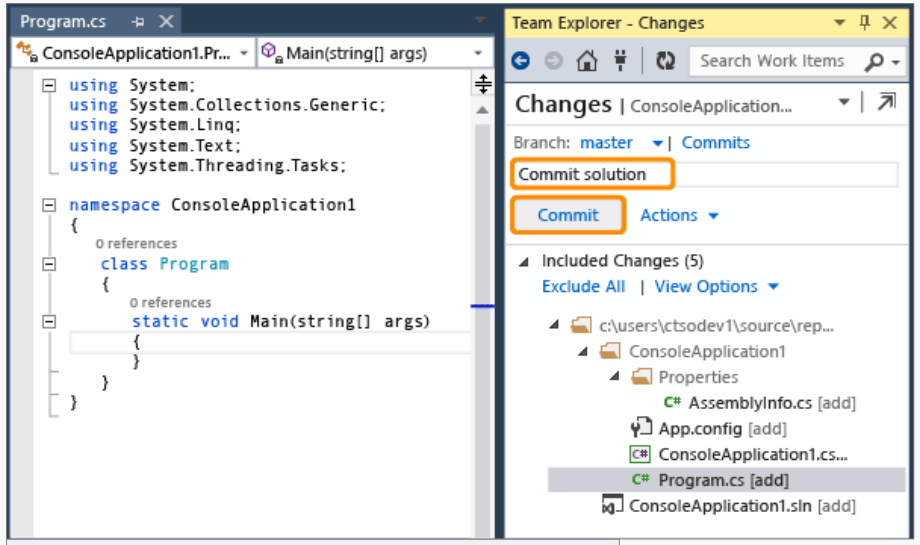
command mapping

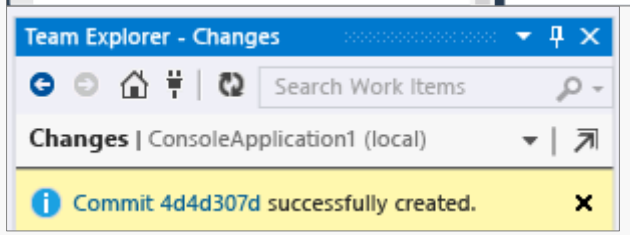


Add an Existing Solution under Local Git Version Control





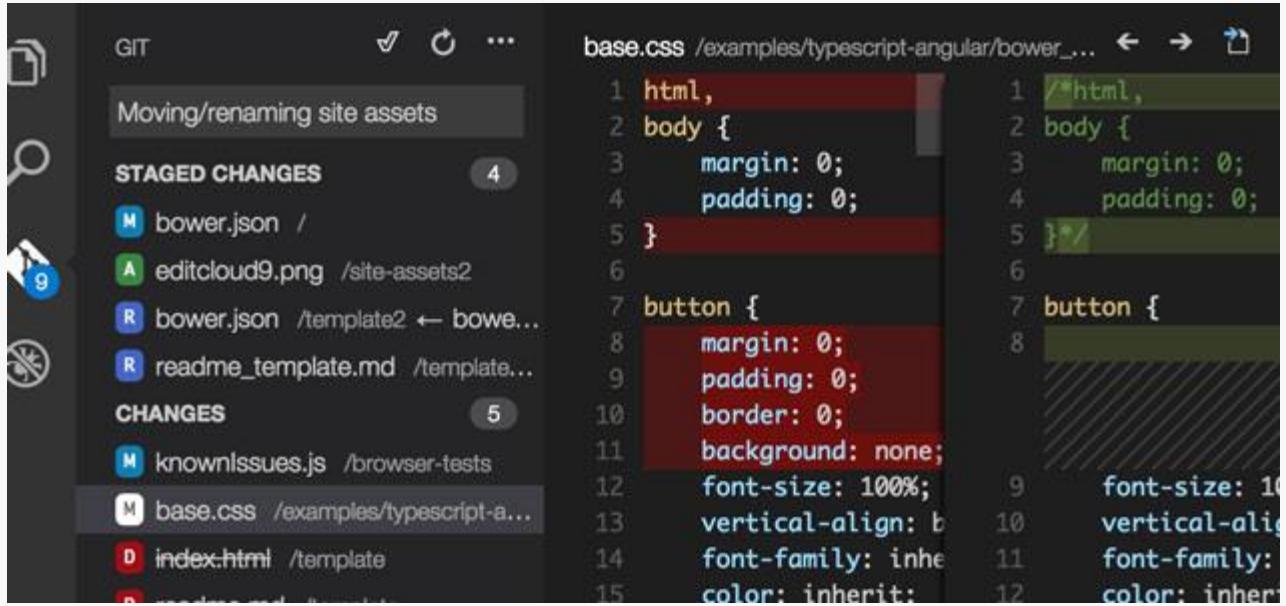




Git Integration into Tools

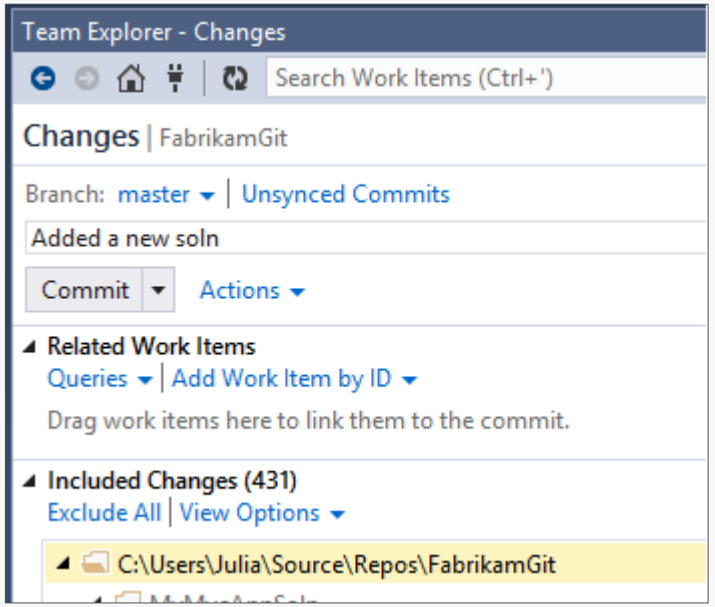
Git Integration in Visual Studio Code

- Visual Studio Code is free and available on your favorite platform — Linux, OS X, and Windows.
  - Code editing redefined, optimized for building & debugging modern web & cloud applications.
  - VS Code is relatively New – announced at //Build 2015
  - https://www.visualstudio.com/products/code-vs.aspx
  - https://code.visualstudio.com/docs/editor/versioncontrol

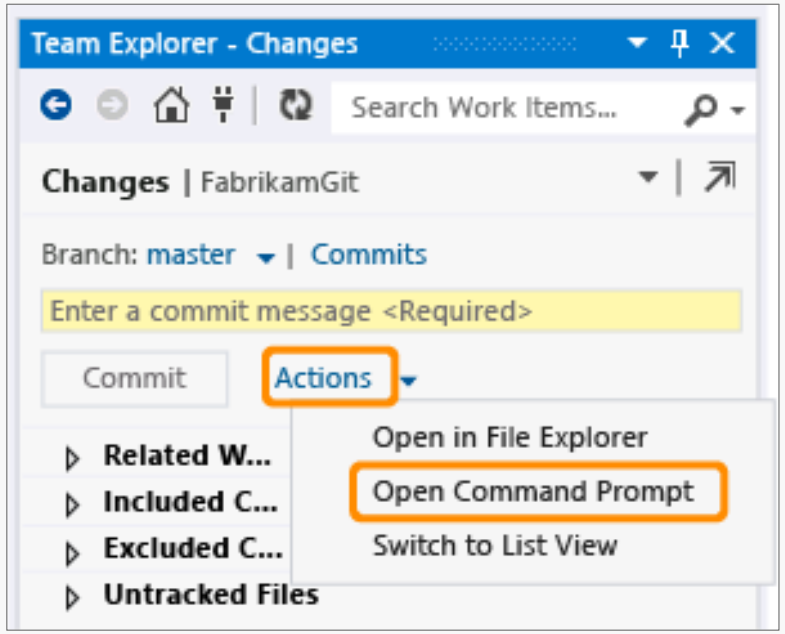


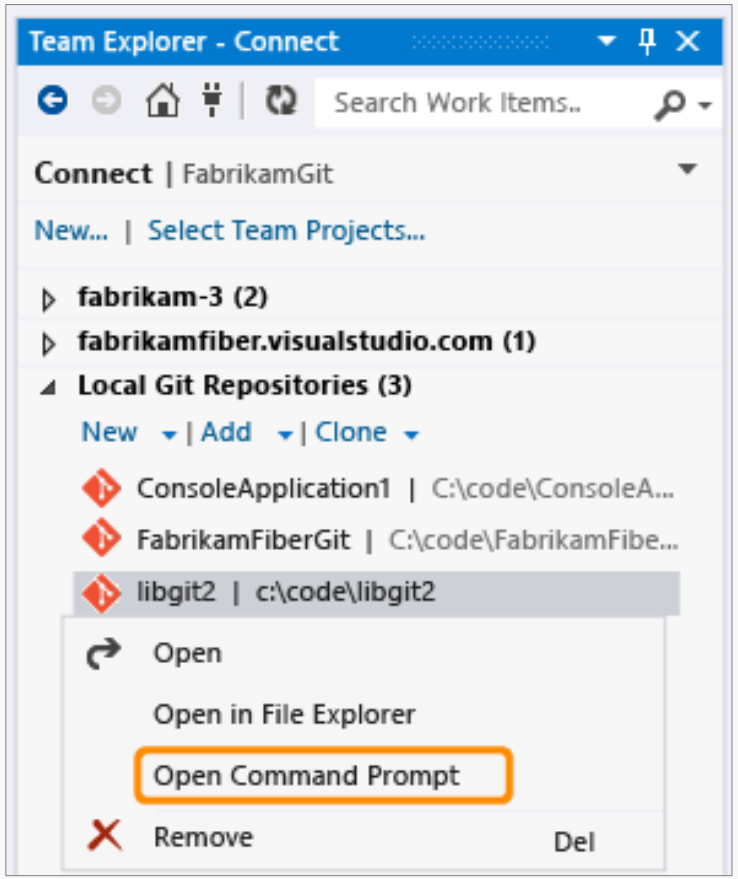
Use Visual Studio with Git

- Use Visual Studio and Git to collaborate with your team using:
  - Team Foundation Server (on-premises or in the cloud),
  - CodePlex,
  - Or a third-party service such as GitHub or Bitbucket



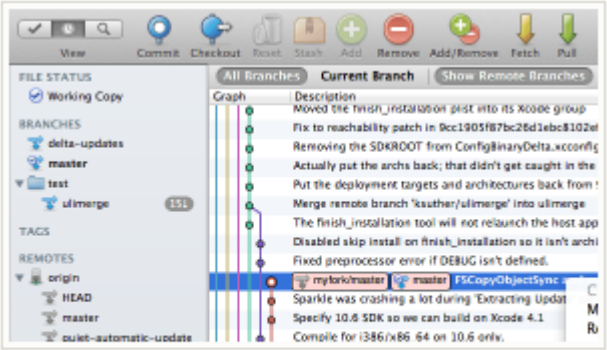
Launch the Git Command Prompt



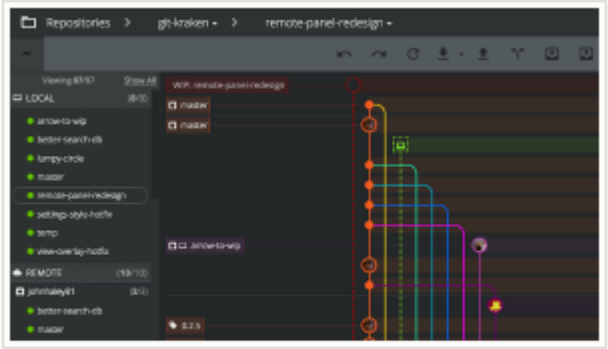


Git GUI Clients

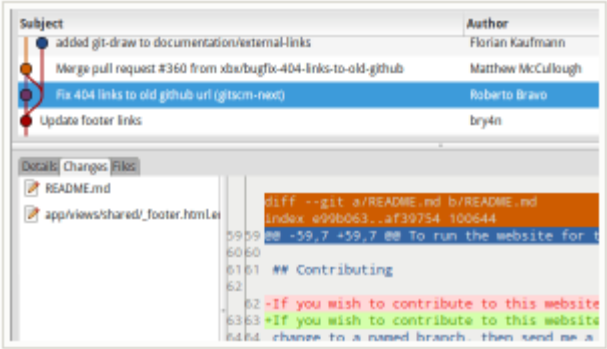
- There are multiple additional GUI Clients available
  - https://git-scm.com/downloads/guis



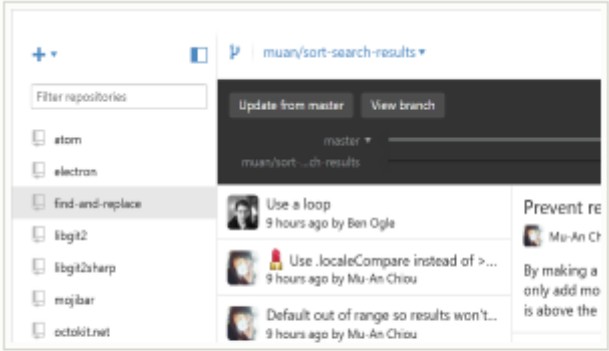
SourceTree
Platforms: Mac, Windows
Price: Free



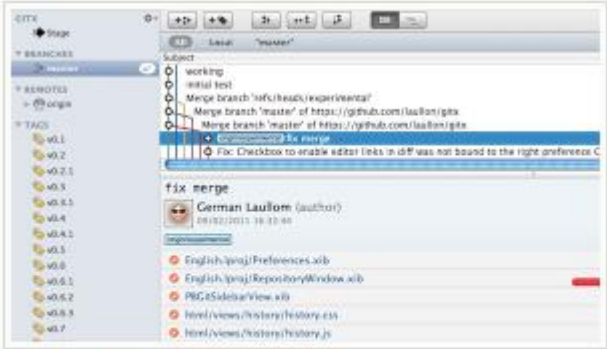
GitKraken
Platforms: Windows, Mac, Linux
Price: Free



gitg
Platforms: Linux
Price: Free



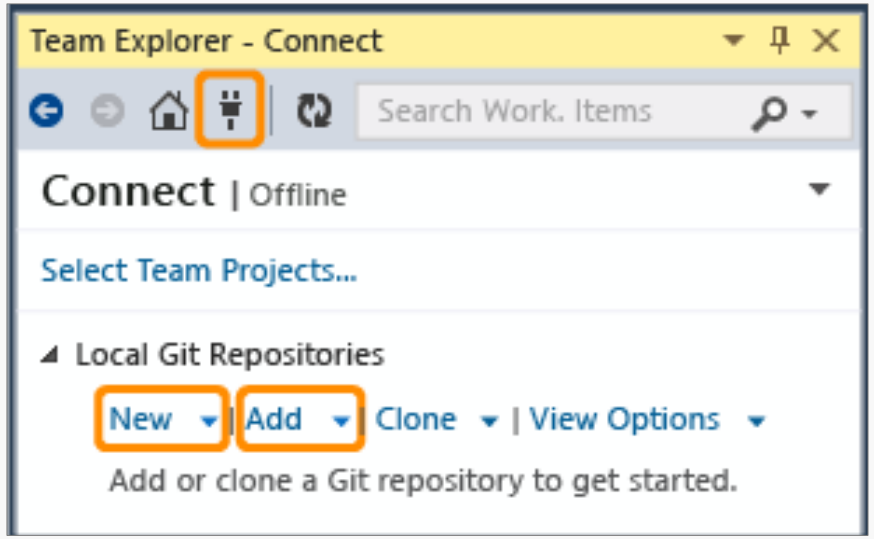
GitHub Desktop
Platforms: Windows, Mac
Price: Free

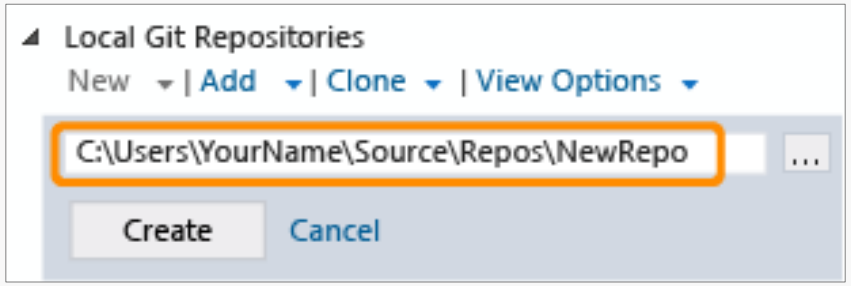


GitX-dev
Platforms: Mac
Price: Free

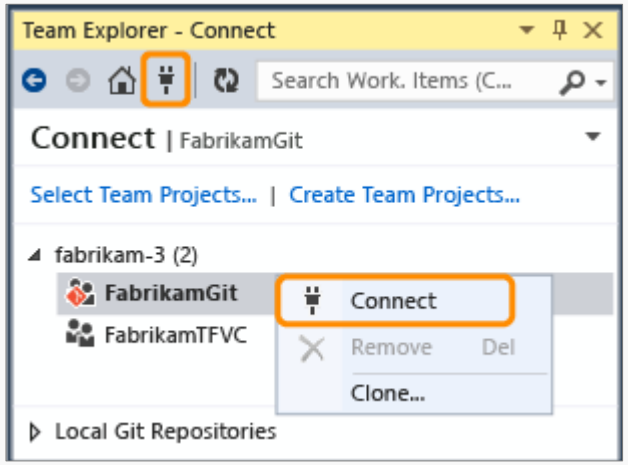
VS Git Plugin – Connecting to a Project

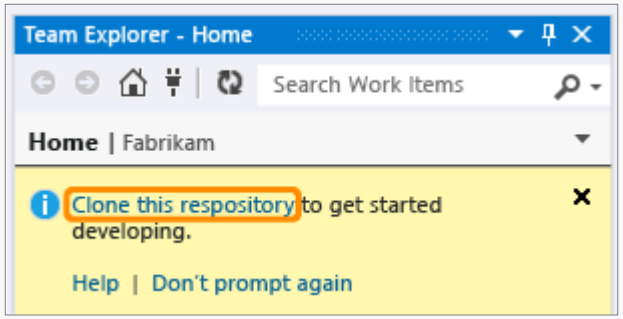
Create or Add a Local Repository

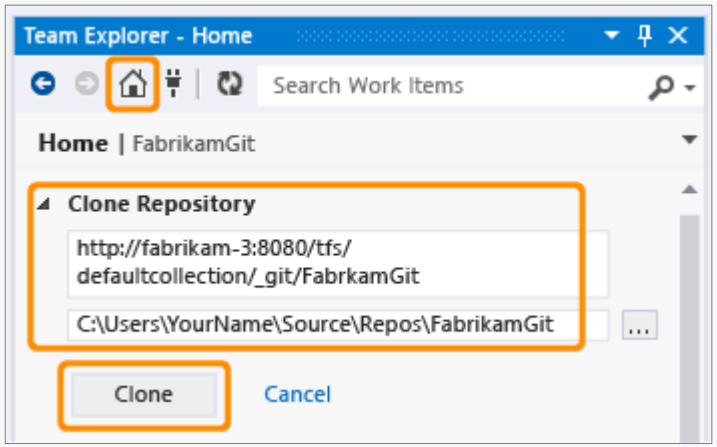




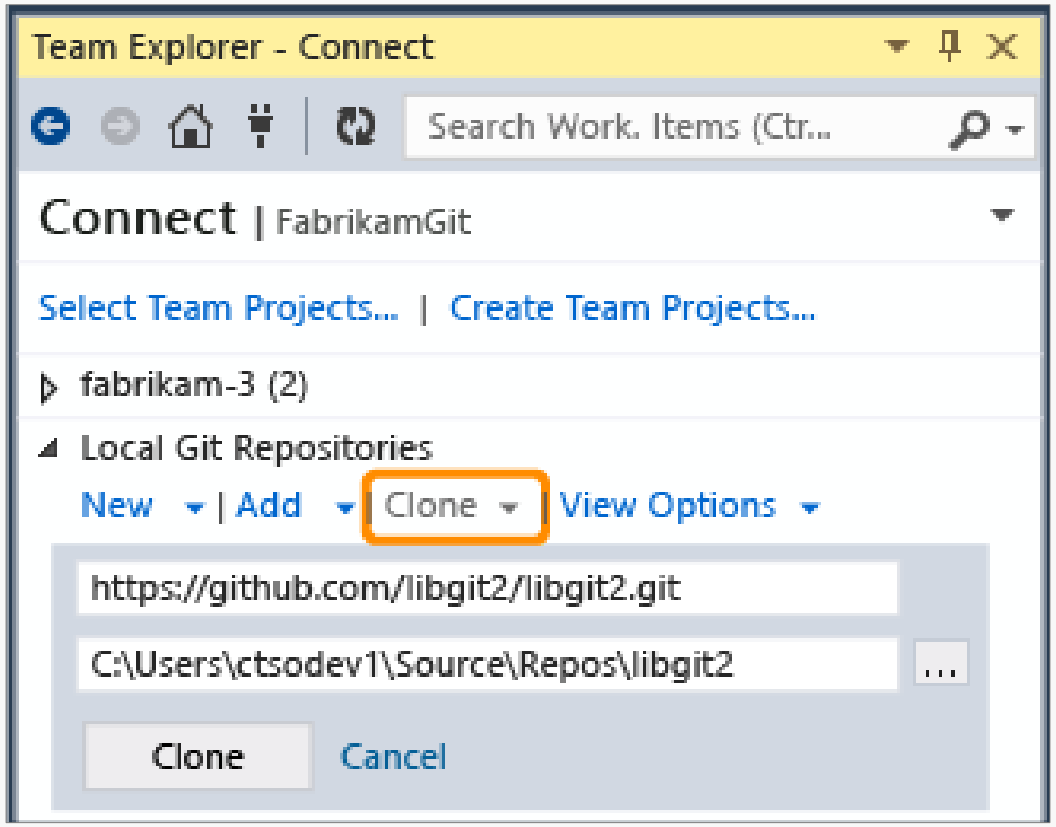
Open and Clone a Git Team Project







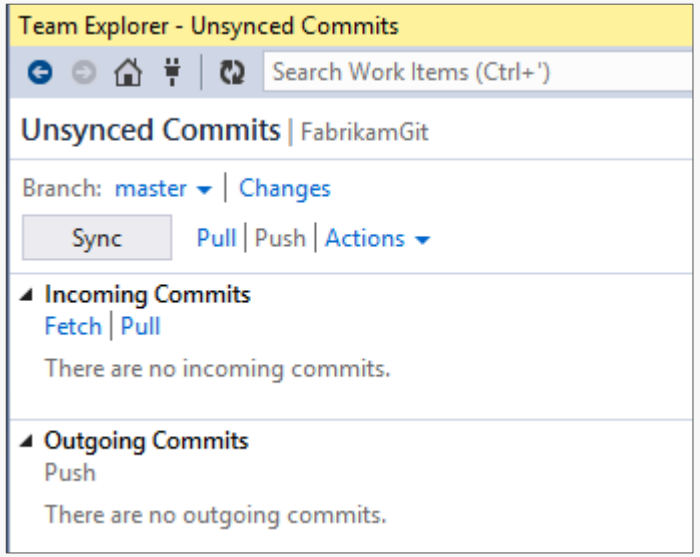
Clone a Remote Git Repository from a Third-party Service



VS Git Plugin – Interacting with a Project

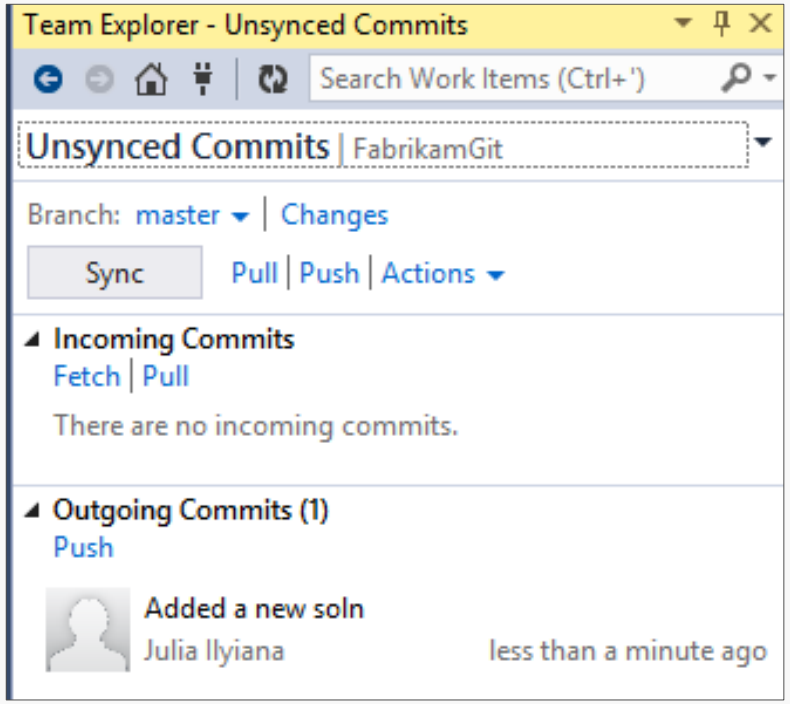
Fetch

- Fetch from Commits page
- Fetch commits from your team project before you pull
- Fetch before you can get a copy of a published branch
- After you fetch a commit, you can open its context menu and choose View Commit Details



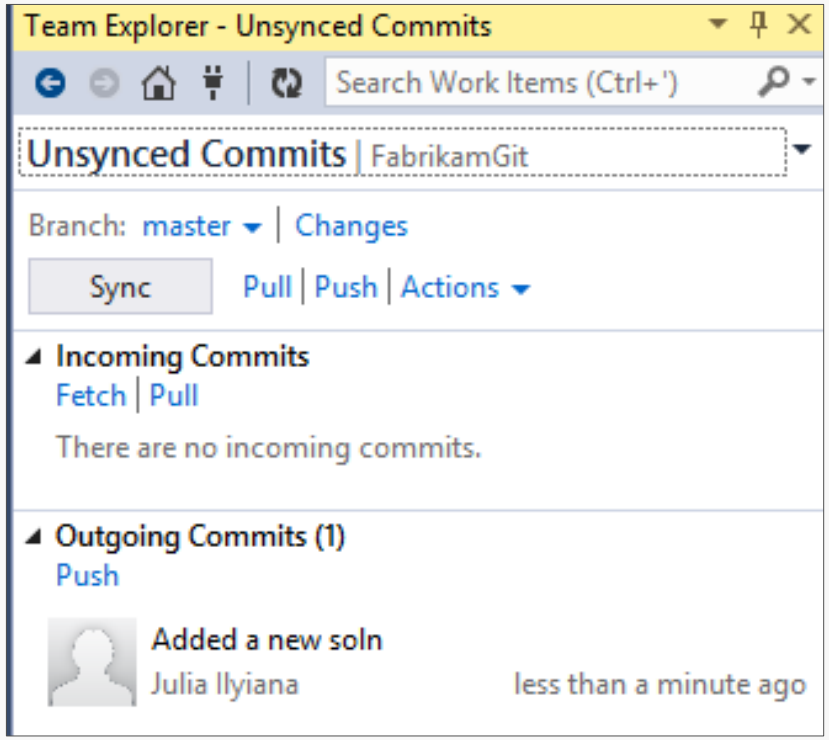
Pull

- Integrate changes from your team into your local repository
- Pull any commits pushed by your team



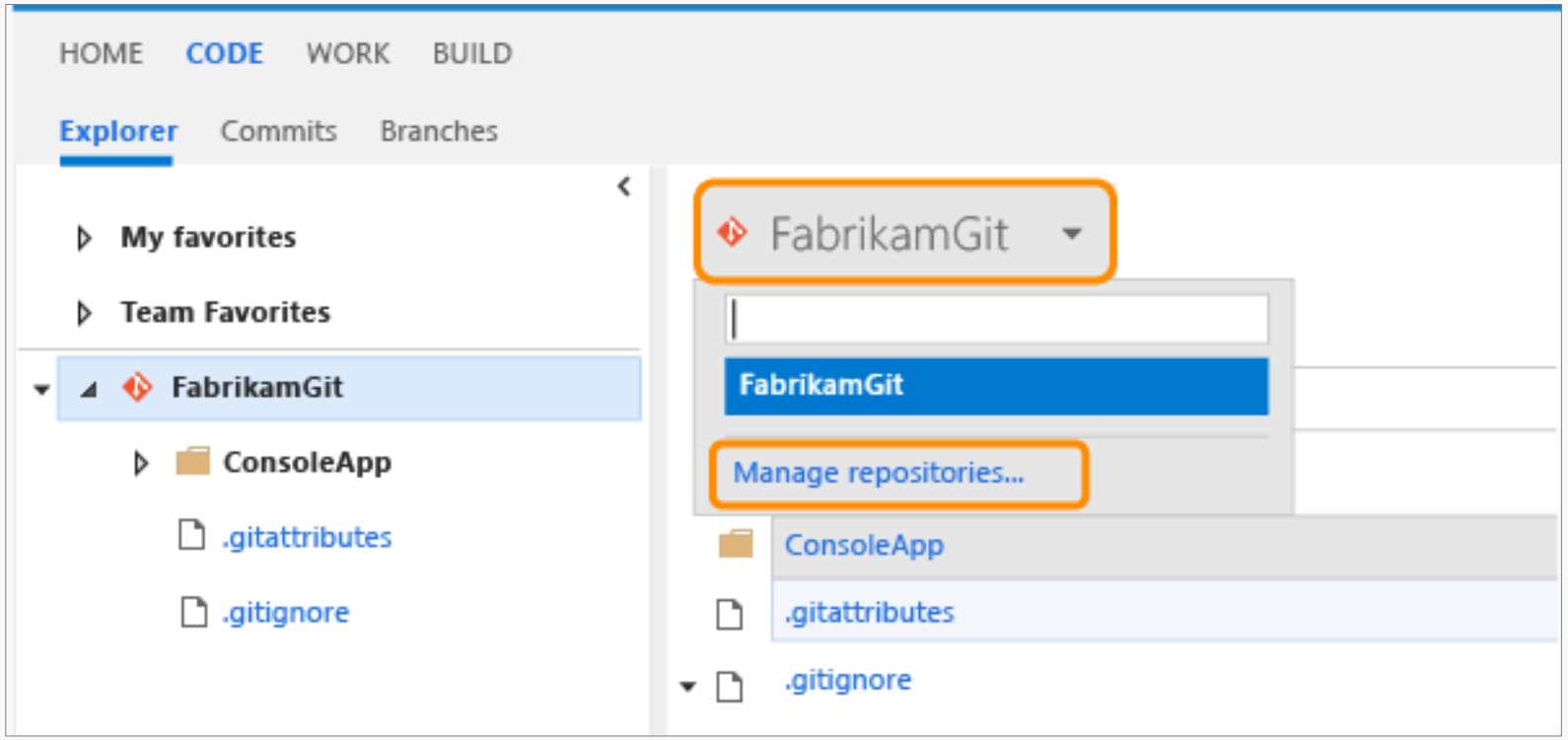
Push

- Push changes to commit them to the team's remote Git repository



Multiple Repositories

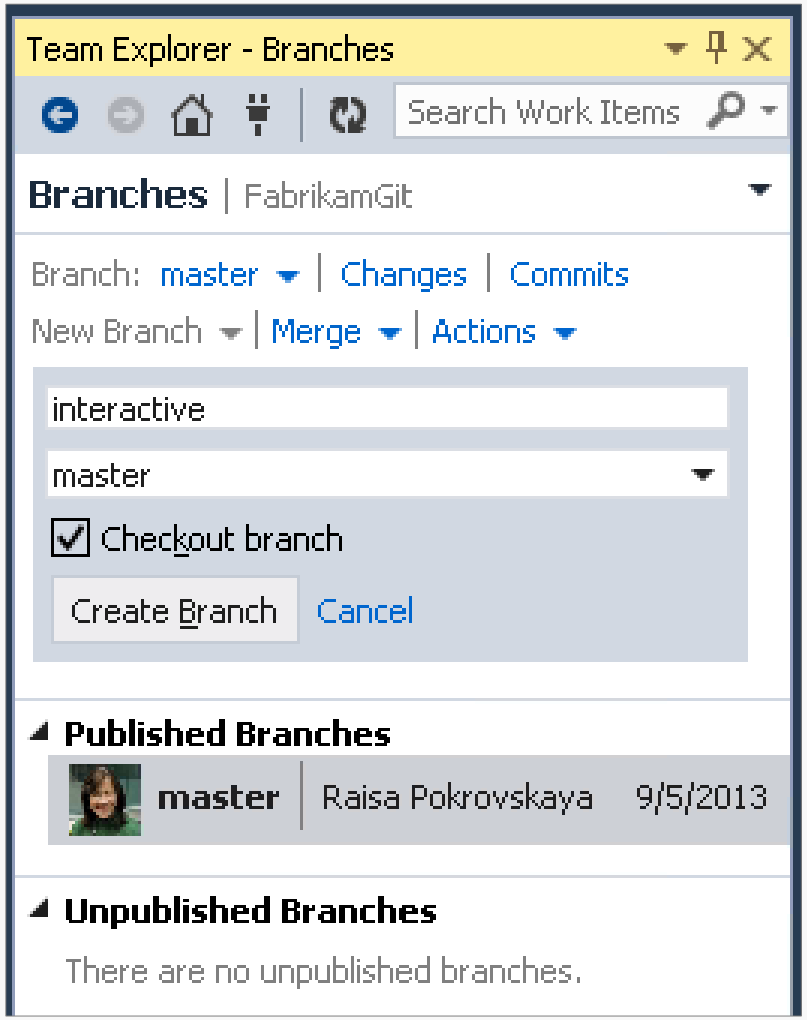
- Add additional repositories to your Git team project from web portal



VS Git Plugin – Branches

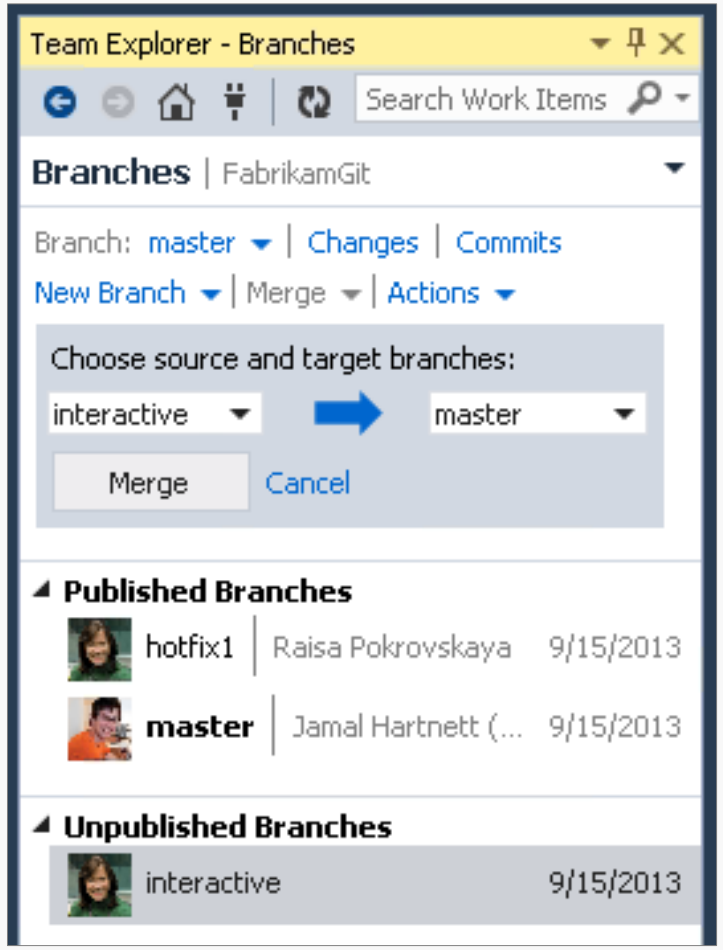
Create a Branch

- Create a branch from the branches page



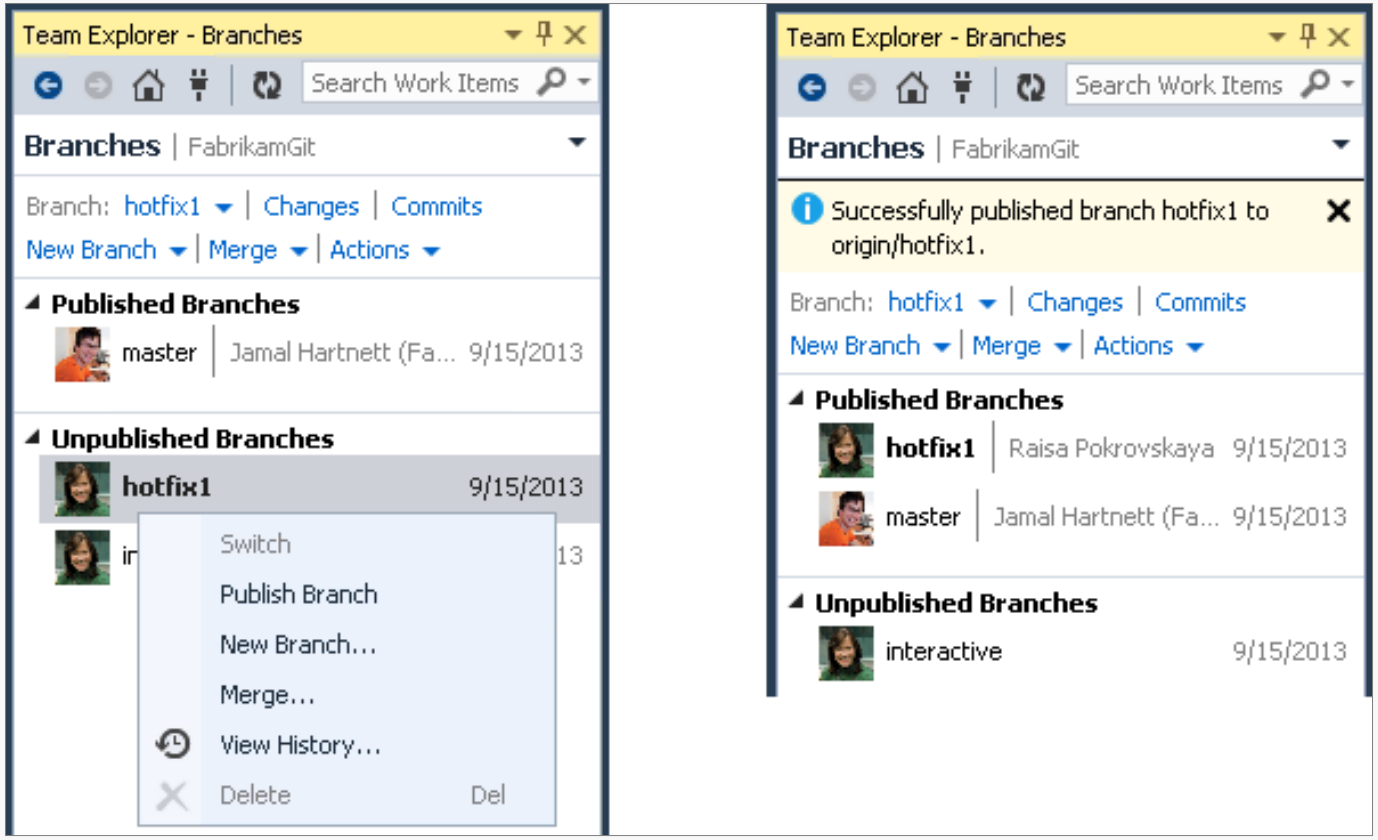
Merge a Branch

- Merge the work you have done in one branch into another branch



Publish a Branch

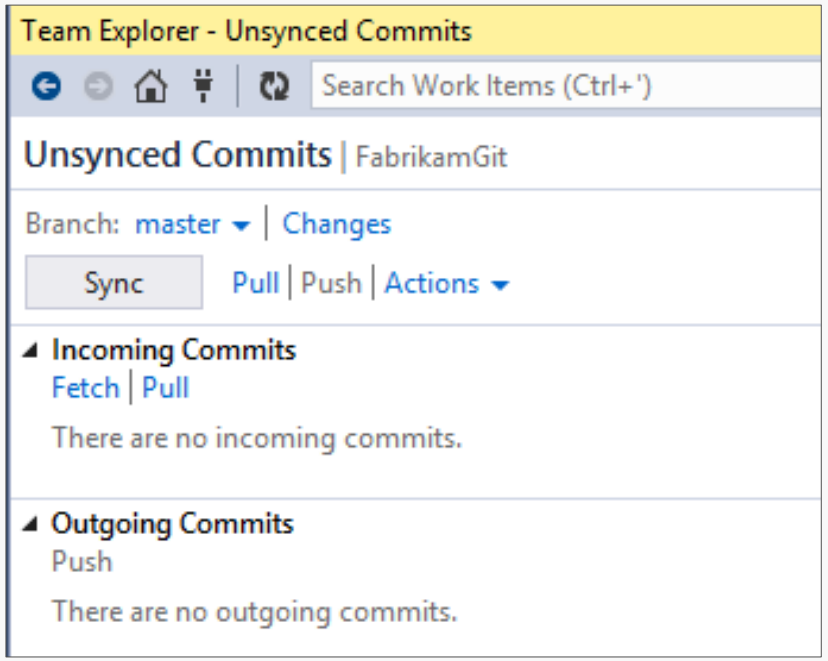
- Collaborate or preserve the work you have done on a branch by publishing it



VS Git Plugin – Conflicts and Merging

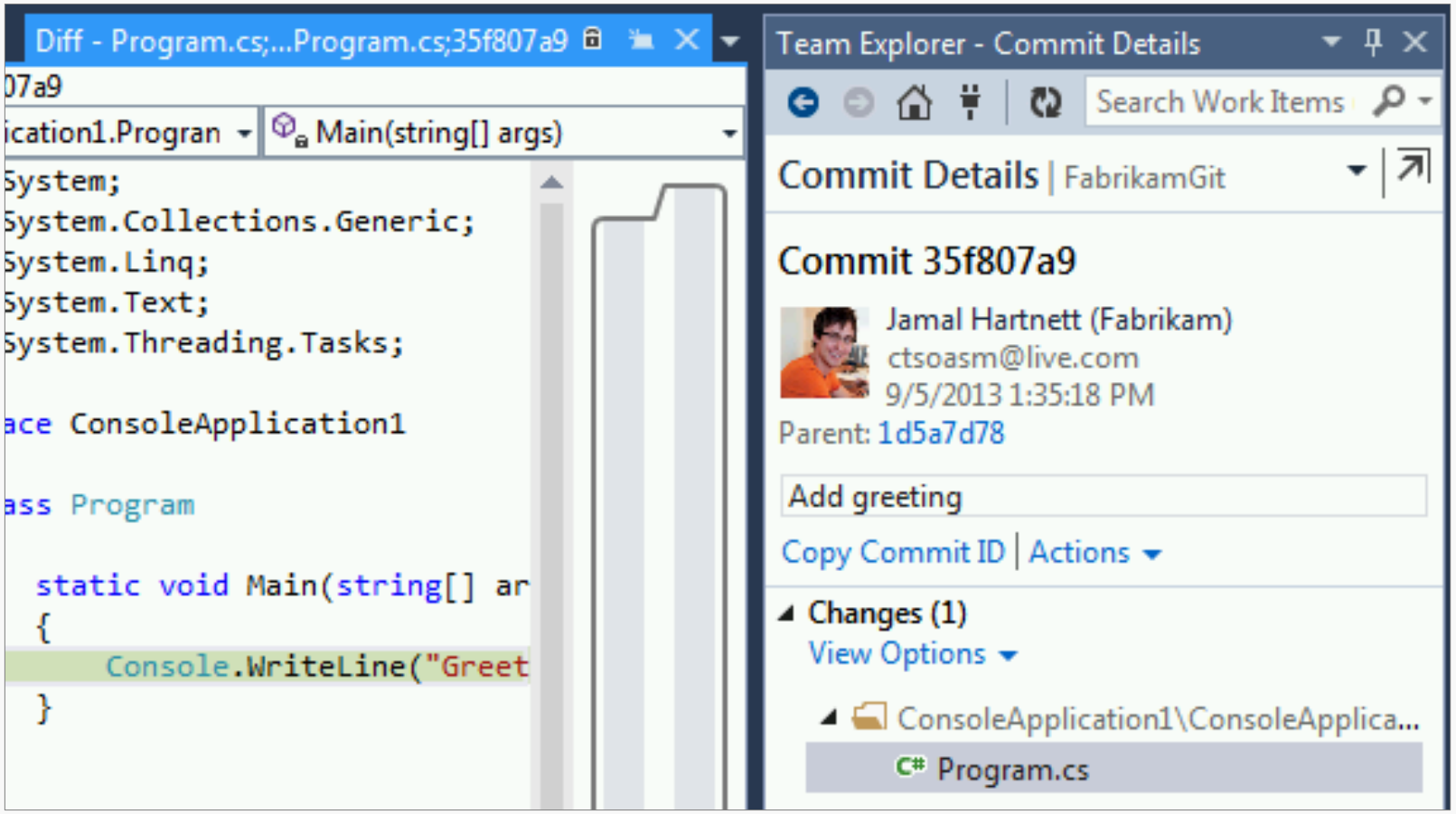
Examples of Conflicts—Pull

- Raisa wants to push a change to a file
- If any commits have been pushed since her last pull, she must pull them down before she can push her commit



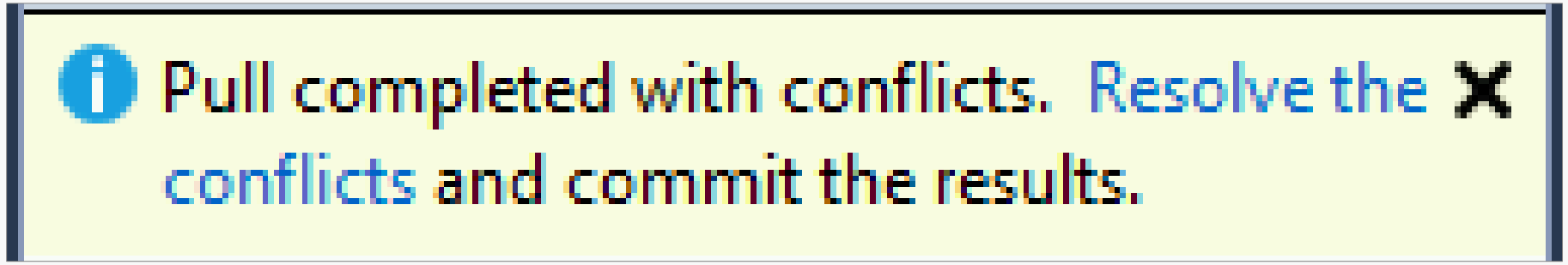
Examples of Conflicts—Pull (continued)

- She can view details about the incoming commit from Jamal and see that he has modified the same line of code that she modified



Examples of Conflicts—Pull (continued)

- When she tries to pull, Visual Studio shows her the conflict



Resolve Content Conflicts

Merge - vctmp2272_...Program.eb1d3e41.cs*

Accept Merge

1 Conflicts (0 Remaining)

Source: ConsoleApplication1\ConsoleApplicat...

ConsoleApplicatio | Main(string[] args

```
lain(string[] args)

riteLine("Greetings, planet.");
```

100 %

Target: ConsoleApplication1\ConsoleApplicati...

ConsoleApplicatio | Main(string[] args

```
ain(string[] args)

riteLine("Hello world!");
```

100 %

Result: ConsoleApplication1\ConsoleApplication1\Program.cs

ConsoleApplication1.Program | Main(string[] args)

```
:ic void Main(string[] args)

Console.WriteLine("Hello world!"); //greet user
```

Team Explorer - Resolve Conflicts

Search Work

Resolve Conflicts | FabrikamGit

Commit Merge | Undo Merge

Conflicts (1)

Name	Path
Program.cs [both modified]	Console...

Merge

Conflicting content changes have been made.
Compare Files

Edited on **Remote** | Diff | Take Remote
Edited on **Local** | Diff | Keep Local

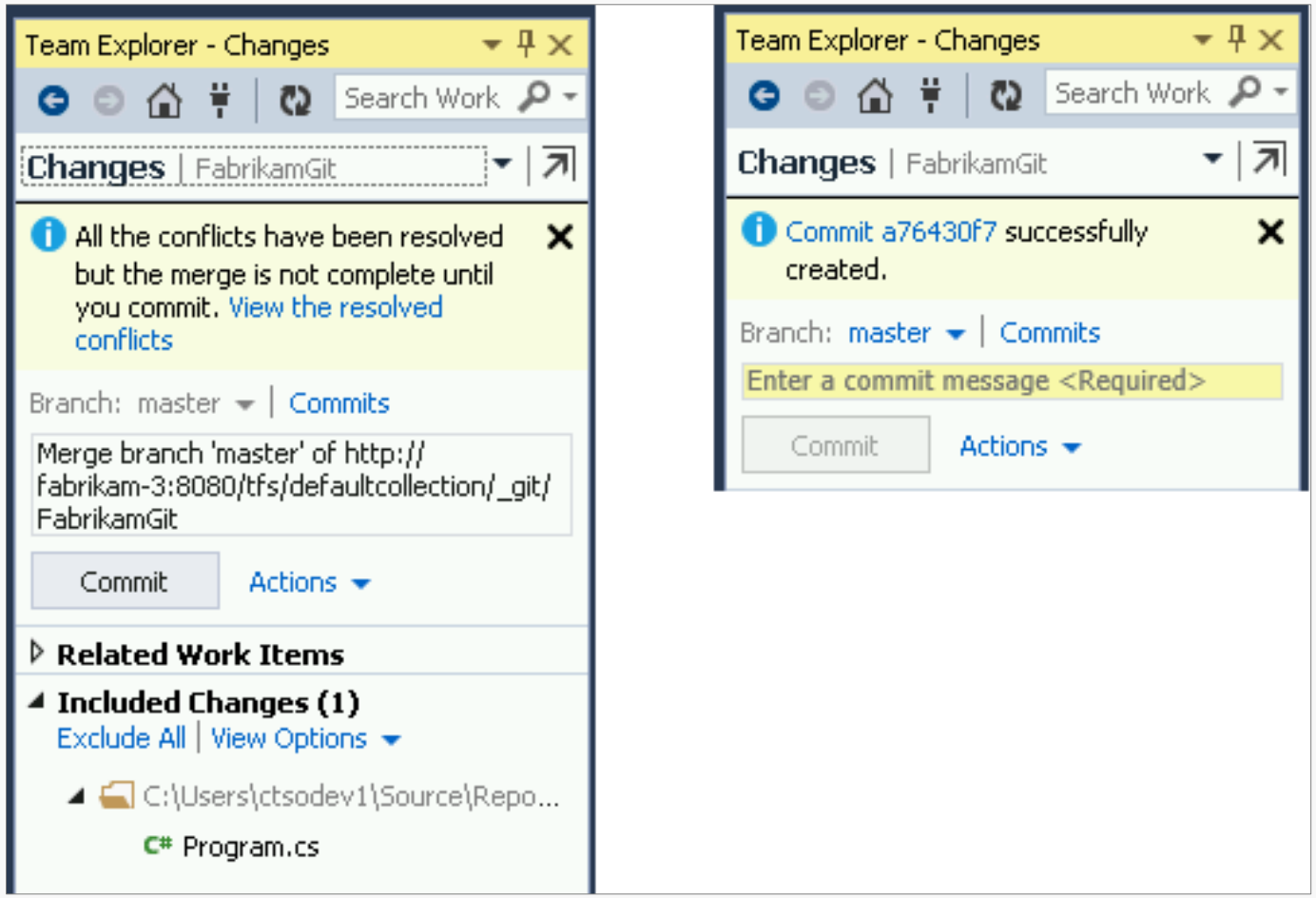
Resolved
There are no resolved conflicts

Commit the Merge

- You can commit the merge if you are ready
- Or, if you cannot resolve all the conflicts, you can undo all your resolutions

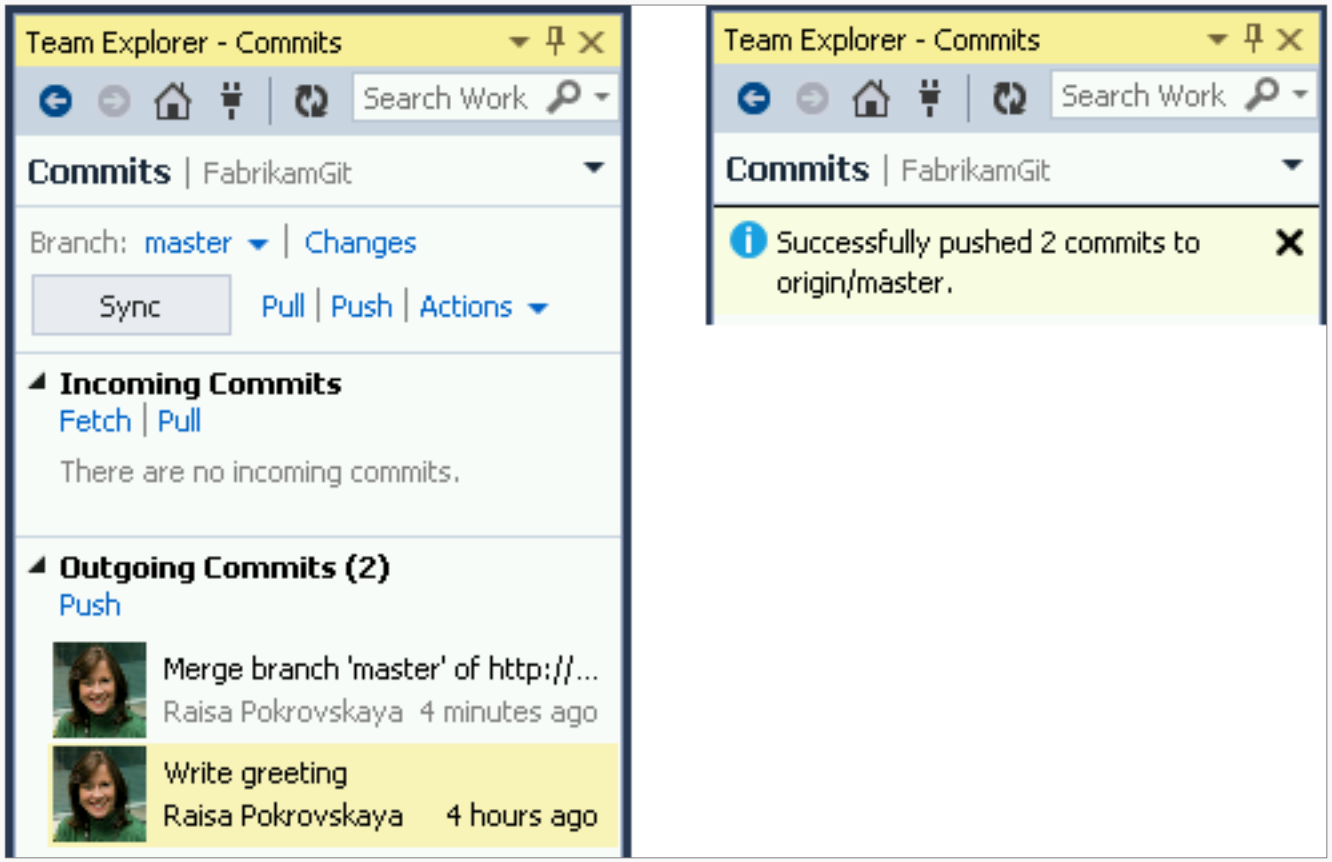
Commit the Merge (continued)

- Commit the merge



Commit the Merge (continued)

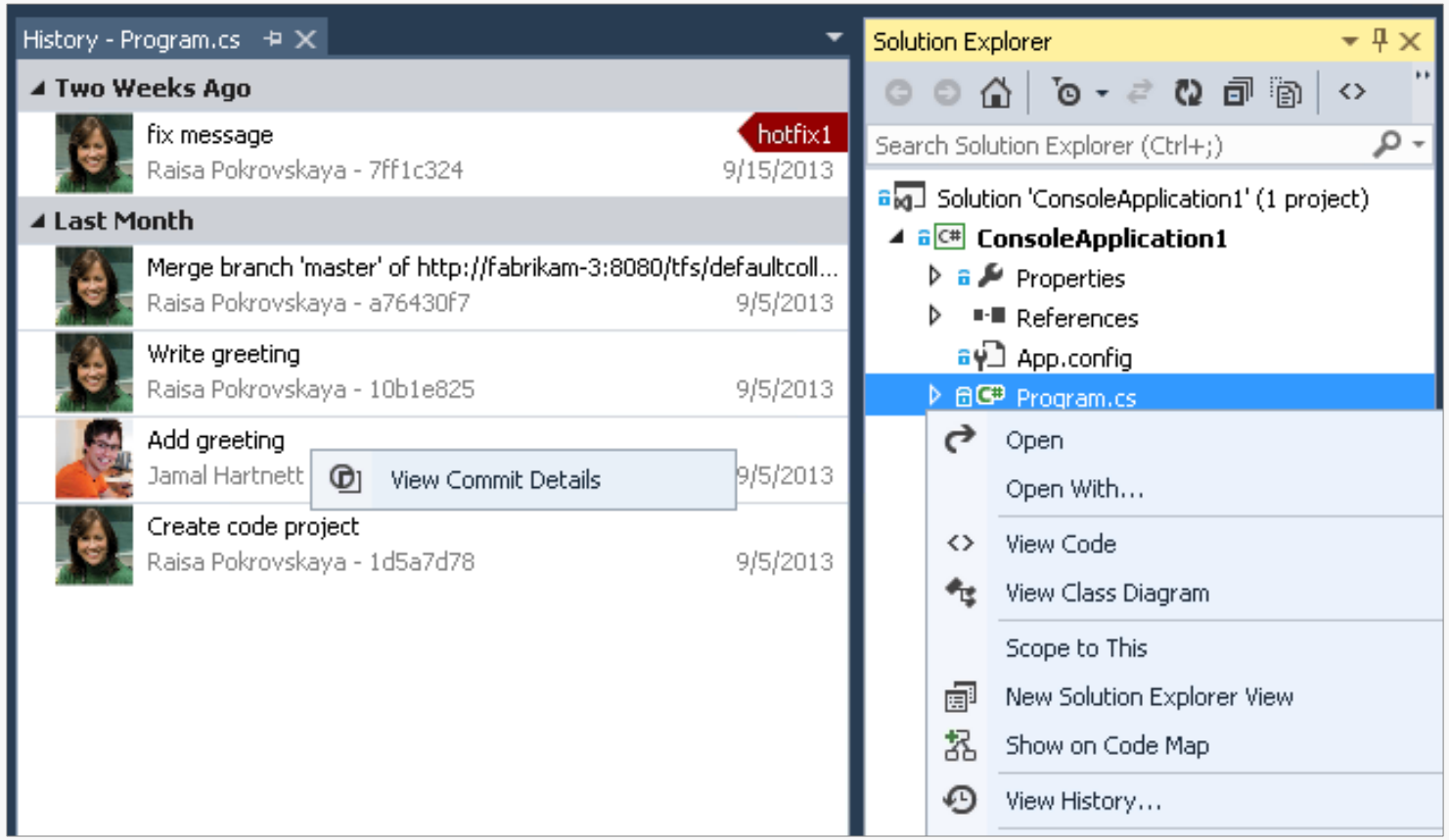
- Push changes into the remote repository



VS Git Plugin – History

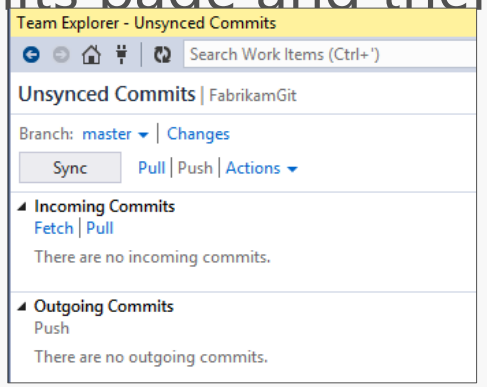
View Historical Data in Visual Studio

- File History

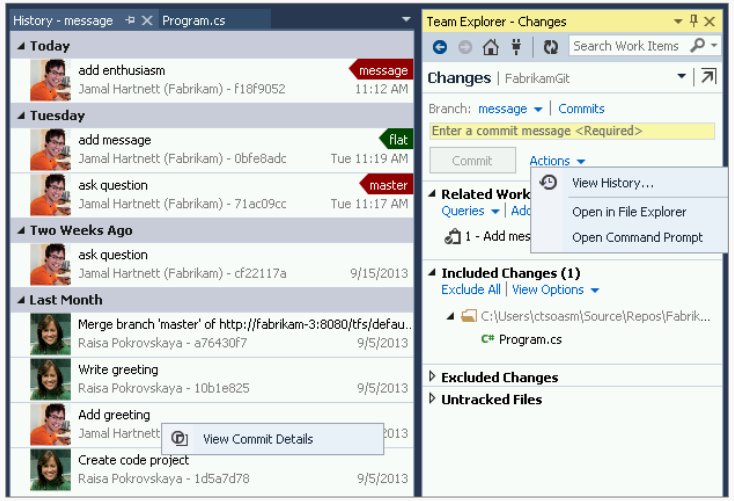


View Historical Data in Visual Studio

- Branch History
  - Go to the Unsynced Commits page and then fetch the latest changes

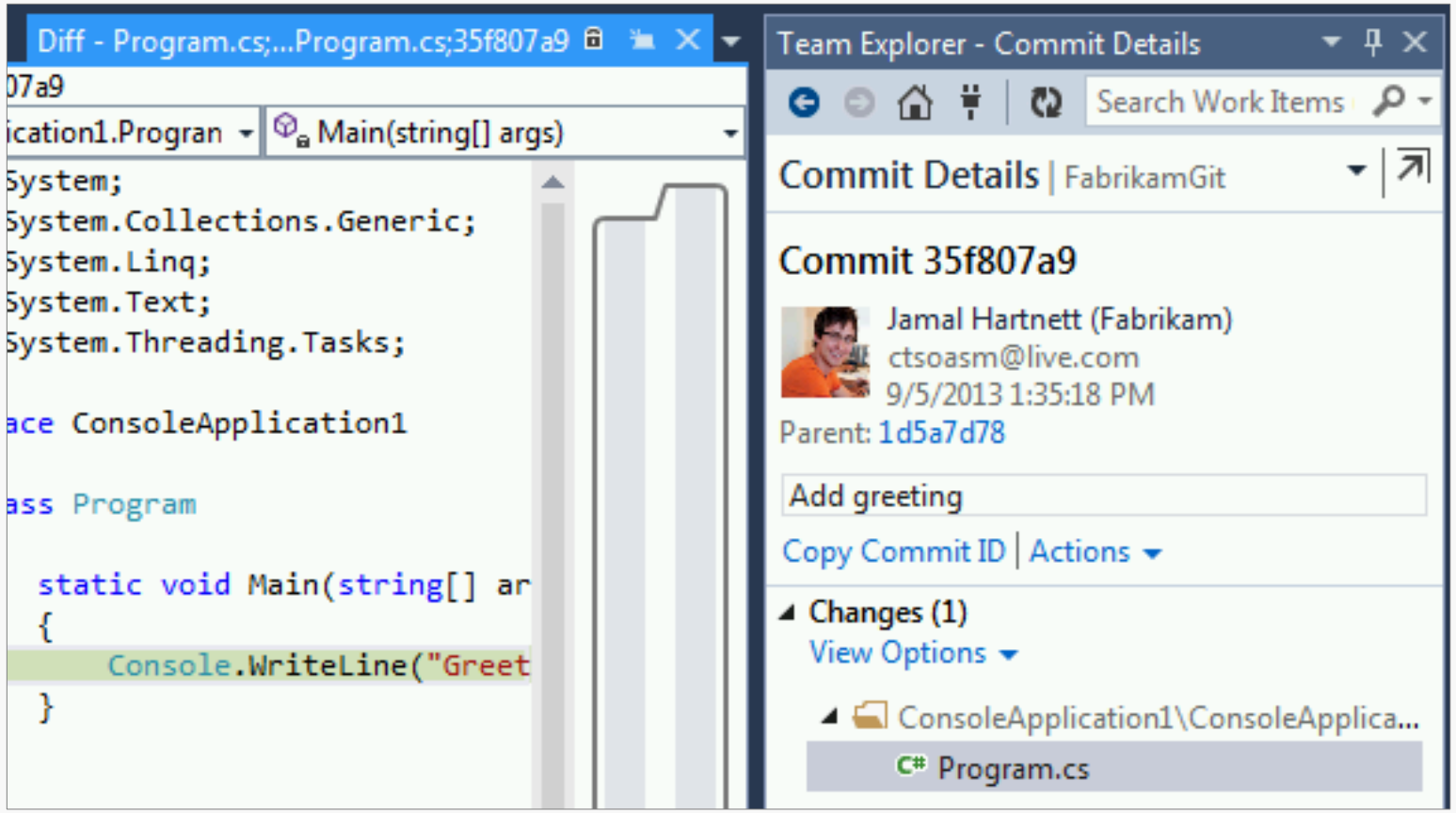


  - View the history



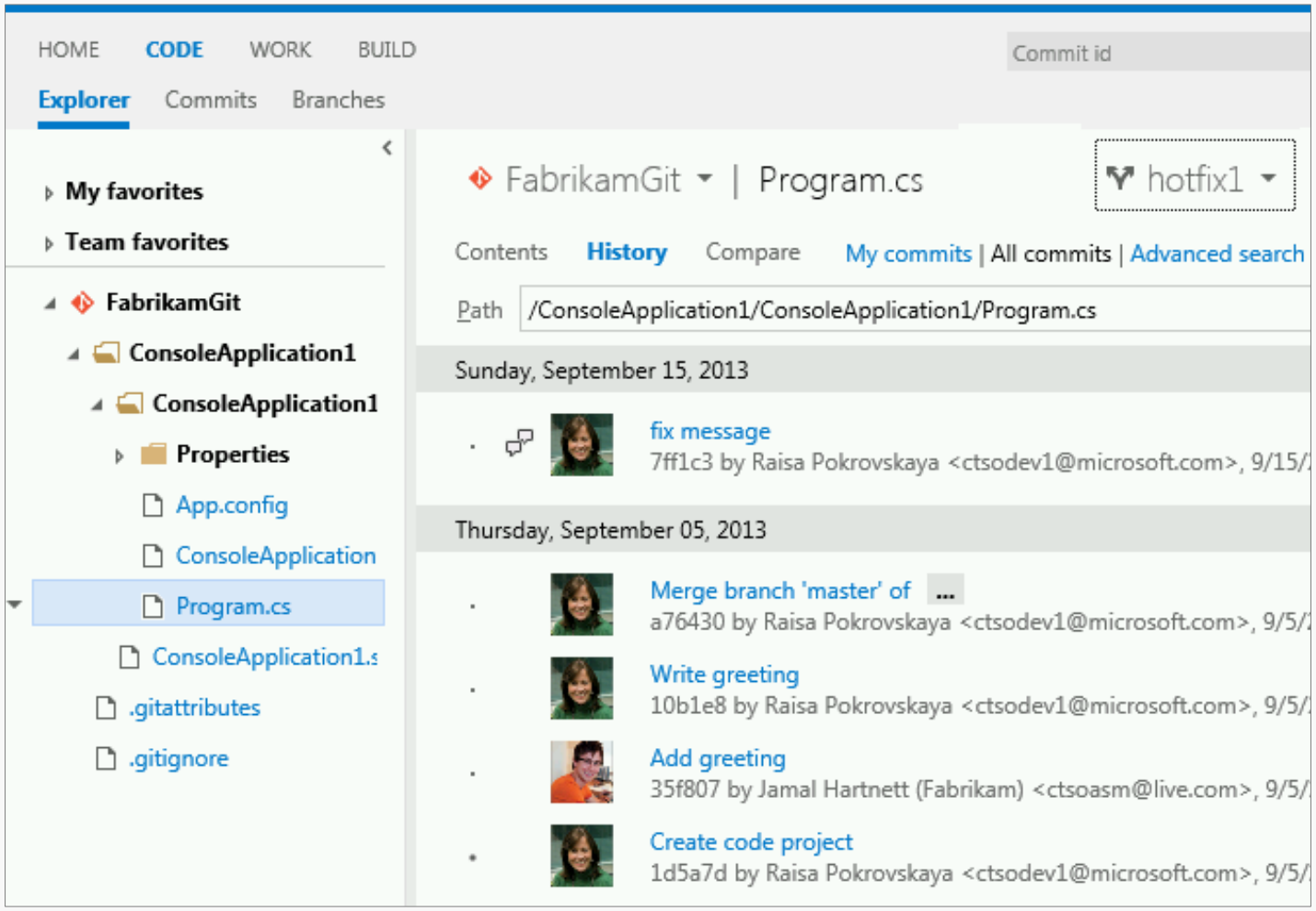
View Historical Data in Visual Studio

- Commit Details



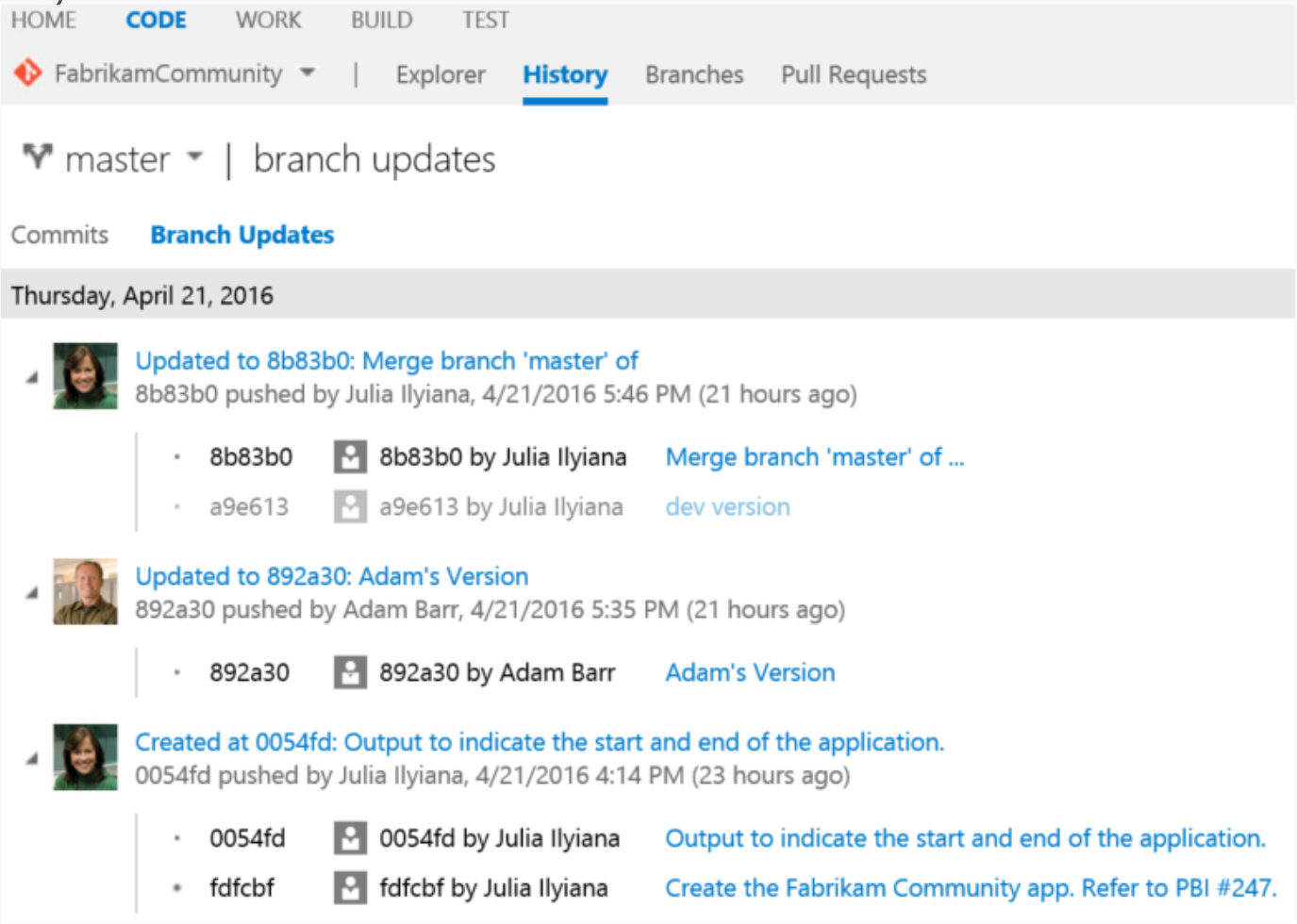
View Historical Data in Web Browser

- File History



View Historical Data in Web Browser (continued)

- Branch History



View Historical Data in Web Browser (continued)

- Commit Details

HOME CODE WORK BUILD TEST

FabrikamCommunity | Explorer History Branches Pull Requests

Commit explorer

Create the Fabrikam Community app. Refer to PB #247.

Julia Ilyiana authored fdfcbf
Thursday, April 21, 2016 4:12 PM

1 associated work item

Product Backlog Item 247 Create HelloWorld Java App
Current state is New. Currently not assigned to anyone.

4 items added

/

.classpath [+]

.gitignore [+]

.project [+]

/src [+]

FabrikamCommunity.java [+]

```
1
2  public class FabrikamCommunity {
3
4          public static void main(String[] args) {
5                  // TODO Auto-generated method stub
6                  System.out.println("Hello Fabrikam Community");
7          }
8
9  }
10
```

View Historical Data in Eclipse

- Change made on the Development branch

Problems | Javadoc | Declaration | Console | Git Staging | History

Repository: FabrikamCommunity

Id	Message	Author
a9e6132	Development HEAD dev version	Julia Ilyiana
0054fdc	master origin/master Output to indicate the start and end of the application.	Julia Ilyiana
fdfcbf4	Create the Fabrikam Community app. Refer to PBI #247.	Julia Ilyiana

Repository: FabrikamCommunity

Id	Message	Author
3bb3126	master HEAD Merge branch 'master' of http://vsalm:8080/tfs/Fabrika	Julia Ilyiana
436d3ad	origin/master Adam's Version	Adam Barr
2861501	Development origin/Development Dev version	Julia Ilyiana
1c4cf85	Output to indicate the start and end of the application.	Julia Ilyiana
237bb3d	Create the Fabrikam Community app. Refer to PBI #247.	Julia Ilyiana

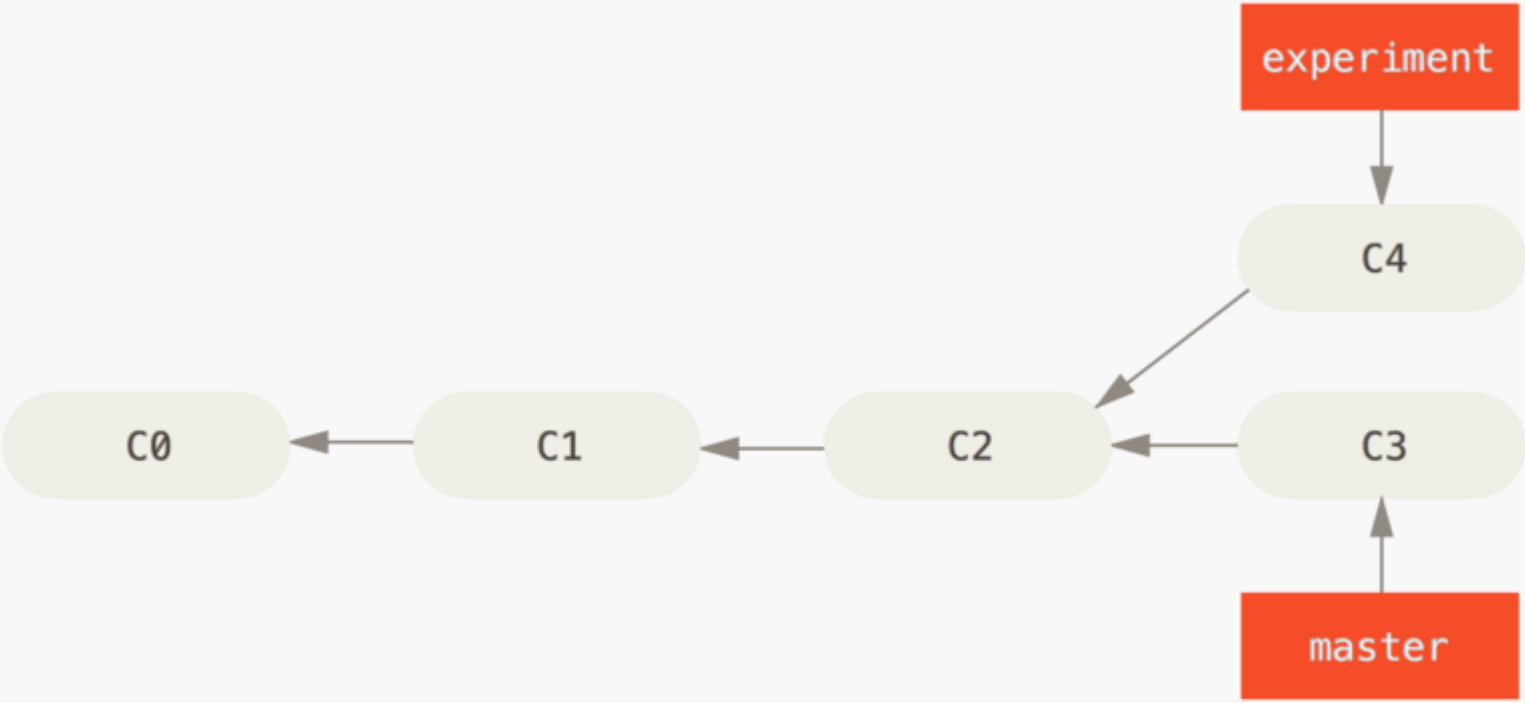
Rebase or Merge?

Rebase or Merge

- In Git, there are two main ways to integrate changes from one branch into another: the merge and the rebase.
  - **Merge** takes all the changes in one branch and merges them into another branch in one commit.
  - **Rebase** is recreating your work of one branch onto another. For every commit that you have on the feature branch and not in master, new commit will be created on top of the master. It preserves the original commits.
  - The project's history then looks as if it had evolved in a single, straight line. No indication remains that it had been split into multiple branches at some point.

Merge

- The easiest way to integrate the branches is the merge command. It performs a three-way merge between the two latest branch snapshots (C3 and C4) and the most recent common ancestor of the two (C2), creating a new snapshot (and commit).



Rebase

- Say we are on the feature branch. We want to rebase our feature branch work with the current master branch.
- We pick up our work from commits 4 and 6, Git will undo these temporarily and store them for us.
- Our feature branch will take on the current master branch, which has commits 3 and 5.
- Git will now reapply our commits 4 and 6 we temporarily saved, but will give those commits a new ID. Change 4 gets committed as 7. Change 6 gets committed as 8. It's the same code changes, just a new ID.

Start

```
      +--3--5   master
      |
1--2--+
      |
      +--4--6   feature
```

End

```
      +--3--5   master
      |
1--2--+
      |
      +--3--5--7--8   feature
              (4)(6)
```

Fast-Forward Merge

- The only time a merge creates no new commits is the fast-forward merge. It happens in a situation when there are no commits in an another branch.
- After you do fast-forward merge, you will have:

Start

```
      +--3--5   master
      |
1--2--+
      |
      +--4--6   feature
```

End

```
      +--3--5   master
      |
1--2--+
      |
      +--3--5--7--8   feature
            (4)(6)
```

```
1--2--3--5--7--8   master/feature
```

Which one do I want to choose?

- Merge

Let's say you have created a branch for the purpose of developing a single feature. When you want to bring those changes back to master, you probably want merge (you don't care about maintaining all of the interim commits).

- Rebase

A second scenario would be if you started doing some development on your feature branch and then another developer made an unrelated change (that you wanted in your branch). You probably want to pull and then rebase to base your feature branch changes on top of the newer version.

Microsoft