**Implement a code workflow in your build pipeline by using Git and GitHub**

**Learning objectives**

In this module, you will:

* Collaborate with others by choosing an appropriate branching and merging strategy.
* Add a badge to your GitHub repository to show the status of the latest build.
* Add a dashboard widget to help visualize your build history.
* Set up a rule on your GitHub repository to require a review.

**Prerequisites**

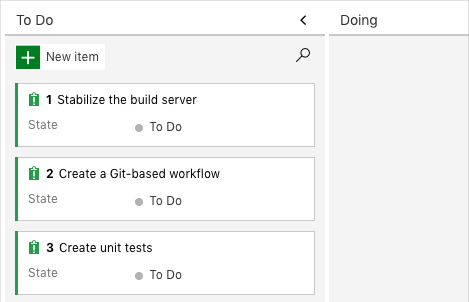
* An Azure DevOps organization
* Visual Studio Code
* .NET Core 3.1
* Git
* A GitHub account

**Choose a code flow strategy**

* 8 minutes

It's important to choose a code flow strategy that fits in with how your team works. There are several strategies to consider, and you'll find links that explain them at the end of the module. The Tailspin web team uses a code flow strategy that's based on Git and GitHub.

When Mara set up Azure Boards, she and the team identified a few initial tasks to address. One task was to create a Git-based workflow.



Let's listen in on the team as they work out a better way to collaborate. Currently, they're using a centralized version control system, but the plan is to move to Git, a distributed system.

Mara is diligently working on her assigned features when Andy walks in.

**Andy:** Hi Mara. In the leadership meeting this morning it was brought up that our team and the game development team are using different version control systems. In an effort to streamline how we share resources between teams we have been asked to move to a distributed version control system that can better handle the collaboration.

**Mara:** That's good to know. If you remember, we put it on our board. Currently, we are using a centralized version control system. This works great for us now, but I agree that when we start to share between teams and our team gets bigger, a distributed version control system is a better choice. It's also a task on our board to increase visibility so all the stakeholders know what everyone is doing. I think a distributed source control system like Git would help there as well.

**Andy:** I've been wanting to try Git for a while. I just never seem to have the time. Is it difficult to learn or set up? If it seems reasonable, maybe we could work on it now. I'm tired of always putting things off. And it would be nice to be able to see what everyone is doing and to have access to the entire repository. OK, what's it all about?

**Mara:** Let me explain it, and then you can decide if it sounds like something we want to implement right away.

Mara and Andy move to the whiteboard for a discussion on version control.

**What is Git and distributed version control?**

A hand-drawn illustration of centralized versus distributed source control

**Mara:** The drawing on the left is *centralized version control*, like what we're using now. We have a central version of the code base Callout 1 in Team Foundation Version Control (TFVC) that everyone uses. We each work on the files we need to change and then merge them back into the master repository when we're finished with them.

**Andy:** Yes, and that is working for us. Well, except when I was blocked that time that a breaking change got merged into the central repo.

**Mara:** Right! You were blocked Callout 2. We could use a branching strategy with TFVC to solve the blocking issue but, in our current configuration, merging might get a bit more complicated. And when we had that breaking change Callout 3 no one could get any work done until we got that resolved. That problem is always lurking, because we're all using the same copy of the code.

On the right is a drawing of *distributed version control*. We still have a main, or master, repository Callout 4 that's the stable version of the code base, but each developer has their own copy Callout 5 of it to work from. This frees us up to experiment and try a variety of approaches without affecting the main repository.

Distributed version control also ensures that only working code Callout 6 gets merged into the master repository. We could even set it up to where code can't be merged until it's been reviewed.

What's cool about Azure DevOps is that it works well with both centralized and distributed version control systems.

**Andy:** What happens when more than one person changes the same file?

**Mara:** Often, Git can merge multiple changes automatically. Of course, we want to always make sure the combination of changes results in working code. When Git can't automatically merge changes, it marks the conflicts directly in the files so that a human can choose which changes to accept.

**Andy:** Right now, our code is stored in our own server. Where will it be if we move to using distributed version control?

**Mara:** I'm glad you asked. That's where hosting comes in.

**Where can I host my repository?**

**Mara:** When we're deciding where to host our repositories, we have a few options. For example, we can host them on a local server, in Bitbucket, or in GitHub. Bitbucket and GitHub are web-based hosting solutions. We can access them from anywhere.

**Andy:** Have you used either of them?

**Mara:** I've used GitHub in the past. It has features that are important to developers, like easy access to change logs and version control features from either the command line or the online portal.

**Andy:** So how does Git work?

**How do I work with Git?**

**Mara:** Like I mentioned before, with distributed systems, developers are free to access any file they need without affecting other developers' work, because they have their own copy of the repository. A *clone* is your local copy of a repository.

When we work on a feature or a bug fix, we usually want to try out different approaches until we find the best solution. But trying out code on your copy of the main codebase isn't a good idea, because you may not want to keep the first few tries.

To give you a better option, Git has a feature called *branching*, where you can maintain as many copies as you want and merge back only the one you want to keep. This keeps the main branch stable.

**Andy:** I get the concepts so far. How do I check in my code?

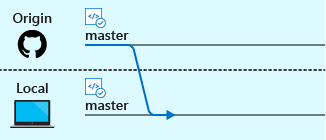
**How do my local changes get up to the main codebase?**

**Mara:** In Git, the main branch, or *trunk*, is typically called master.

After you feel that your code is ready to be merged into the master branch in the main repository that's shared by all developers, you create what's called a *pull request*. When you create a pull request, you're telling the other developers that you have code ready to review and you want it merged into the master branch. When your pull request is approved, it becomes part of the master codebase.

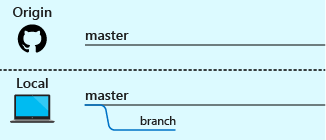
**What does a branching workflow look like?**

**Step 1**: When you begin to work on a new feature or bug fix, the first thing you want to do is make sure you're starting with the latest stable codebase. To do this, you can synchronize your local copy of the master branch with the server's copy. This pulls down all other developers' changes that have been pushed up to the master branch on the server since your last synchronization.

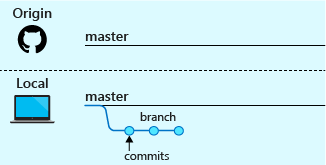


**Step 2**: To make sure you're working safely on your copy of the code, you create a new branch just for that feature or bug fix. As you can imagine, having many branches for all the things you're doing might get hard to remember, so a good naming convention is critical here.

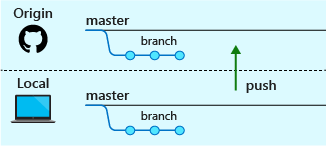
Before you make changes to a file, you check out a new branch so that you know you're working on the files from that branch and not a different branch. You can switch branches anytime by checking out that branch.



**Step 3**: You're now safe to make whatever changes you want, because these changes are only in your branch. As you work, you can *commit* your changes to your branch to make sure you don't lose any work and to provide a way to roll back any changes you've made to previous versions. Before you can commit changes, you need to stage your files so that Git knows which ones you're ready to commit.

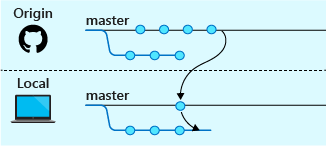


**Step 4**: The next step is to *push*, or upload, your local branch up to the remote repository (such as GitHub) so that others can see what you're working on. Don't worry, this won't merge your changes yet. You can push up your work as often as you'd like. In fact, that's a good way to back up your work or enable yourself to work from multiple computers.

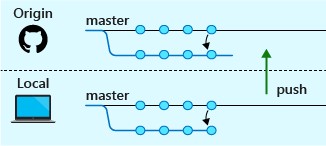


**Step 5**: This step is a common one, but not required. When you're satisfied that your code is working as you want it to, you can *pull*, or merge, the remote master branch back into your local master branch. Changes have been taking place there that your local master branch doesn't have yet. After you've synchronized the remote master branch with yours, merge your local master branch into your working branch and test your build again.

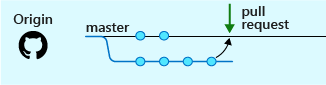
This process helps ensure that your feature works with the latest code. It also helps ensure that your work will integrate smoothly when you submit your pull request.



**Step 6**: Your local code now needs to be committed and pushed up to the hosted repository. This is the same as steps 3 and 4.



**Step 7**: You're finally ready to propose your changes to the remote master branch. To do this, you begin a pull request. When configured in Azure Pipelines or another CI/CD system, this step triggers the build process and you can watch your changes move through the pipeline. After the build succeeds and others approve your pull request, your code can be merged into the remote master branch. (It's still up to a human to merge the changes.)



**Andy:** This all looks complicated and hard to learn.

**Mara:** Git can seem intimidating because it's so powerful. But after you get the hang of the flow, it starts to feel natural.

You'll use only a few commands daily. Here's a summary:

|  |  |  |
| --- | --- | --- |
| TABLE 1 | | |
| Category | **To perform this task** | **Use this command** |
| Repository management | Create a Git repository | git init |
|  | Download a remote repository | git clone |
| Branch | Create a branch | git checkout |
| Stage and commit changes | See which files have been changed | git status |
|  | Stage files to commit | git add |
|  | Commit files to your branch | git commit |
| Remote synchronization | Download a branch from a remote repository | git pull |
|  | Upload a branch to a remote repository | git push |

**Andy:** That sounds like a great starting point. I can definitely handle that. I can learn more advanced commands as I need them.