madcap software

USER GUIDE

DOCUMENTATION BIBLE

Source Control

MadCap Software
9191 Towne Center Drive, Suite 150
San Diego, California 92122
858-320-0387
www.madcapsoftware.com

THIS PDF WAS CREATED USING MADCAP FLARE.

CONTENTS

CHAPTER 4

CHAPTER 1

Introduction

Source control is essential for us to work efficiently as a team and to keep different iterations of documentation organized.

This chapter discusses the following:

Our Projects

Because we have different types of projects, the way source control is used varies:

- Shared Project—Single-Bound For the main Shared project, we use a single-bound source control structure. The source control binding is with Central, which is MadCap's cloud-based platform for managing projects across an organization; a Git-based repository and program. The main reasons we bind our Flare "Shared" project with Central are: its branching capabilities, its checklist features, and its ability to quickly build and host output.

  The Flare interface supports basic functionality for enabling a single-bound project with Central. Simple tasks to complete are: importing projects, creating and publishing branches, committing content changes, synchronizing pushes and pulls, or showing differences in text. On occasion, there are advanced source control tasks required (e.g., creating tags in branch history, troubleshooting issues), and for those we use Visual Studio, Git CMD, or Git Bash to complement our other single-bound tools.

- Documentation Bible and "Develop" Projects—Single-Bound The Documentation Bible and our "Develop" projects all retrieve files from the Shared project via Global Project Linking, and have only one source control binding to Central. We simply commit our changes in Flare and then synchronize the projects with their clones on Central. This configuration is necessary with our "Develop" projects because it allows us to import files from the develop branch in the Shared project, and then synchronize the child project's master branch with Central. This lets us manage checklists on Central for content that is in development, since Central does not yet support branching for checklists.

How We Use Source Control

See the following (preferably in order) for more details on different concepts and how we use source control:

Tasks and Procedures

See the following for details on the various tasks that we perform with source control:

- "Daily Tasks" on page 26
- "Occasional Tasks" on page 47
- Step-by-Step Procedures:
  - "Opening the Shared Repository" on page 30
  - "Viewing Branches" on page 31
  - "Creating Branches" on page 48
  - "Getting Branches" on page 52
  - "Checking Out Branches" on page 32
  - "Pulling Commits" on page 34
  - "Committing and Pushing Changes" on page 43
  - "Merging From Release to a Feature Branch" on page 35
  - "Merging From Release to the Develop Branch" on page 38
  - "Merging From a Feature Branch to the Develop Branch" on page 40
  - "Merging From a Feature Branch to the Release Branch" on page 53
  - "Merging From Release to the Master Branch" on page 58
  - "Dealing With Issues, Conflicts, and Messages" on page 63
  - "Reverting Commits" on page 74
  - "Using Hot Fixes to Update Old Outputs" on page 76
  - "Deleting Old Feature Branches" on page 79

CHAPTER 2

How We Use Source Control

You should become quite familiar with different concepts and the way that we use source control on the MadCap Documentation Team.

This chapter discusses the following:

Branches

Our source control process is based on a method called “Gitflow”:

https://datasift.github.io/gitflow/IntroducingGitFlow.html

This system allows for continuous, parallel development and collaboration through the use of branches. Ours is a simplified version of the Gitflow system, and we’ve tailored it just for our needs.

Here's our movie about it: https://youtu.be/rukOqOihVWQ

A branch is not a separate copy of a Flare project. It is the same project but with changes recorded independently of the other branches. One person could be documenting a certain feature in one branch while another writer documents a different feature in a second branch. Then, documentation from each of those branches could be merged into yet another branch. But it is still the same project.

In our process, we use the following types of branches:

- Master This is the *current version of the documentation*; its output is available to end users. This branch is typically ignored until the very end of a release cycle. Unlike the other types of branches, most writers should not be merging changes into the master branch at all. Only one writer will merge changes to the master branch when the time is right.
- Release This is the *next version of the documentation*. It holds changes from finished feature branches as development progresses. Each writer merges changes from the feature branch to the local release branch when the documentation for that feature is ready and it is assured the feature will be included in the next release, rather than backlogged. Usually, we wait for the programmers to move the feature into *their* release branch before we move the documentation into *our* release branch.
- Feature There are multiple feature branches, and they are created as necessary to write *documentation for new features*. This is where the bulk of the actual work is done. Changes in feature branches might be part of the next version of the documentation (i.e., the release branch) or they might be held back for another product release. We organize our feature branches according to the product (e.g., Flare, Capture, Lingo) or under the banner of “Global” (if the changes pertain to multiple products).
- Develop This branch is a *staging area*, and consists of various feature branches as they are being developed. They may or may not be included in the next release of the product. With the develop branch checked out, we import files that are in progress into a child “Develop” Flare project. The changes in the “Develop” project are then pushed up to Central. From there, we can create and manage checklists to track the progress of files for a given release cycle. In addition to merging individual feature branches into the develop branch, writers can merge

the release branch into the develop branch periodically to get the latest files. The develop branch should NEVER be merged into any other branch.

Before We Begin

Here are a few things to know before you delve into the source control process:

- Flare and Central We create and manage branches using mainly Flare and Central. You need to have Flare installed, connectivity to a licensed version of Central, and a binding between the two products for the Shared project's repository.
- Visual Studio 2017 Flare supports branching and basic source control functionality. It is a good idea to install Visual Studio 2017 for advanced source control tasks, or for things that Flare does not support yet. Git Bash should also be installed for other source control tasks. It is perfectly fine to use Visual Studio and/or Git Bash in conjunction with Flare and Central for additional tasks.
- Understand Commits A commit allows you to add your local changes in Flare to the local Git repository; you can later push those changes to the remote repository (on Central). A commit is really a collection of your changes, and you decide what goes into that commit. Sometimes you might have a group of unrelated changes that get added to the same commit. But it is preferable that you try to work on logical chunks of content, consolidate them into a single commit, add a comment to the commit that best describes those changes, and then push that commit to the remote repository. Doing it this way helps us identify certain changes that we might have to pull out of a branch later if necessary.

  When you move or rename files, Git sees those actions as deleting the old file and then adding a new file with the same content (just in a separate location or with a different name).
- Remote and Local Branches There are remote branches that can be accessed by all writers, and there are local versions of branches as needed by each user (e.g., there might be a remote version of a feature branch called "style-editor" and a local version of the same name on a writer's computer). One writer might have some local feature branches, whereas another writer might have entirely different feature branches. It all depends on who is working on what.
- Moving Changes Between Branches Changes in a branch can be synchronized (i.e., pulled and pushed) between remote and local versions of the same branch. Changes can also be moved (i.e., merged) from one branch to another (e.g., merge changes from a feature branch into the release branch). Theoretically, any branch could be merged with any other branch. But in practice, we want to maintain rules as to which branches should be merged with which, when, and by whom. Merging is typically done between local branches (e.g., merging from the local release branch into the local feature branch), and then the resulting commits are pushed from the local branch up to the remote branch.

- Focus on Daily Tasks There is a lot of information to absorb about our source control process, and it might not all make complete sense immediately. That's okay. As you work in this system, it will become more familiar over time. Also, keep in mind that some tasks need to be done only occasionally. Although you should read this source control documentation in its entirety, it is recommended that you mostly focus at first on the tasks that you need to do every day. See "Daily Tasks" on page 26.

Best Practices and Rules

Following are a few important best practices and rules to keep in mind as you work with this source control system:

- Work in Correct Branch When a branch is selected, Flare automatically adjusts to show only the changes that are part of that branch. If you check out a different branch, Flare adjusts to that branch. There isn't anything special you need to do, but it is important to always look at which branch is selected in Flare before you begin making a lot of changes. See "Checking Out Branches" on page 32.
- Pull Then Commit/Push One of the most important rules when dealing with branches in Git is to always do a pull from the remote branch *before* you commit and push files. This helps keep things in sync better and results in a cleaner history. In other words, every time you check out a branch, you should immediately do a pull; there's no downside to doing a pull and lots of upside. And after you've done some work and create a commit, you should use the Synchronize option, which automatically does a pull before it pushes your commit(s). See "Pulling Commits" on page 34 and "Committing and Pushing Changes" on page 43.
- Sync Release Branches Although most work is done within feature branches, you also need to pull changes every day from the remote release branch into your local release branch. And when the time comes toward the end of the release cycle that you merge changes to the local release branch, you'll also need to push those changes up to the remote release branch. See "Pulling Commits" on page 34 and "Committing and Pushing Changes" on page 43.
- Merge Between Release and Feature Branches Merging from a feature branch to the release branch *should only be done when you are confident the documentation is ready and the feature will definitely be included in the next product release*. See "Merging From a Feature Branch to the Release Branch" on page 53.

  Merging from the release branch to feature branches should be done daily. That's because other writers might have finished working on some of their feature branches and merged them into the release branch, and you want to make sure you integrate those changes into the feature branches you are working on. Doing this regularly helps to offset potential issues that could arise. The longer you wait between merges, the greater the chance for file conflicts, especially those where it becomes hard to remember exactly what you did in particular file. See "Merging From Release to a Feature Branch" on page 35.

- Never Merge From Develop There is a lot of merging happening in both directions between different branches. This includes merging the release and feature branches into the develop branch regularly. One thing you should NEVER do is merge from the develop branch into any other branch. The develop branch contains content from branches that may or may not be included in the upcoming product release, and we don't want that content to accidentally end up in the release or master branches.
- One Person Merging to Master Everyone should be synchronizing local and remote branches, and everyone should be merging to and from the release branch (when appropriate). But *only one person* should merge from the release branch into the master branch when we are ready to build and publish the final output for a product release. See "Merging From Release to the Master Branch" on page 58.
- Delete Old Branches Over time, there can be a lot of feature branches. Many of them become obsolete because they were used only for a particular release cycle. After a final product release and merge to the master, those old branches should be deleted. It is also a good idea to periodically delete the develop branch and create a new one to keep it fresh. See "Deleting Old Feature Branches" on page 79.

How It Works

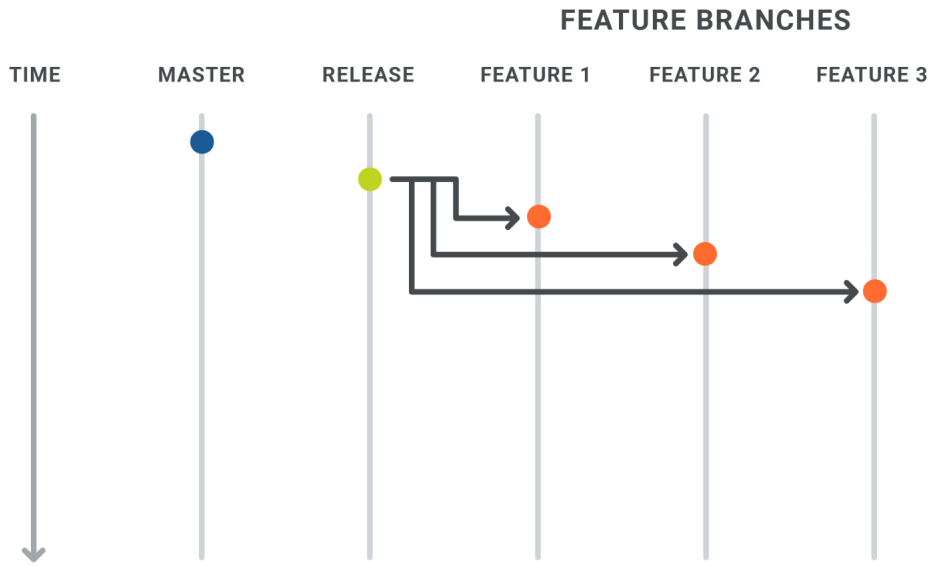
Following is a mostly sequential look at how the overall source control process works.

Create and Check Out Branches

When you want to document a new feature for a product, or make certain global changes that affect multiple products, you create a new feature branch that is based on the release branch (see "Creating Branches" on page 48).

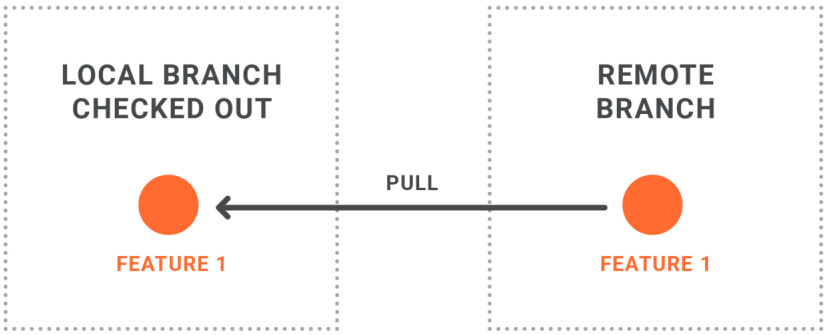
If you create a new local branch in Flare, be sure to publish the branch; this pushes the branch to Central. In the Source Control Explorer's **Branches** page, select **Unpublished Branch** (on the new branch created), and click the **Publish Branch** button.

If the branch already exists, select it or "switch" to it in Flare to check it out (see "Checking Out Branches" on page 32). Then, simply work in that branch, documenting your changes like you normally would.



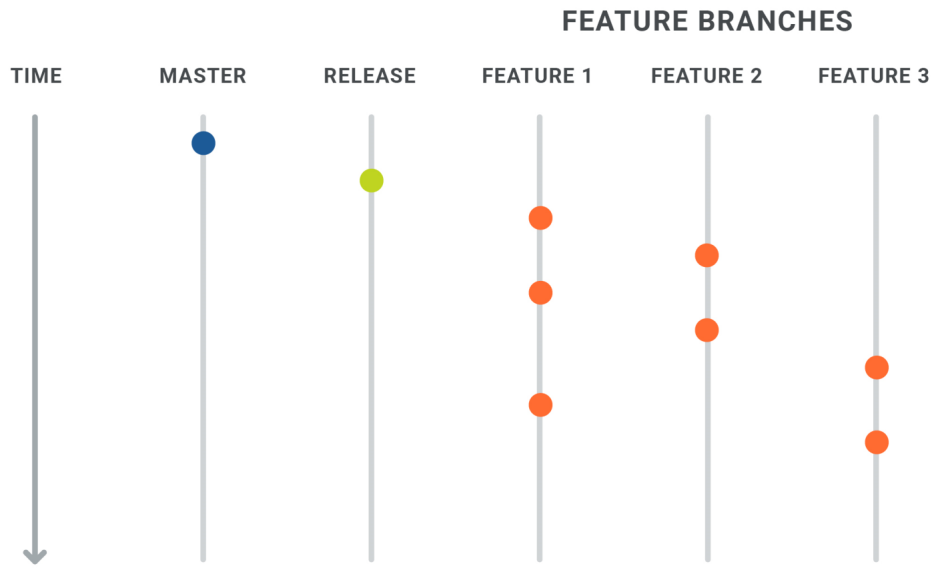
Pull From Remote Branch

Whenever you select an existing branch, you should immediately pull from the remote branch. See "Pulling Commits" on page 34.



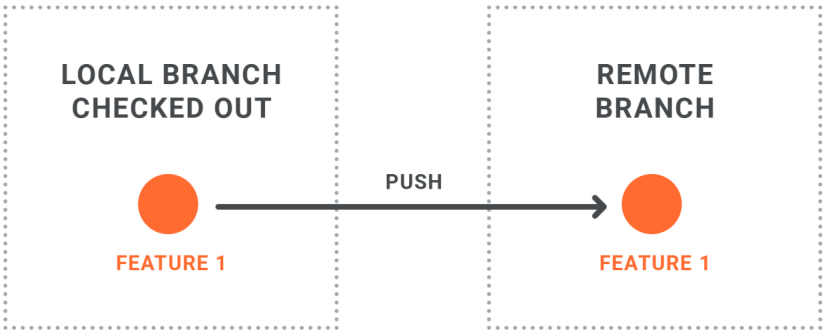
Commit

Anytime you commit your work in a branch (see "Committing and Pushing Changes" on page 43), a new record of those changes is recorded (represented by dots in the following graphic). A commit also occurs when you merge from branch to another.



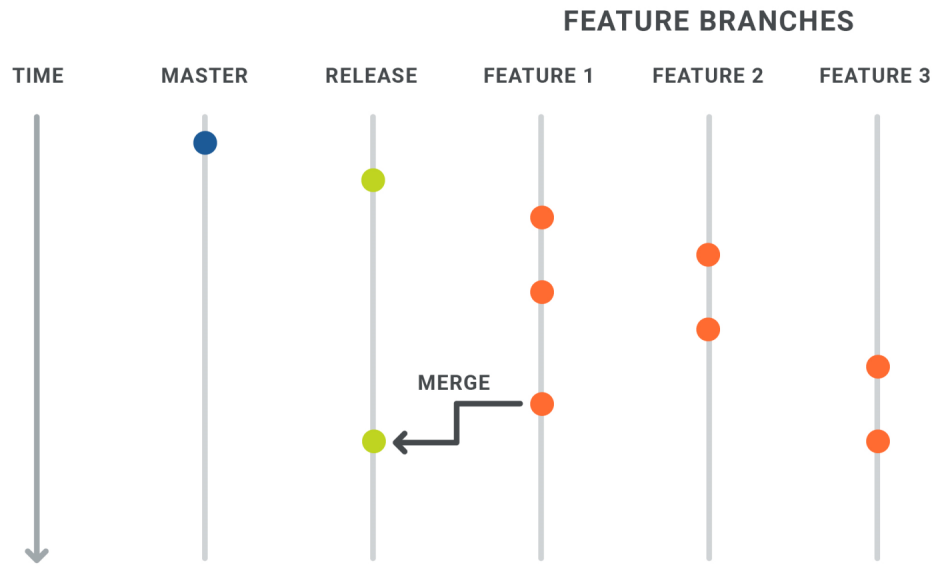
Push to Remote Branch

After you commit changes, you push those changes from the local to the remote branch. Again, if you use the Synchronize feature, it will automatically first do a pull before it pushes, which is always a good thing. See "Committing and Pushing Changes" on page 43.



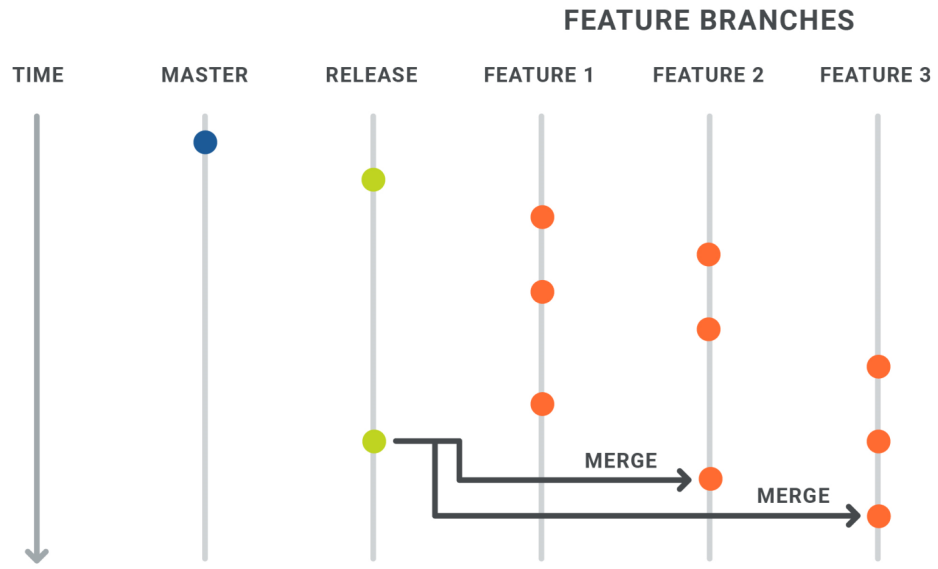
Merge From Feature to Release

When you are finished documenting in a feature branch (i.e., it is considered it ready for final publication), you merge that branch into the release branch. However, we always wait until the developers merge their code into their release branch first, because that gives us a higher degree of certainty that a particular feature will indeed be included in the next product release. See "Merging From a Feature Branch to the Release Branch" on page 53.



Merge From Release to Feature

Every day, you should merge from the release branch into your ongoing feature branches (see "Merging From Release to a Feature Branch" on page 35). This ensures that changes from *finished* feature branches are integrated into the *ongoing* feature branches.



Work With Develop Branch

There is one more special kind of branch called develop to use during a release cycle. The main purposes of this branch are:

- It lets us work with checklists in Central for topics that are in development.
- It lets us run reports on files that are in development and specific to a particular product.
- It lets us see how new files from various branches integrate with one another during development.

Basics of the Develop Branch

Even though we support branching, we do not yet support branching work for checklists or reports in Central. When it comes to these tasks, we want to have access to new and updated files as the release cycle progresses. We create child products for each product, naming them with the word "Develop" (e.g., Flare - Develop, Central - Develop, Lingo - Develop).

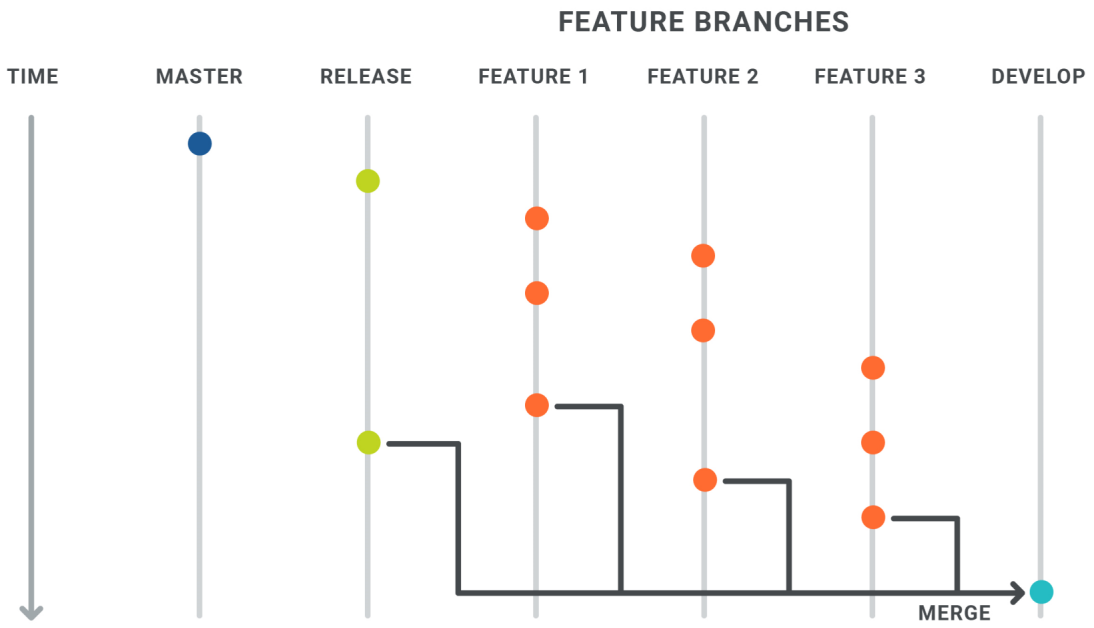
We check out the develop branch for the Shared project (see "Checking Out Branches" on page 32). Then we open a child "Develop" project and use Global Project Linking to import the files pertinent to that product (e.g., Flare, Central, Lingo). Within Flare, we do a commit and push the changes in the child project up to Central. From Central, we can then work on things such as checklists based on files in the develop branch.

Gitflow and the Develop Branch

So how does the develop branch fit in with the Gitflow system? When you need to update files in Central for one of the purposes previously mentioned (e.g., creating checklists), here is what you need to do…

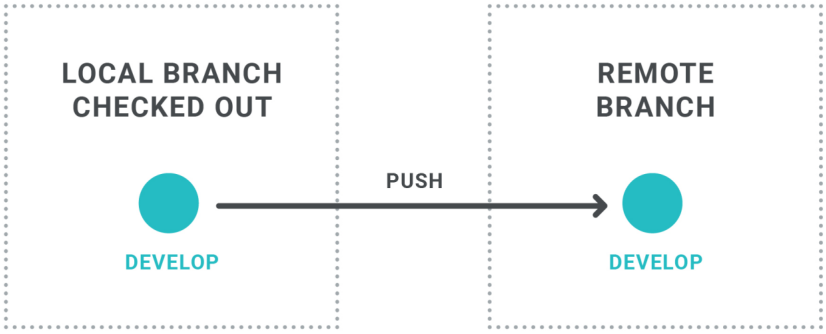
If the develop branch already exists remotely, but you don't have it locally yet, retrieve it. See "Getting Branches" on page 52.

To make sure you have all of the latest files, merge changes from the release branch, as well as from each feature branch you need, into the develop branch. See "Merging From Release to the Develop Branch" on page 38 and "Merging From a Feature Branch to the Develop Branch" on page 40.



Ideally, this is something that should be done daily during a release cycle. See "Daily Tasks" on page 26.

Next, push your changes from the *local* develop branch up to the *remote* develop branch (see "Committing and Pushing Changes" on page 43).



This way, others can pull those same files to their *local* develop branch.

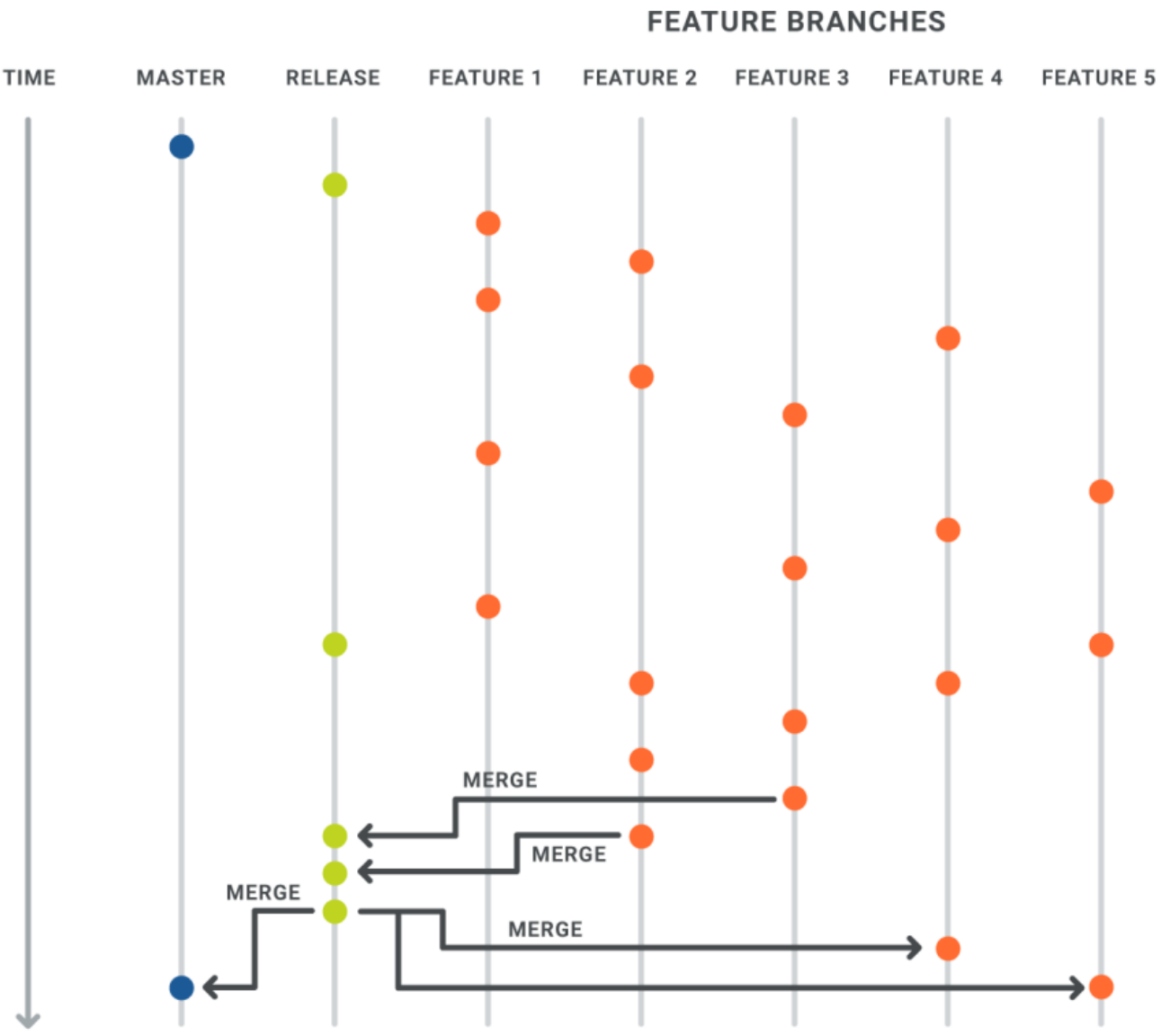
With the develop branch checked out for the Shared project, you can open one of the child "Develop" Flare projects and re-import Shared files into it. It is important to first make sure the develop branch is checked out, because the child project will import files from whatever branch you have opened. If you mistakenly have another branch checked out, you probably won't be importing all of the files that you actually need.

After you've imported the files into the child "Develop" project, push the changes up to Central (this push is done from the Central window pane within Flare). Then you can work with checklist(s), etc.

The develop branch is just a staging area holding changes from lots of different branches. It's not going to be used to do any work that will be part of the actual release, and it's certainly not used for the final output. It's really there to help you track the progress of files during a release cycle and perform other "in development" chores. This means that the develop branch can be deleted anytime you like. After all, it can always be easily re-created later when you need to use it again.

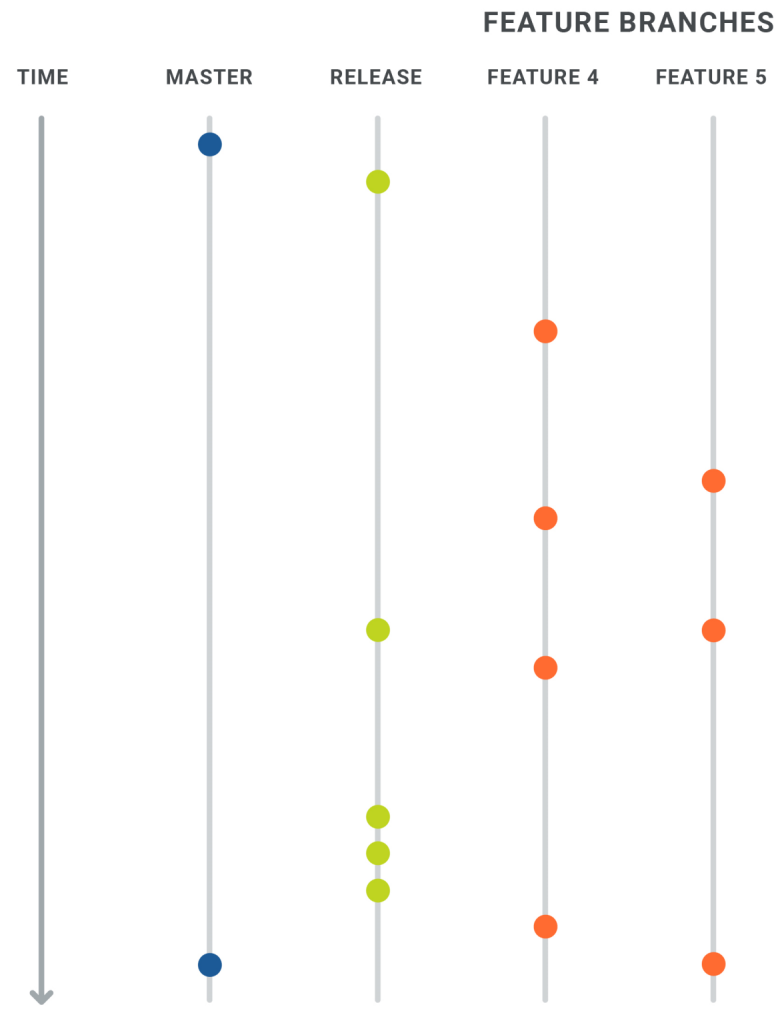
Merge From Release to Master

After all the feature branches that are part of the release cycle are completed and merged into the release branch, any final work (additional documentation, fixes, cleanup) can be done directly in the release branch until we are satisfied that everything is okay. At that point, one person merges the release branch into the master (see "Merging From Release to the Master Branch" on page 58). Also, the release branch is merged into other feature branches that are not part of this release cycle, but will be finished in the future (see "Merging From Release to a Feature Branch" on page 35).



Delete Old Branches

A few days after the product release, the feature branches no longer used are deleted, leaving only feature branches that are still in development. We usually coordinate this so that each writer is deleting the necessary branches both locally and remotely. See "Deleting Old Feature Branches" on page 79.



CHAPTER 3

Daily Tasks

If there is one section of this source control documentation that is important for you to internalize, this is it. Following are tasks that you should do every day, in order. If everyone on the team follows these steps consistently, we should be able to limit any conflicts or issues that might arise.

This chapter discusses the following:

Sync Release Branches

1. Open Flare and connect to the active license on Central that holds the Shared repository (see "Opening the Shared Repository" on page 30).
2. In Flare, select the *local* release branch. See "Viewing Branches" on page 31 and "Checking Out Branches" on page 32.
3. From the Source Control ribbon, select **Pull**. (If you already have the release branch open when you first open the project, and you have the feature enabled in Flare's Options dialog to automatically prompt you to get the latest files, you won't have to remember to do this manually each time.)
4. In the Select Remote for Pull of 'release' dialog, click **OK**. This pulls commits from the *remote* release branch into the *local* release branch. See "Pulling Commits" on page 34.

Sync Feature Branches and Merge From Release

1. In Flare, select a *local* feature branch. See "Checking Out Branches" on page 32.
2. From the Source Control ribbon, select **Pull**.
3. In the Select Remote for Pull dialog, click **OK**. This pulls commits from the *remote* feature branch into that *local* feature branch. See "Pulling Commits" on page 34.
4. From the Source Control ribbon, select **Merge**.
5. In the Select Branch to Merge dialog, select the release branch, and click **OK**. This merges the *local* release branch into the *local* feature branch. See "Merging From Release to a Feature Branch" on page 35.
6. Repeat these steps for each *local* feature branch that you have.

Sync Develop Branch and Merge From Release/Feature

1. In Flare, select the *local* develop branch. See "Checking Out Branches" on page 32.
2. From the Source Control ribbon, select **Pull**.
3. In the Select Remote for Pull dialog, click **OK**. This pulls commits from the *remote* develop branch into the *local* develop branch. See "Pulling Commits" on page 34.
4. From the Source Control ribbon, select **Merge**.
5. In the Select Branch to Merge dialog, select the release branch, and click **OK**. This merges the *local* release branch into the *local* develop branch. See "Merging From Release to the Develop Branch" on page 38.
6. Merge each of your *local* feature branches into the *local* develop branch. See "Merging From a Feature Branch to the Develop Branch" on page 40.

Work in Feature Branch

1. In Flare, select or switch to a *local* feature branch to work in it. "Checking Out Branches" on page 32.
2. Pull commits from the *remote* feature branch into that *local* feature branch. See "Pulling Commits" on page 34.
3. Work as usual in Flare, and commit your changes at logical breaks throughout your work day. Make sure to add a descriptive comment to your commits. Then push your commits to the *remote* feature branch. See "Committing and Pushing Changes" on page 43.

   You should not have any outgoing commits at the end of the day. Everything should be pushed to the remote branches.

IMPORTANT You should NEVER merge the develop branch into any other branches. The develop branch only receives commits from other branches; it should not send them out.

NOTE You might encounter certain issues, conflicts, and messages. For details on addressing the most common problems, see "Dealing With Issues, Conflicts, and Messages" on page 63.

Daily Task Step-by-Step Procedures

Following are step-by-step procedures that might be part of your daily tasks.

Opening the Shared Repository

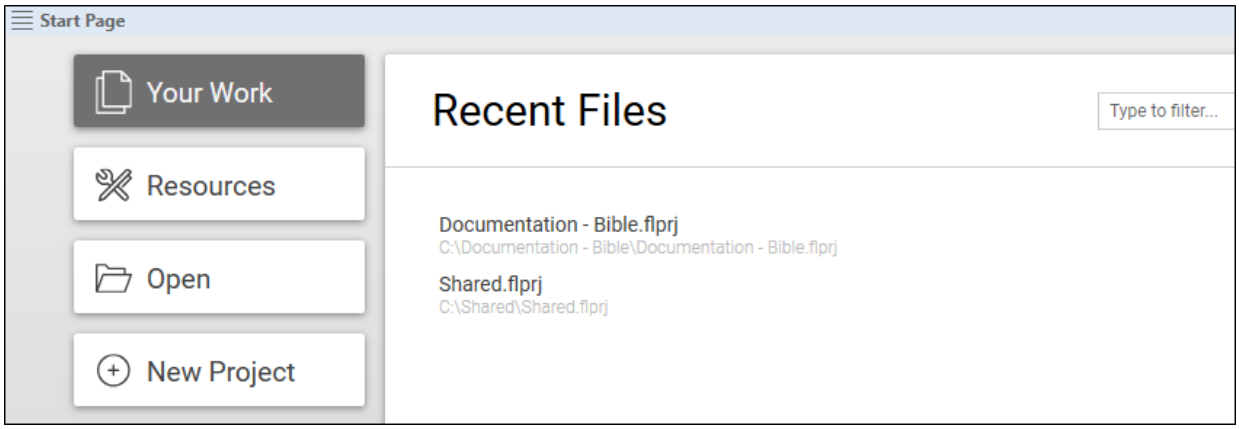
Before you perform any source control task, you must first open the Shared repository.

How to Open the Shared Repository

1. Open Flare.
2. Log into the Central license that hosts the Shared project.

> NOTE You can make changes in Flare without logging into Central, but if you plan on commiting, pushing, or pulling files, then it is necessary to log in. For our purposes, log into Central so we can do a pull first on any branches we intend to work in. That way the project has a better chance for remaining in sync.

3. In Flare, open the Shared project (Shared.flprj).

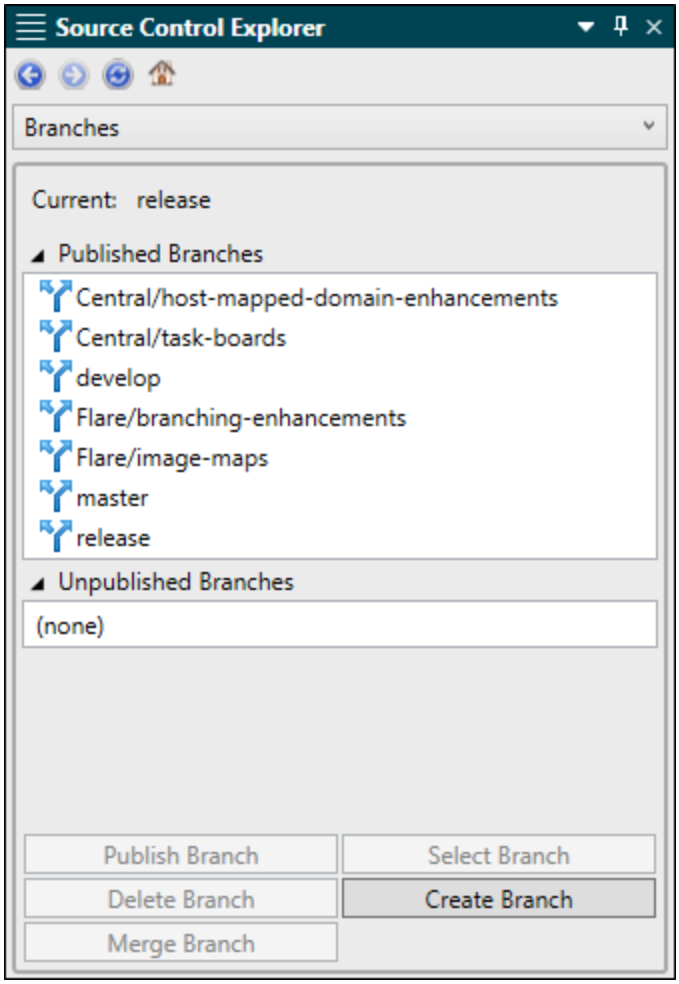


Viewing Branches

To see all of the branches in your repository and take different actions on them, open the Branches page of the Source Control Explorer.

How to View Branches in the Source Control Explorer

1. From the Source Control Explorer Home page, select **Branches**.
2. In the **Branches** page you can view all published (pushed to remote Central) and unpublished (local only) branches.



Checking Out Branches

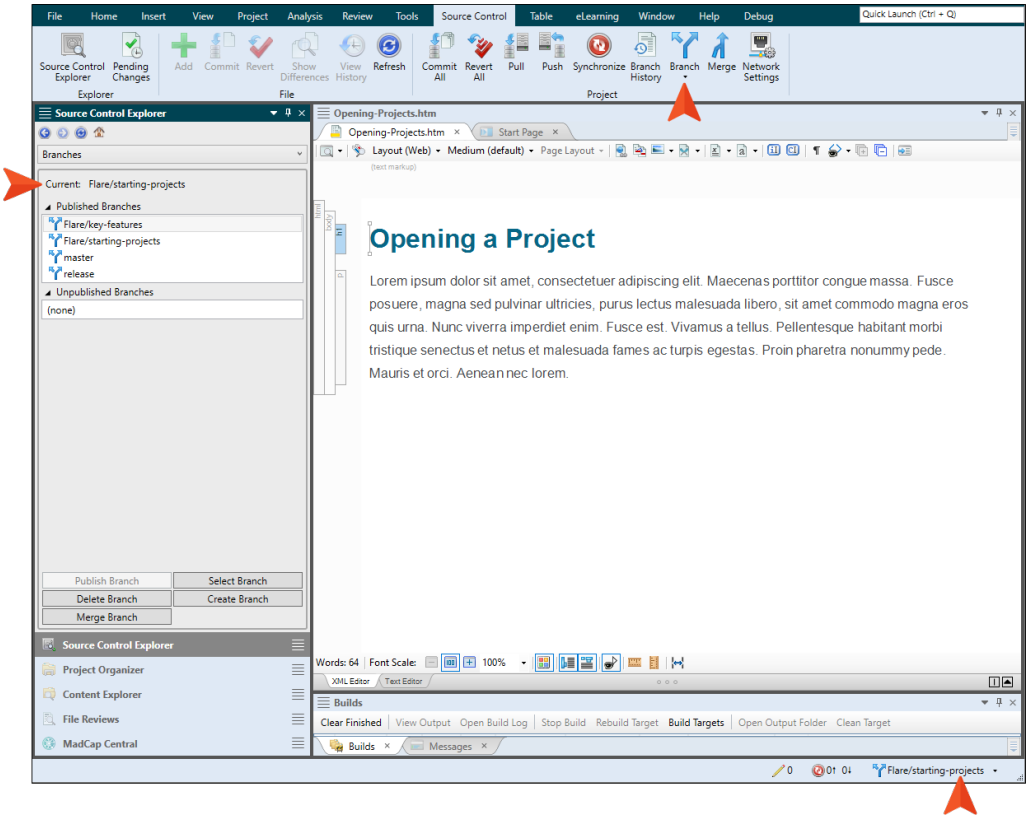
When you want to work in a branch, you need to first select it or "switch" to it. (This is also known as checking a branch out.)

How to Select a Branch

Do one of the following, depending on the part of the user interface you are using:

- In the lower-right area of the Status Bar, from the branches drop-down, select the branch.
- In the Source Control Explorer's Branches page, choose the branch you want to work with, and click **Select Branch**.
- From the Source Control ribbon, select **Branch**, and from the drop-down select the branch.

You will know that it is the active branch because in the Source Control Explorer is displays as the "Current" branch, and it also displays in the lower-right Status Bar at the bottom of the Flare interface.



If you have the Shared project open in Flare, you will see the files and edits pertaining to that branch. When you switch to a different branch, the Flare project will automatically change to show files and edits for that branch. In other words, you do not have multiple copies of the Flare project for each branch, but rather when you switch from branch to branch, only the files and changes that are current in the active branch are shown.

For example, you have a branch called “style-editor” and another called “fixed-headers.” If you have the style-editor branch selected, you will see the entire Shared project and all its files, but you will also see the files and edits that are unique to that branch. You won’t see any of the edits from the “fixed-headers” branch, and someone working in that branch won’t see your style-editor changes.

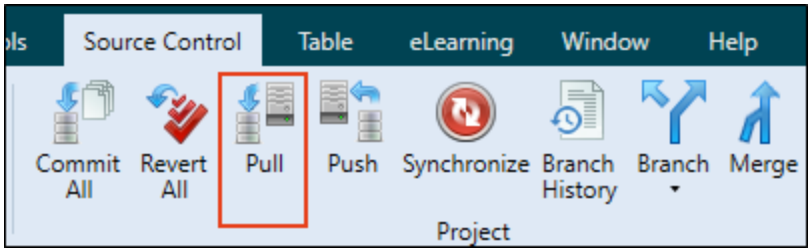
NOTE If you have made changes in a branch that is checked out, you must commit or discard the local changes before switching branches (or you will get an error message saying just that).

Pulling Commits

As a general rule, perform this step immediately after you select or switch to a branch.

How to Pull Commits

1. After you have selected a branch, you should see it listed as the current branch.
2. From the Source Control ribbon, click **Pull**.



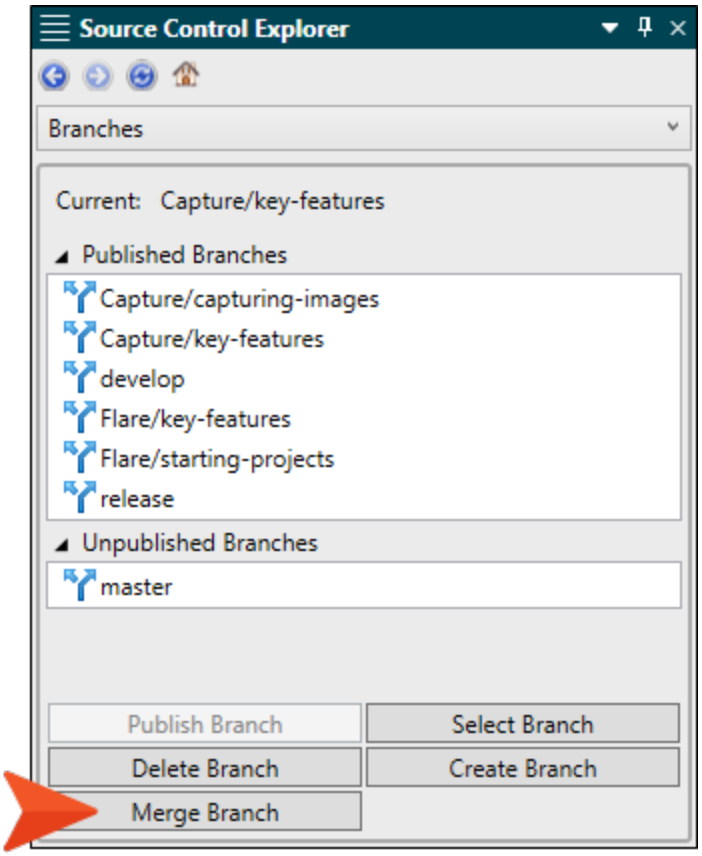
Merging From Release to a Feature Branch

Do this every day for each feature branch that you have locally. You should only have feature branches that you will be working on. Another writer will probably have different feature branches than you have. If you still have old branches from a previous release cycle, you should delete those so that you are not unnecessarily merging into them every day (see "Deleting Old Feature Branches" on page 79).

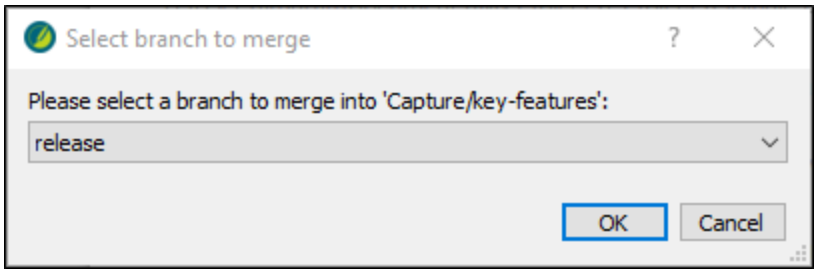
How to Merge From Release to a Feature Branch

1. In Flare's Source Control Explorer, open the **Branches** page. See "Viewing Branches" on page 31.
2. Choose the feature branch you want to work with, and click **Select Branch**. The name should be listed as the current branch in the **Branches** page and you should see it in the lower-right Status Bar of the interface. The idea is to update the current feature branch with the latest updates from the release branch.

3. From the Source Control Explorer, select **Merge Branch**. (Or, from the Source Control ribbon, select **Merge**.)

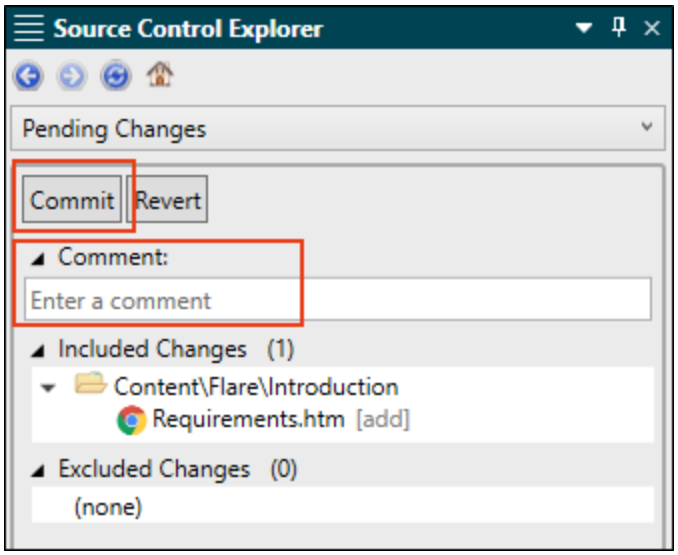


4. In the Select Branch to Merge dialog, select the release branch. Click **OK**.



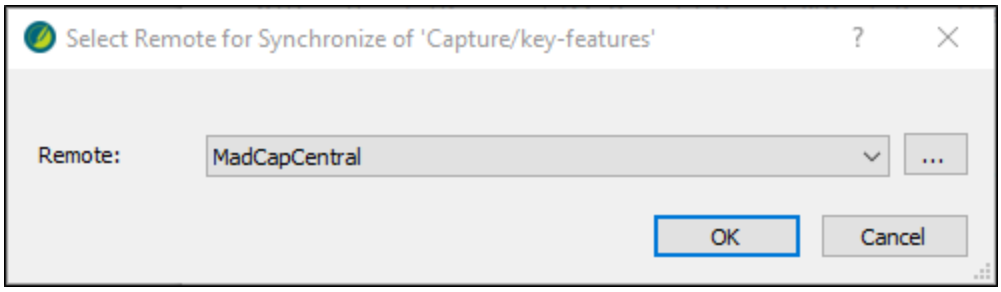
5. In the popup dialog, click **OK**. (This acknowledges a successful merge.)
6. The feature branch should already be selected in the interface Status Bar. A number next to the first button indicates how many files were changed and therefore need to be committed. Click the button.

7. The Source Control Explorer opens to the **Pending Changes** page. Enter a comment, and click **Commit**. You will see the file(s) from the release branch that need to be committed to the local branch back up to the remote branch.



8. The lower-right Status Bar displays the number of commits that need to be pushed. Click this button to synchronize.

9. In the dialog to synchronize, click **OK**.

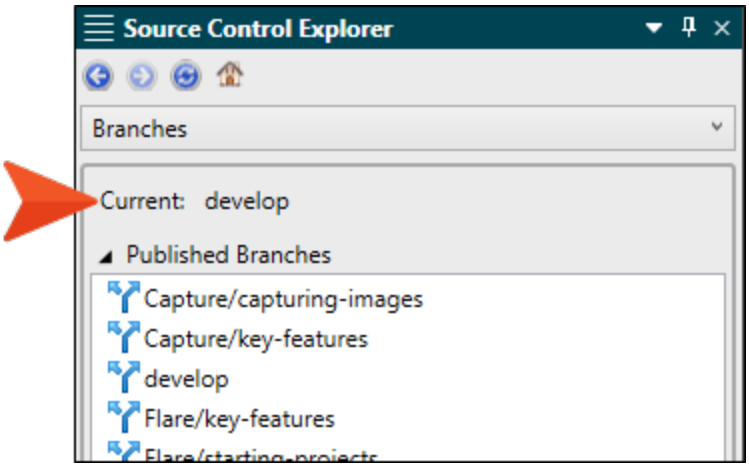


Merging From Release to the Develop Branch

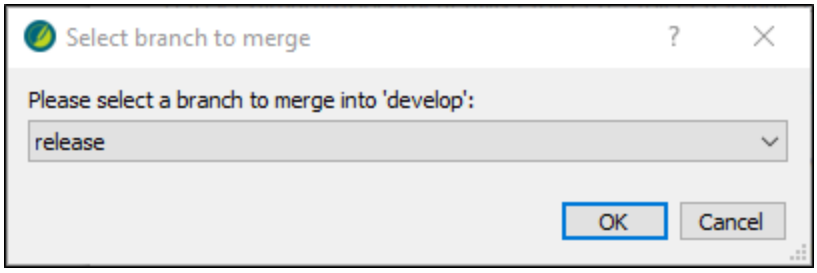
Do this every day for the local develop branch.

How to Merge From Release to the Develop Branch

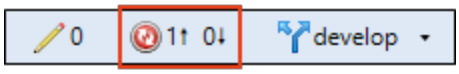
1. From the Source Control Explorer, open the **Branches** page. See "Viewing Branches" on page 31.
2. Choose the develop branch, and click **Select Branch** (if not already selected). The name should be listed as the current branch in the **Branches** page and you should see it in the lower-right Status Bar of the interface. The idea is to update the develop branch with the latest updates from the release branch.



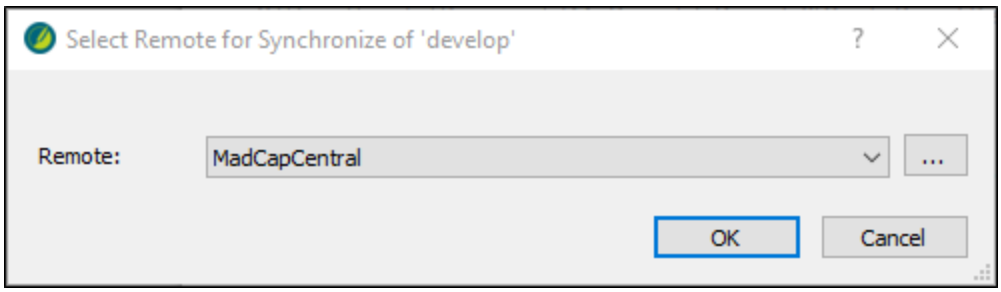
3. From the Source Control Explorer, select **Merge Branch** (or from the Source Control ribbon, select **Merge**.)
4. In the **Select Branch to Merge** dialog, select **release**.



5. Click **OK** (acknowledging a successful merge).
6. In the lower-right Status Bar, you will see a number next to the up arrow. That's because you have merged the release branch with the local copy of the develop branch, but now you have to push those changes from the *local* develop branch up to the *remote* develop branch.



7. In the Select Remote for Synchronize of 'develop' dialog, click **OK**.

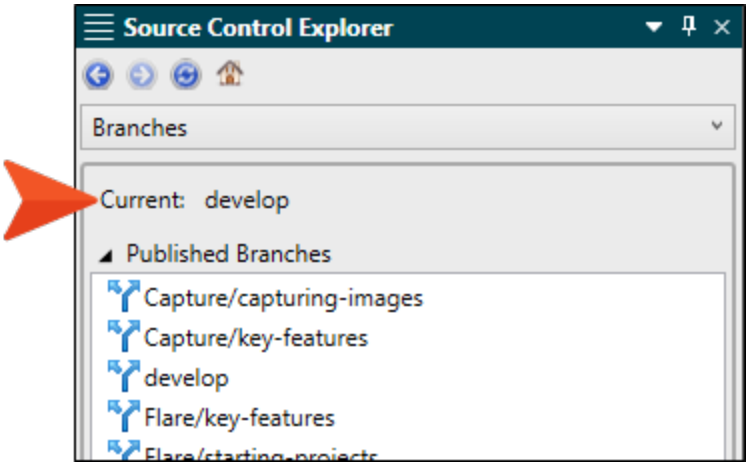


Merging From a Feature Branch to the Develop Branch

Every day you should merge each of your local feature branches into the local develop branch.

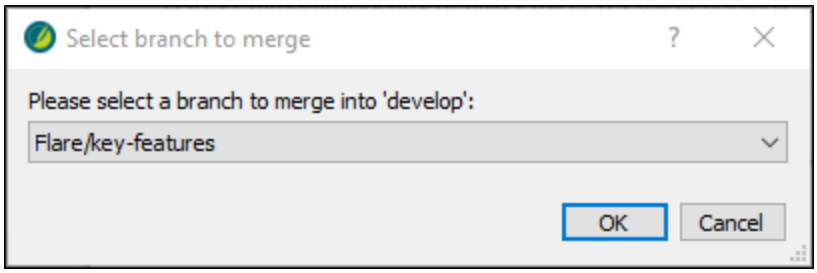
How to Merge From a Feature Branch to the Develop Branch

1. From the Source Control Explorer, open the **Branches** page. See "Viewing Branches" on page 31.
2. Select the develop branch by clicking **Select Branch** (if not already selected). The name should be listed as the current branch in the **Branches** page and you should see it in the lower-right Status Bar of the interface. The idea is to update the develop branch with the latest updates from each of your local feature branches.

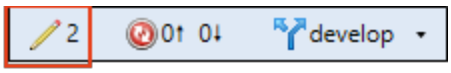


3. From the Source Control Explorer, select **Merge Branch** (or From the Source Control ribbon, select **Merge**).

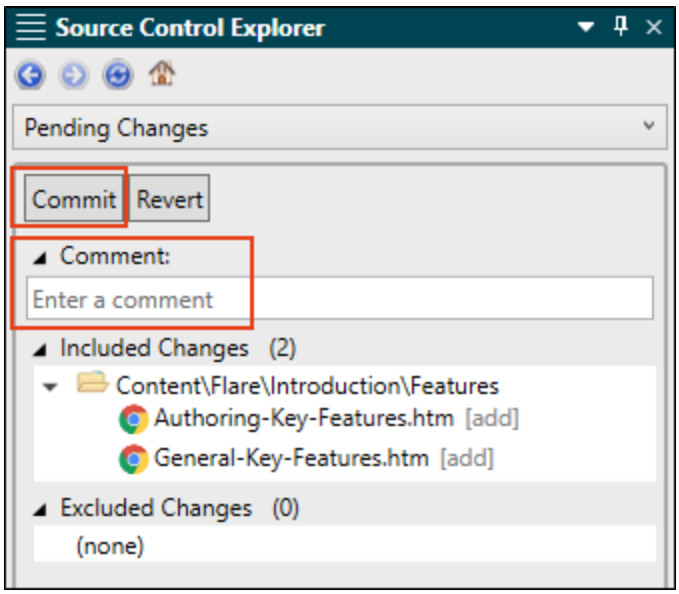
4. In the **Select Branch to Merge** dialog, select the local feature branch, and click **OK**.



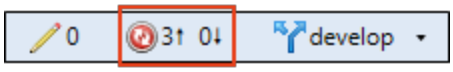
5. Click **OK** (acknowledging a successful merge).
6. The develop branch should already be selected in the lower-right Status Bar of the interface. The number indicates how many files were changed that need to be committed. Click the button.



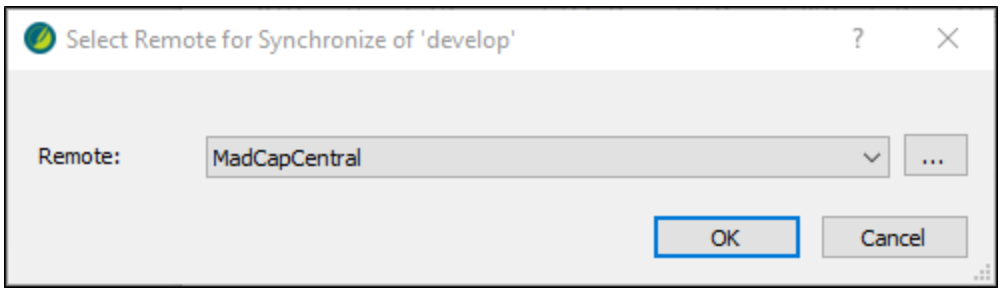
7. The Source Control Explorer opens to the **Pending Changes** page. Enter a comment, and click **Commit**. You will see the file(s) from the feature branch that need to be committed to the local develop branch.



8. The lower-right Status Bar displays all commits that need to be pushed. Click this button to synchronize.



9. In the Select Remote for Synchronize of 'develop' dialog, click OK.



The *remote* develop branch now has the latest changes for that feature branch.

Committing and Pushing Changes

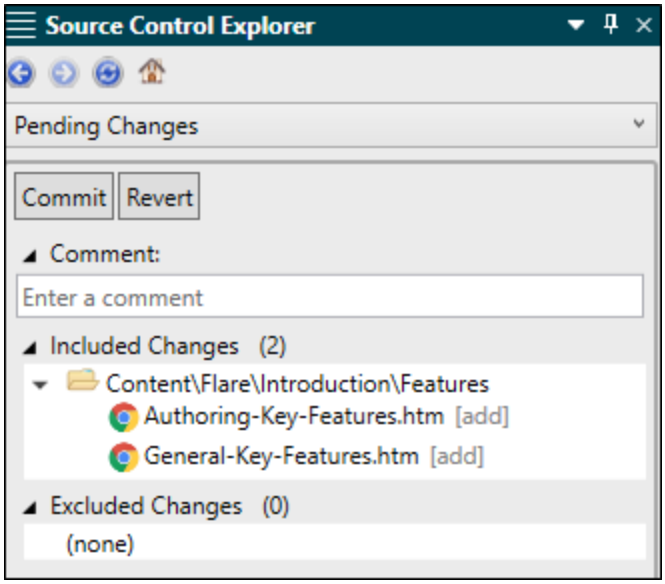
Committing changes means to tell Git that your edits are ready to be moved from the local branch up to the remote branch. You might work all day and not commit your changes until just before you go home (not recommended though). Other times, you might commit changes several times a day. Because each commit is accompanied by a comment that you write, you might decide to perform a commit whenever you come to a natural break in your work. By separating work into multiple commits, it can be easier to access or undo those changes later on if it becomes necessary, without disturbing the other changes.

EXAMPLE You might have documented three particular topics all morning and need to switch to working on the glossary. If you want the topic work to be included in a separate "container," you can commit those changes with a comment that identifies those three topics. Then you can work on the glossary for a while and do another commit, with a comment related to that work. And you might finish up your day by working on a couple of different topics, then committing those changes with a comment pertaining to the work you did.

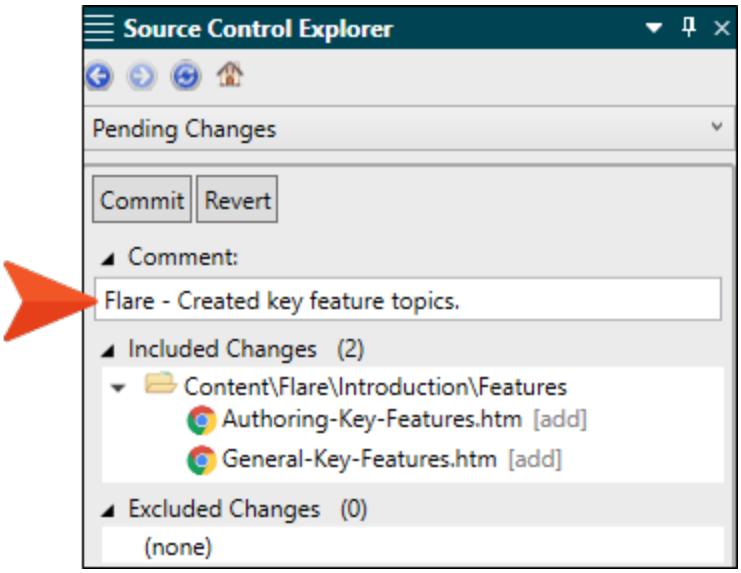
How to Commit and Push Changes

1. Select the local branch and work in Flare as you normally would. See "Checking Out Branches" on page 32.
2. When you want to commit changes, look at the lower-right of Flare's user interface. Next to the pencil icon, you will see a number indicating how many files with uncommitted changes you have in your project.

3. Click the button. The Source Control Explorer displays its **Pending Changes** page, showing the uncommitted files.



4. Enter a comment related to your changes. The comment should start with the name of the product (e.g., Central, D2H, Flare) or "Global" (if the changes affect multiple products), followed by a hyphen and then your comment.

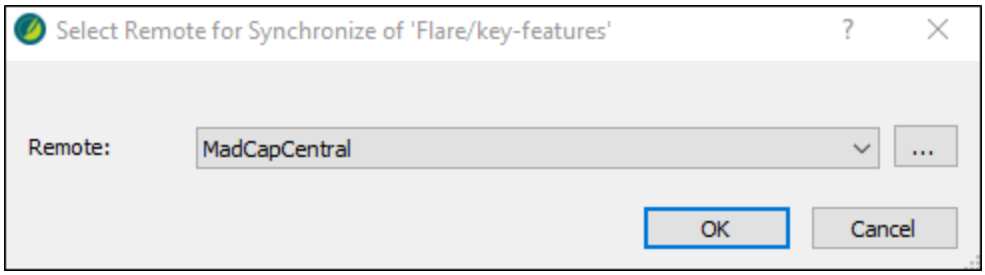


5. Click the **Commit** button.

   You should now see a number next to the up arrow at the bottom of the user interface. This indicates the number of commits that still need to be pushed up to the remote branch.

6. Click this button. This starts a synchronization process between remote Central and Flare. It pulls any remote commits and then pushes local commits to the remote.

7. In the dialog, click **OK**.



CHAPTER 4

Occasional Tasks

Following are tasks that should be performed only as needed:

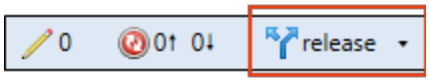
This chapter discusses the following:

Creating Branches

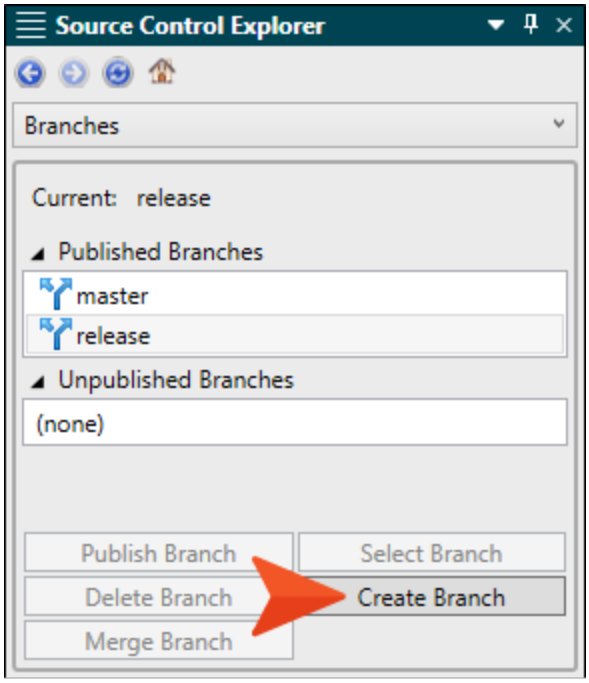
Create a new feature branch (based on the release branch) when there is a new feature for a product that you need to start documenting. You can also create new branches for general documentation changes (for a specific product or globally for multiple products).

How to Create a Branch

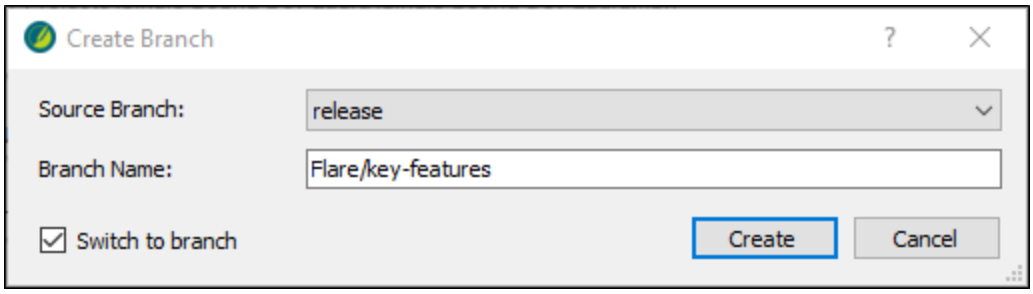
1. In Flare, open the Source Control Explorer.
2. Ensure the release branch is selected by looking at the toolbar in the lower-right corner of the Flare interface. If release does not display, click the drop-down and switch to it.



3. In the Source Control Explorer, select the **Branches** page.
4. Click the **Create Branch** button.

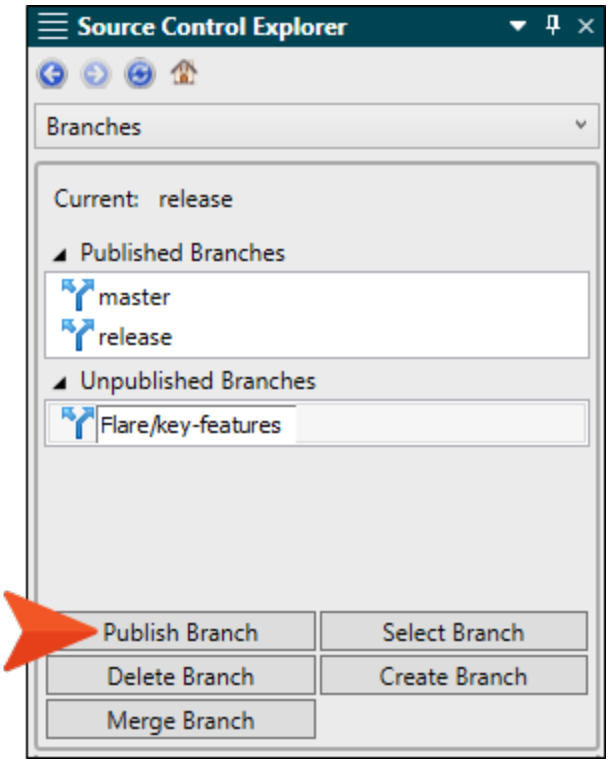


5. In the Create Branch dialog, enter a name for the new branch. When doing this, adhere to the following standards:
    - Start with the name of the product (e.g., Central, D2H, Flare) or the word “Global” if the new branch applies to multiple products. Follow this with a forward slash and then the name of the new branch.
    - Use lowercase for the names of branches.
    - For branches with multiple words, separate them with hyphens.

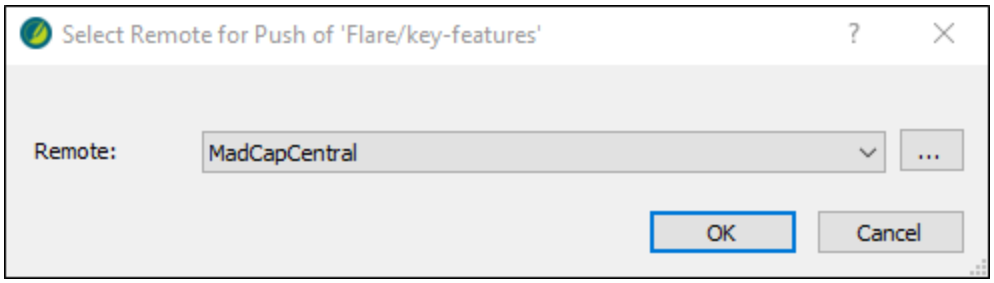


6. Select the Switch to branch check box. If this is selected the new branch will have focus in the Flare interface after you create it, otherwise the source branch (release) will remain active until you switch it manually.
7. Click Create.
8. In Branches page of the Source Control Explorer, the new branch displays. When a branch is “unpublished,” it exists locally. It still needs to be pushed so that it has a remote counterpart that is available to you and other writers.

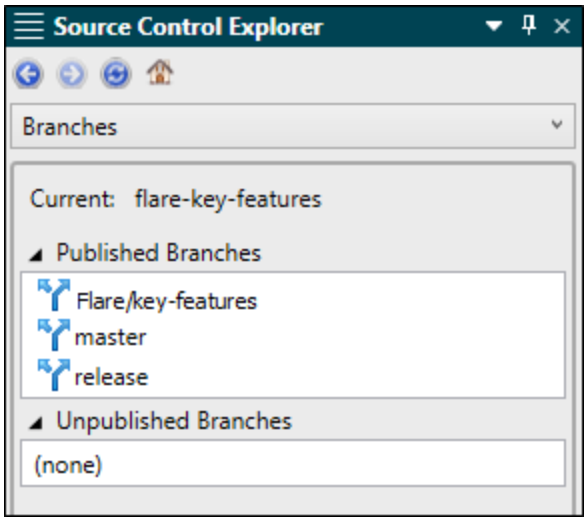
9. Select to highlight the unpublished branch you just created, and select the **Publish Branch** button.



10. In the Select Remote for Push dialog, click **OK**.



11. The new branch is now published and can be seen with the other remote branches in Central. This is what is looks like in Flare.



Getting Branches

Complete the following steps to retrieve branches that you know exist remotely but are not yet local.

How to Get a Branch

1. With any branch selected as the active one, do a pull. This makes all remote branches available for selection in the Flare interface, even if you haven't yet added a remote branch locally.
2. Open the Branch Management dialog in one of the following ways:
    - **Status Bar** In the lower-right of Flare, click the name of the active branch.

      NOTE If you do not see this option, make sure **View > Status Bar** is enabled.

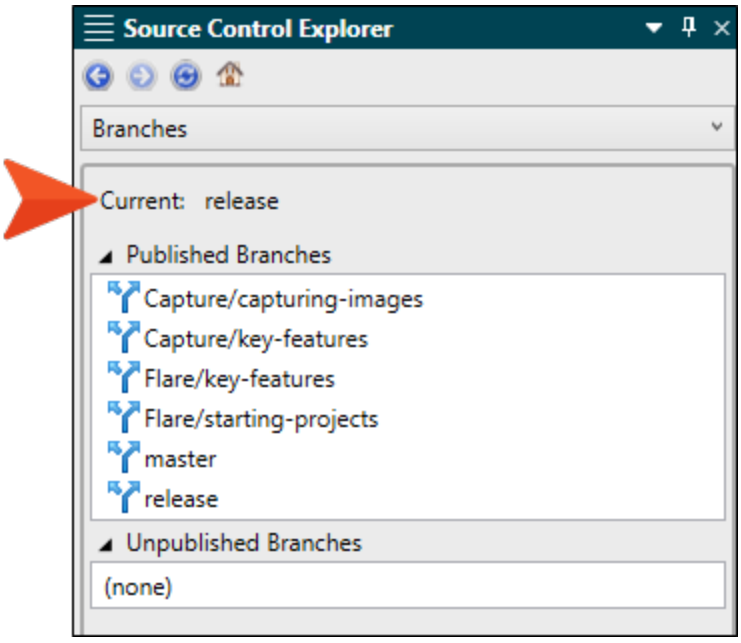
    - **Ribbon** Select **Source Control > Branch** (the face of the button, not the drop-down).
    - **Right-Click** If you have the Content Explorer, Project Organizer, Pending Changes window pane, or File List open, right-click any file and select **Source Control > Project > Branch**.
3. Select the **Remotes** tab.
4. Select the branch you want to get (i.e., check out) from the remote repository, and click **Switch**. This adds the branch to the Locals tab, and it also makes that new local branch the active one in the Flare interface.
5. Close the Branch Management dialog.

Merging From a Feature Branch to the Release Branch

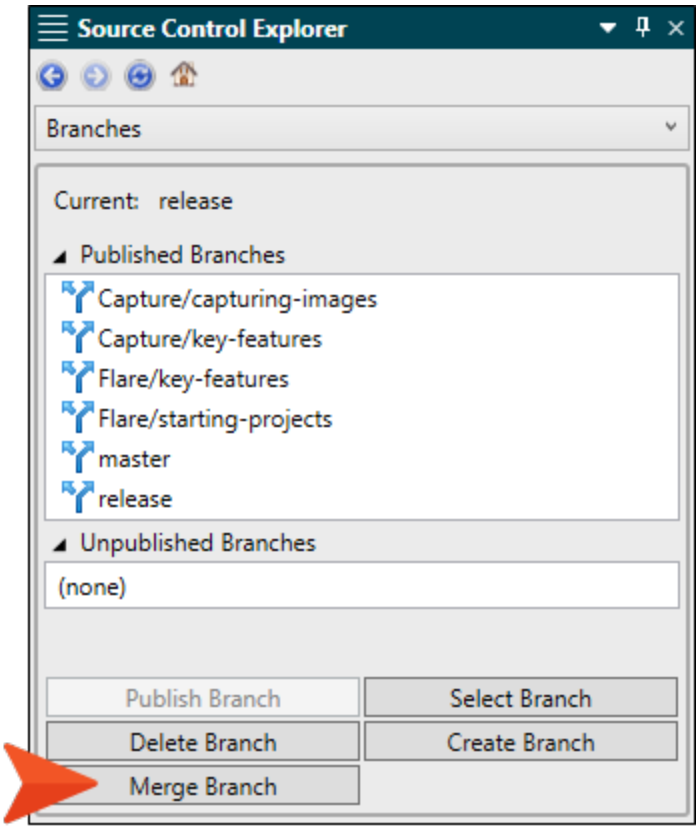
When you are all finished documenting in a branch for the next release, you are ready to merge that feature branch to the release branch. You do not want to do this too soon, because more documentation might become necessary as the product release date nears. The decision might be made to pull that feature out of the release, saving it for a later release. However, you don't want to wait too long to perform this step, because other branches may be dependent on the changes from your branch, or you might need the extra time to resolve any potential conflicts that might arise. When you are fairly confident in the completeness of the documentation and in its inclusion in the next release, you need to merge that branch to the release branch. Typically, we wait for the programmers to move the feature into *their* release branch before we move the documentation into *our* release branch.

How to Merge From a Feature Branch to the Release Branch

1. From the Source Control Explorer, open the **Branches** page. See "Viewing Branches" on page 31.
2. Choose the local release branch, and click **Select Branch** (if not already selected). The name should be listed as the current branch in the **Branches** page and you should see it in the lower-right Status Bar of the interface. The idea is to update the release branch with the latest updates from the feature branch.

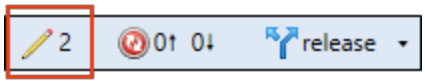


3. From the Source Control Explorer, select **Merge Branch** (or From the Source Control ribbon, select **Merge**).

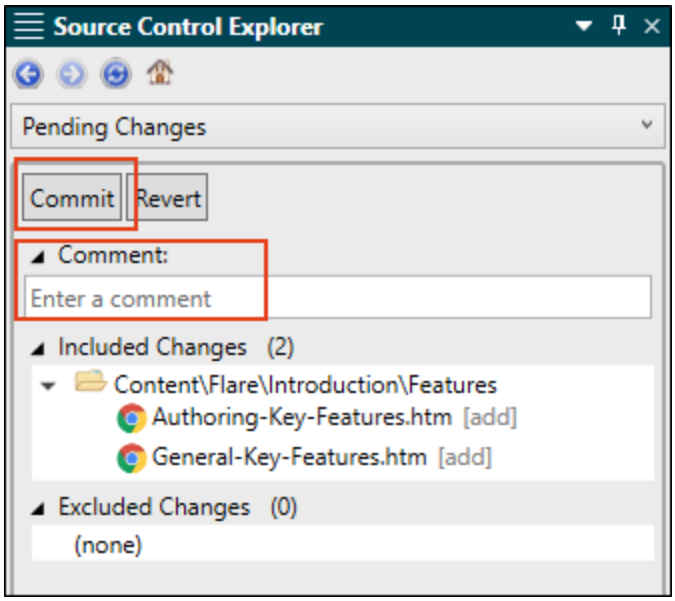


4. From the Select Branch to Merge dialog, select your feature branch.

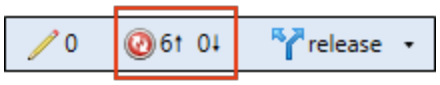
5. Click **OK** (acknowledging the merge is successful).
6. The release branch should already be selected in the lower-tight Status Bar. The number indicates how many files contain changes that need to be commited. Click the button.



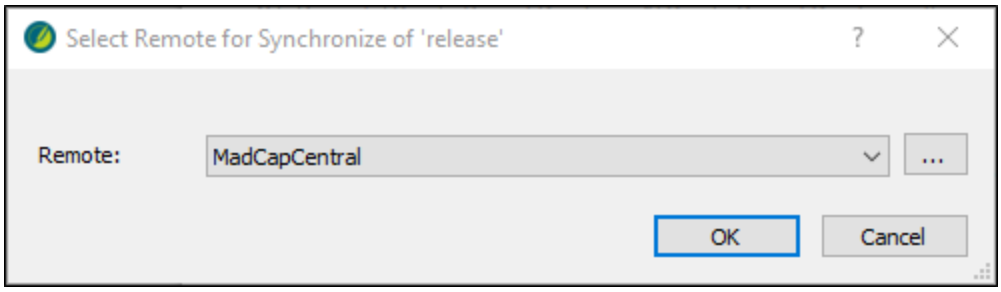
7. The Source Control Explorer opens to the **Pending Changes** page. Enter a comment, and click **Commit**. You will see the file(s) from the feature branch that need to be committed to the local release branch.



8. The lower-right Status Bar displays how many commits need to be pushed. Click this button to synchronize.



9. In the dialog to synchronize, click **OK**.



The *remote* release branch now has the latest changes for that feature branch. You should not need to open and work in that feature branch again, unless additional documentation is required. In that case, you would need to repeat these steps to merge to the release branch again.

> **NOTE** If you merge a feature branch to the release branch and then are informed the feature will not be included in the next version of the product, you will need to revert the branch. See "Reverting Commits" on page 74.

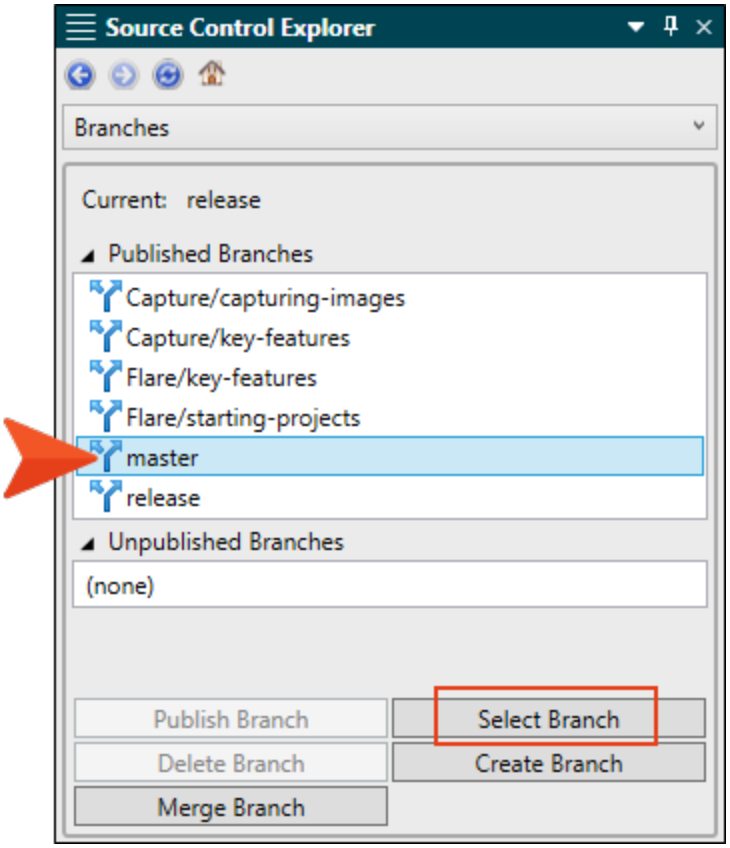
Merging From Release to the Master Branch

When all of the documentation is completed and ready for final publication, one person merges the release branch into the master branch.

Also, we use Git Bash to add a tag to the merge, which makes it much easier to find this action in the history log later; this can be useful if we need to create a new branch based on that point in time (e.g., for the purpose of doing a hot fix).

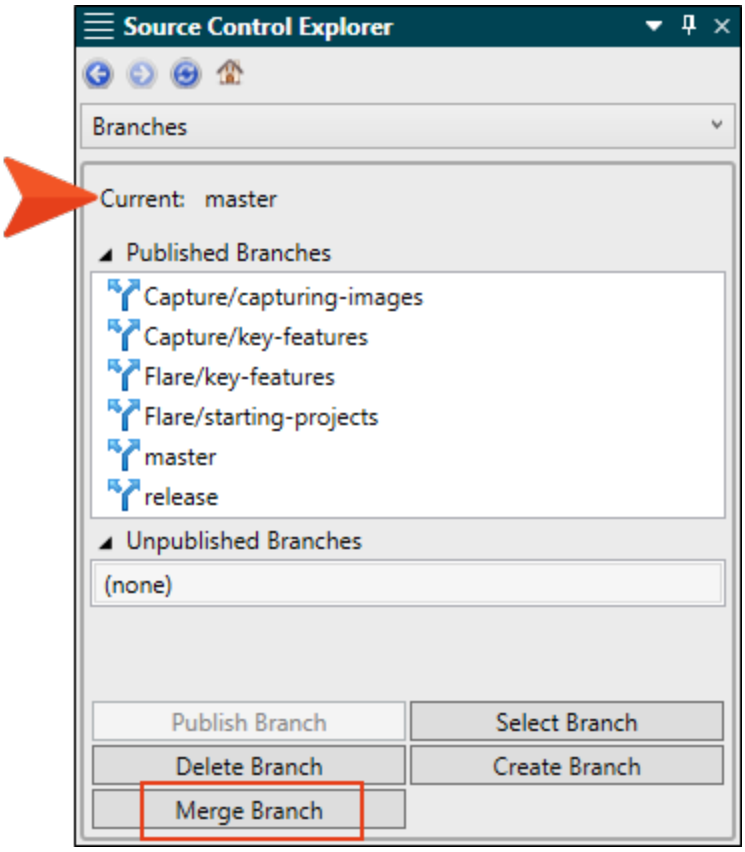
How to Merge From Release to the Master Branch

1. Ensure that all relevant feature branches have been merged into the release branch. See "Merging From a Feature Branch to the Release Branch" on page 53.
2. Ensure that the *local* release branch has been synchronized with the *remote* release branch.
    a. From the Source Control Explorer **Branches** page, select the release branch.
    b. From the Source Control ribbon, select **Synchronize**.
3. From the Source Control Explorer, choose the master branch, and click **Select Branch**.

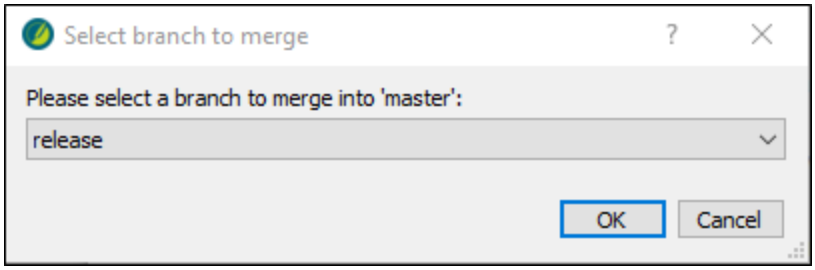


The name should be listed as the current branch in the **Branches** page and you should see it in the lower-right Status Bar of the interface. The idea is to update the master branch with the release branch (for final production).

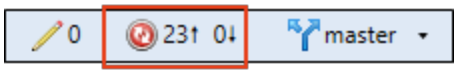
4. From the Source Control Explorer, click **Merge Branch**.



5. In the Select Branch to Merge dialog, select the local release branch, and click **OK**.



6. In the popup, click **OK** (acknowledging a successful merge).
7. The **master** branch should already be selected in the lower-right Status Bar. The number indicates a push is needed. Don't click the button yet.



8. Now we need to create a tag for this moment. So in Windows Explorer, right-click the **C:/Shared** folder and select **Git Bash Here**.
9. Type the following in the command line and press **ENTER**.

```
git tag [name of tag]
```

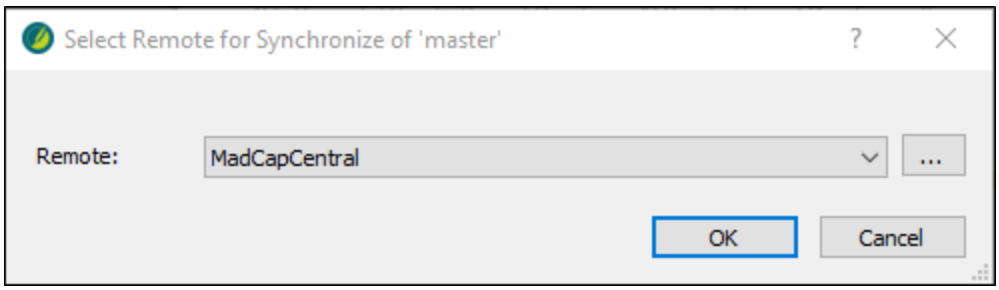
> ☆ **EXAMPLE** git tag doc-team

10. Type the following and press **ENTER**.

```
git push origin [name of tag]
```

> ☆ **EXAMPLE** git push origin doc-team

11. Click the number next to the arrow at the bottom of the interface. That's because you have merged the release branch with the local copy of the master branch, but now you have to push those changes from the *local* master branch up to the *remote* master branch.
12. In the Select Remote for Synchronize of 'master' dialog, click OK.



The remote master branch now has the latest changes. With the master branch checked out, you can open the project in Flare and build the final output(s). Also, you can look at the branch history in Visual Studio to locate the tag that you created.

Dealing With Issues, Conflicts, and Messages

Occasionally, you might encounter issues, conflicts, and messages. Start out by working things through with Flare. If necessary, you can use Visual Studio instead. And if that does not help, try using a command line tool to complete a process.

Command Line

There are a few things that might need to be done from a command prompt instead of the Visual Studio or Flare interface. You can use utilities such as Git CMD or Git Bash. If you do not already have one of these installed, you should do so.

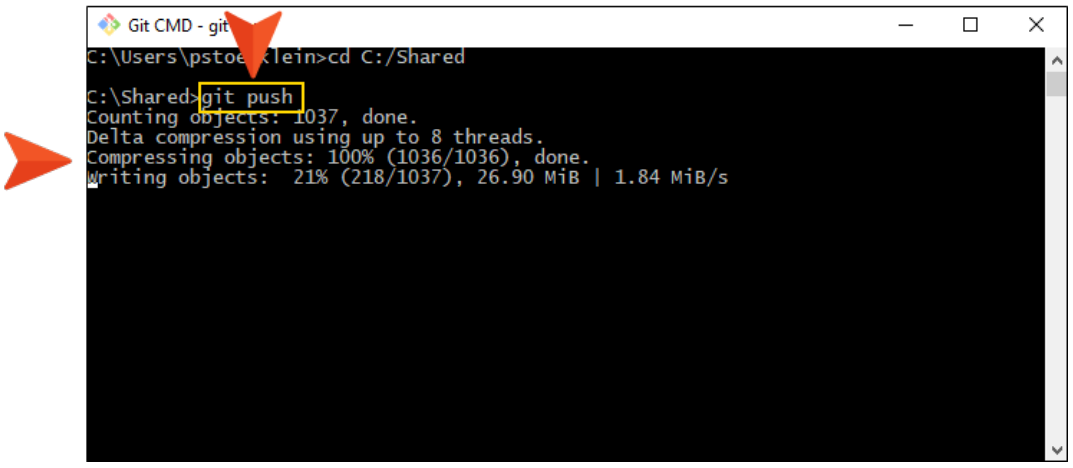
Failed to Push to Remote Repository

Sometimes Git is unable to push your commits to the remote repository. This can happen if the "remote end" gets hung up.

If you attempt to push commits, you might receive an error that indicates it failed to push. This error is more likely to happen if you have a large project with a lot of new images. As a workaround, try pushing your commits using the command prompt.

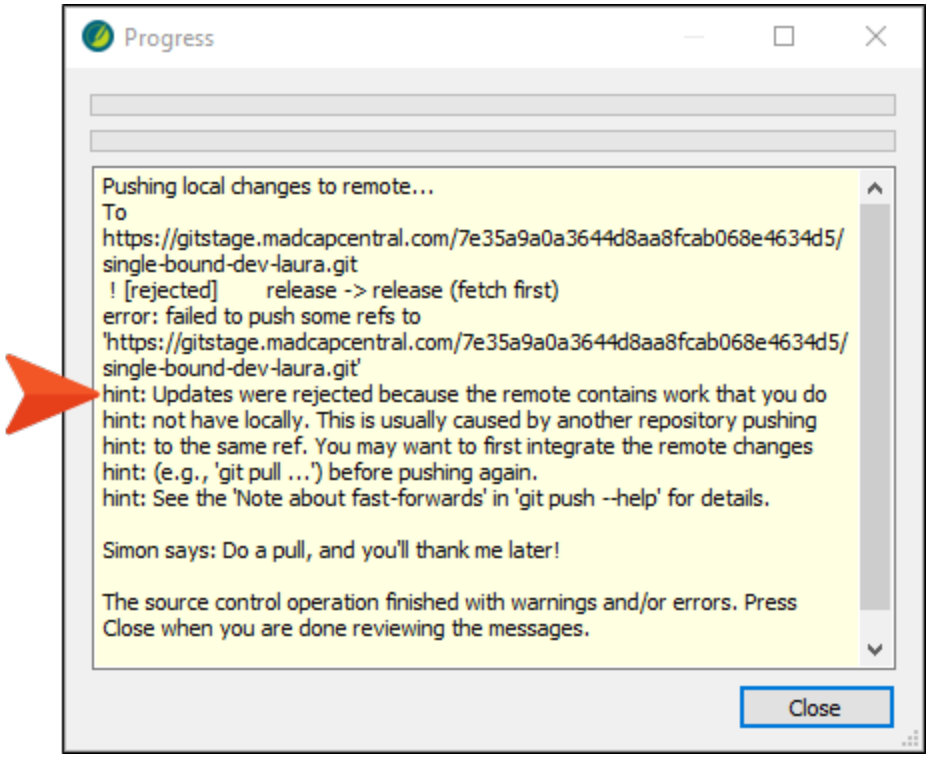
Do the following:

1. Make sure the branch is selected (or checked out). See "Checking Out Branches" on page 32.
2. In Windows Explorer, right-click the **C:/Shared** folder and select **Git Bash Here**.
3. In Git Bash, type `git push` and press **Enter**. You should immediately see some activity under the prompt.



Remote Contains Work You Do Not Have Locally

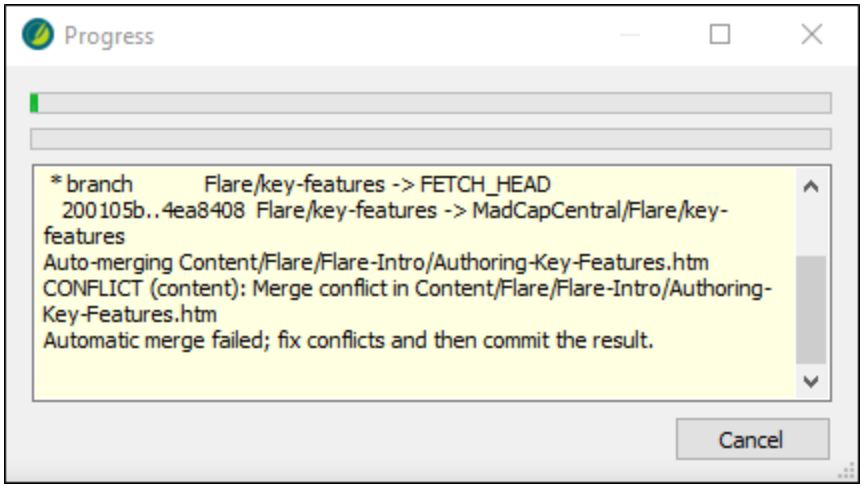
After you attempt to push commits, you might receive an error that indicates “Updates were rejected because the remote contains work that you do not have locally.”



This means that another author has pushed commits to the remote branch that you do not have yet. So you need to get those changes before pushing yours. If this occurs, synchronize the branch with the remote.

Conflicting Changes Detected

Sometimes you might attempt a push, and an error displays where you need to resolve file conflicts.

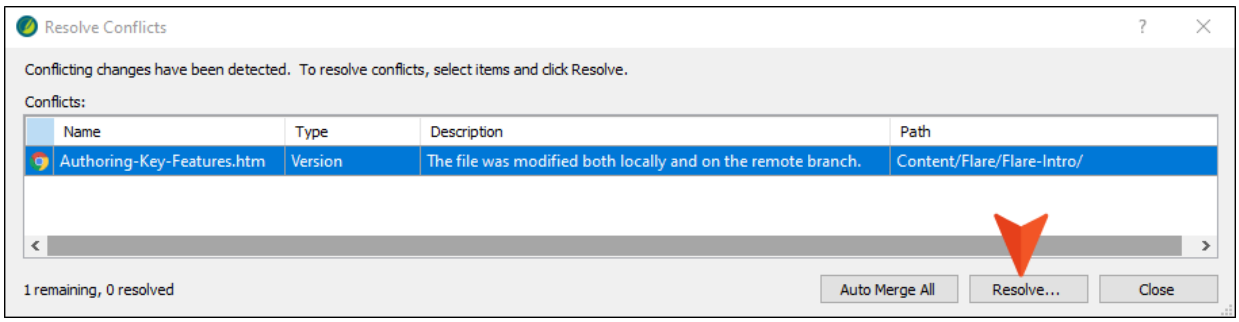


This message means the remote and local branches have differences in the same area of the same file. In other words, another author made changes to the same paragraph that you did. Flare doesn't know whose changes should be accepted, so you need to tell it.

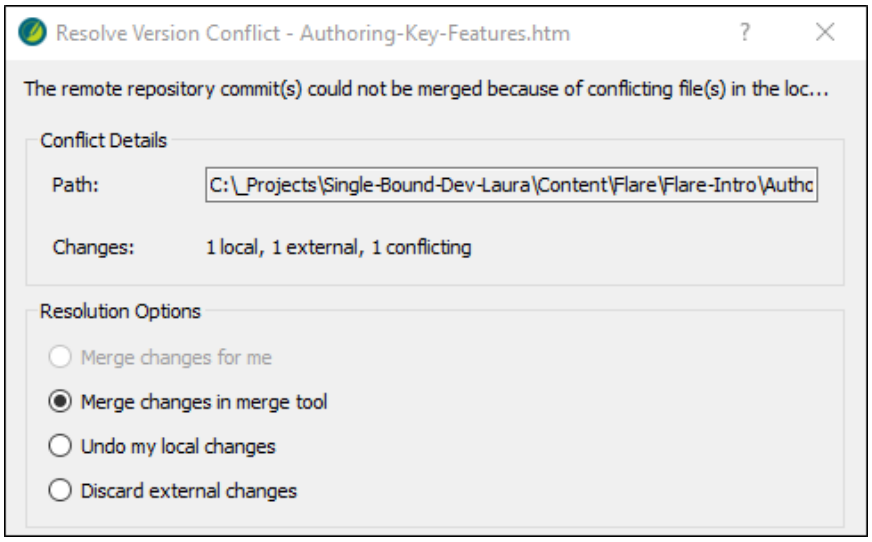
Avoiding conflicts is one reason we try to follow our daily tasks as consistently as possible (see "Daily Tasks" on page 26).

If you encounter file conflicts, do the following:

1. In the Resolve Conflicts dialog, select an item in the list, and click **Resolve**. (If you click **Auto Merge All** instead, it will attempt to auto merge changes together. If it fails to do so, you have to view the conflicts to resolve them.)

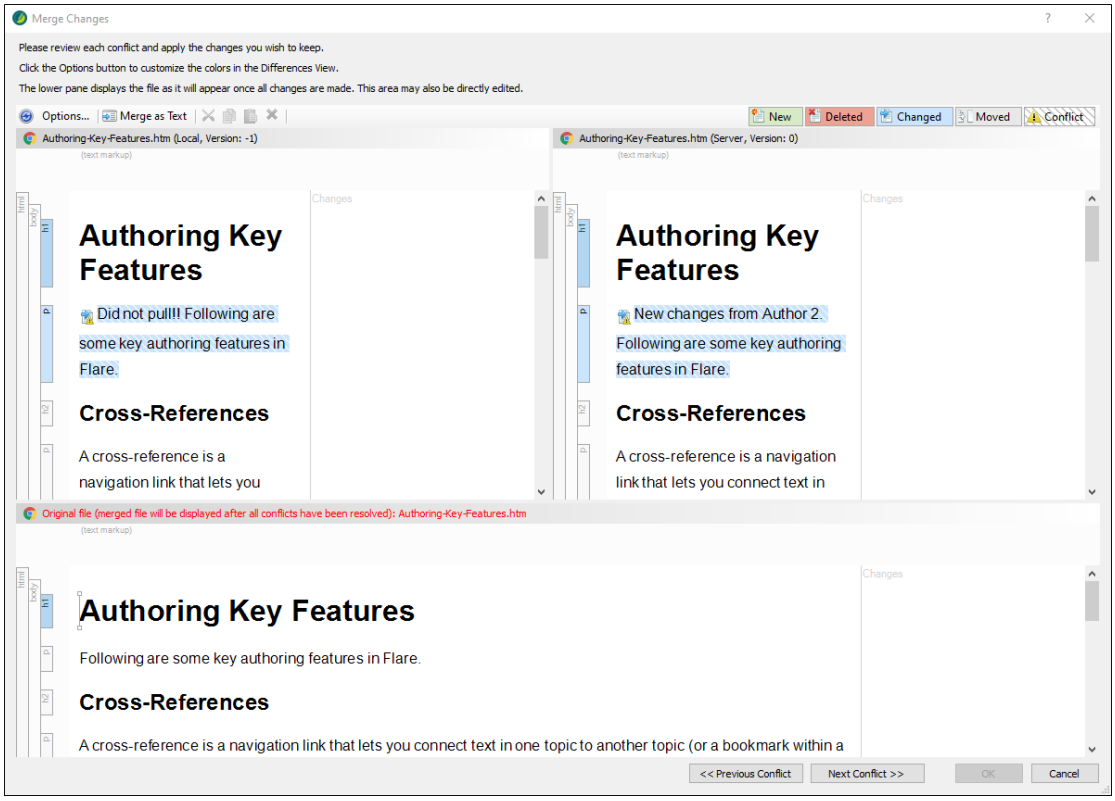


2. The Resolve Version Conflict dialog opens. You have several options.
   - **Merge changes for me** This option is similar to the Auto Merge All function, where Flare tries to merge all the file conflicts together.
   - **Merge changes in merge tool** Opens the Merges Changes dialog to compare conflicting files and resolve issues.
   - **Undo my local changes** Use this to undo your local updates.
   - **Discard external changes** If you are aware of the remote changes and know you want to disregard them, you can discard the external changes.

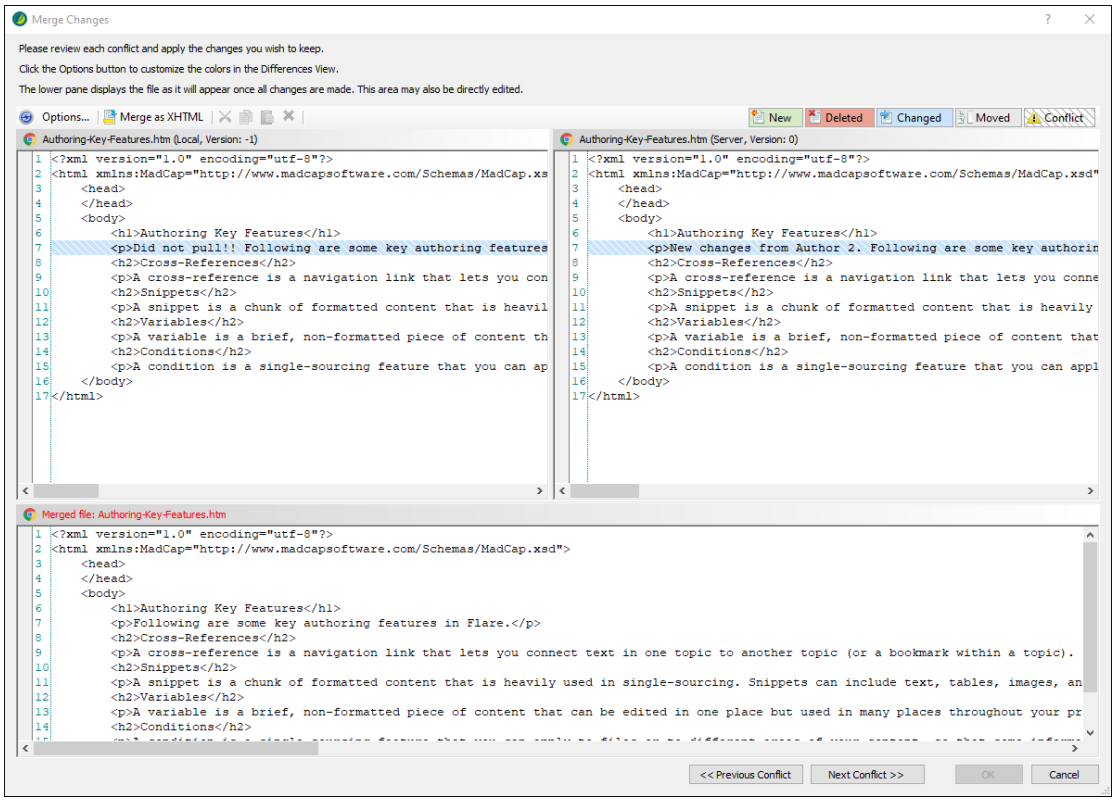


3. More than likely, you'll want to compare the files. In that case, select the **Merge Changes in Merge Tool** option, and click **OK**.
4. The Merges Changes dialog opens with three sections of markup for the file. The top left area shows the local changes, the top right area shows the remote version changes, and the area at the bottom shows you the original file without any of the local or remote updates showing. The bottom area also shows the altered file once the changes are made; or you can edit in that area directly.

   The following image shows content in the dialog as it would show in a WYSIWYG editor.

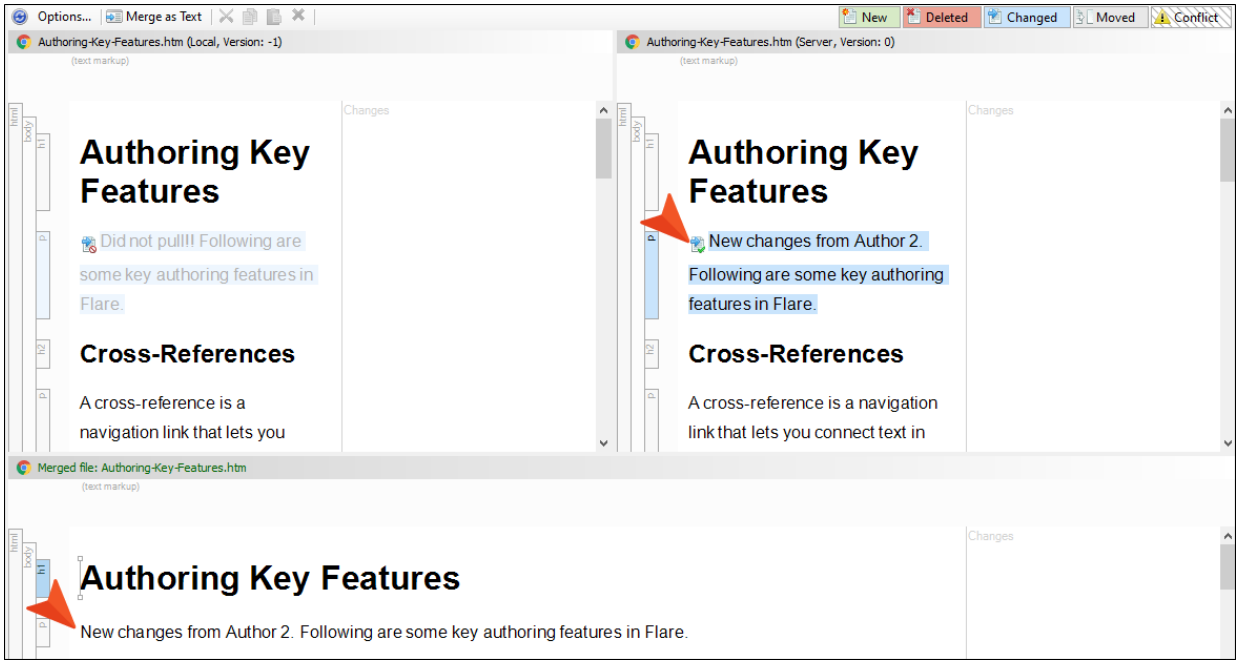


Alternatively, you can select the **Merge as Text** button to view the content as it would show in the Internal Text Editor. (The button then becomes **Merge as XHTML** to switch back to the other display.)

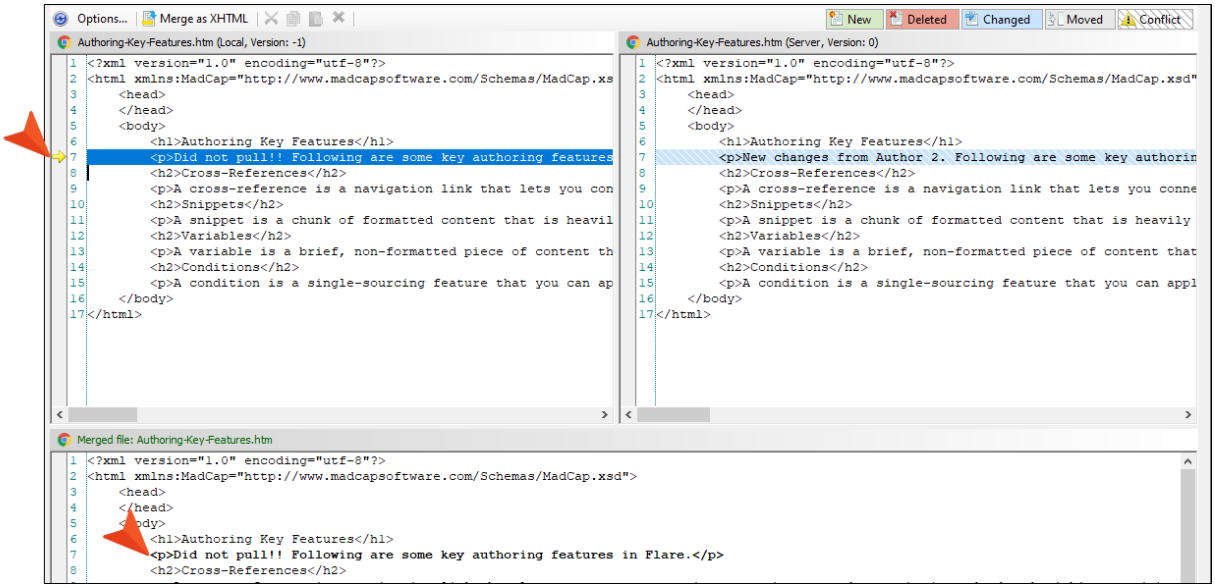


5. Look for the parts of the file where a shaded piece of content is displayed. These are the areas where you and other authors made conflicting changes. Select the change you want to accept.

In the WYSIWYG editor, click the conflicting text icon  for the content you want to keep. The icon will then display with a green checkmark. The bottom area changes from the original file to that with the selected changes merged into it.



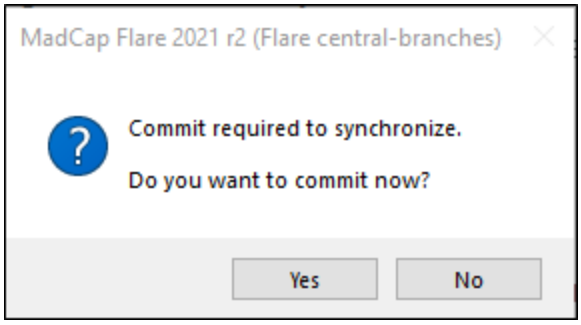
If viewing as the Internal Text Editor, click the line you want to keep. The bottom area changes from the original file to that with the selected changes merged into it.



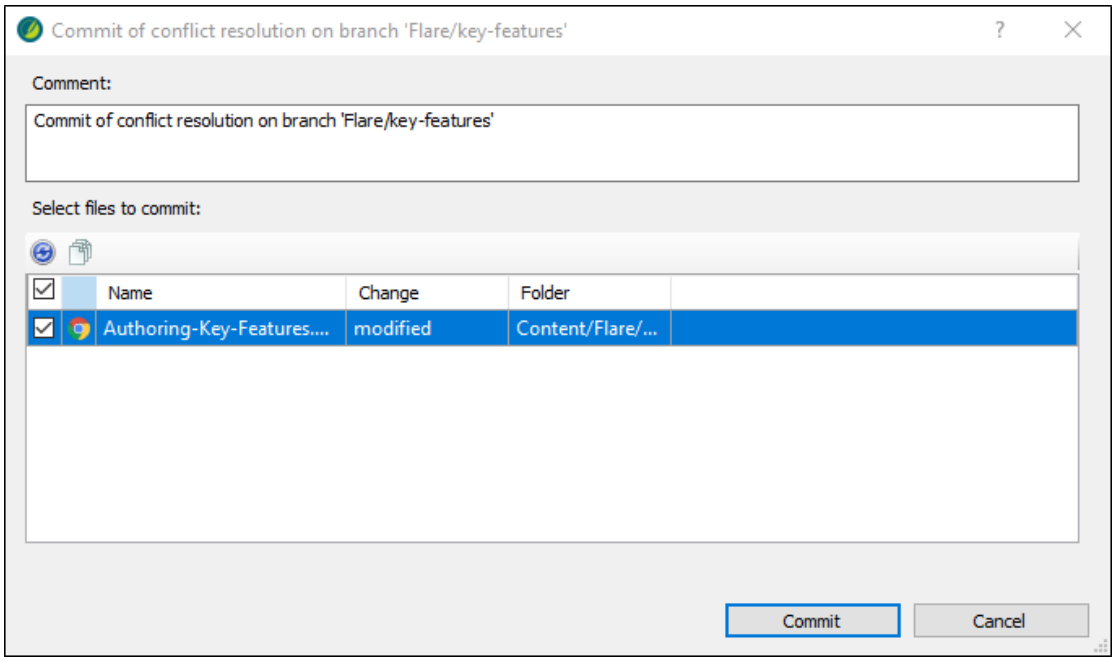
Do this for all the conflicts in the file. (You can sift through multiple conflicts by selecting the **Previous Conflict** or **Next Conflict** buttons in the UI.)

6. When all the conflicts are resolved, click the **OK** button to accept the changes.
7. In the Backup File Created dialog, click **OK**. You can select to not show this prompt again.
8. In the Conflicts Resolved dialog, click **OK**.
9. In the Synchronize dialog (no files pushed due to initial conflicts), click **OK**.
10. In the lower-right Status Bar, click the synchronize button.

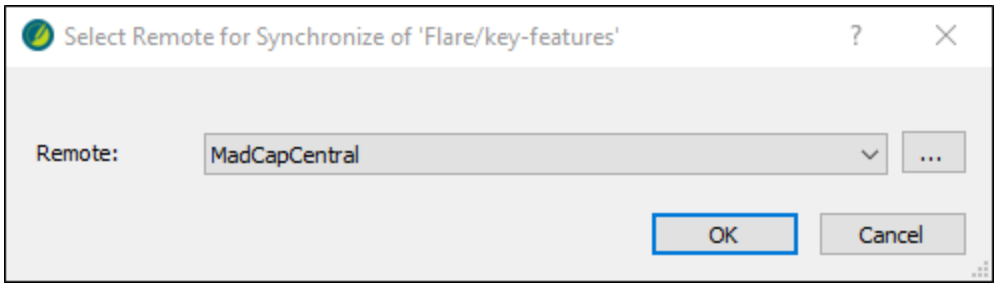
11. Click **Yes** to commit files.



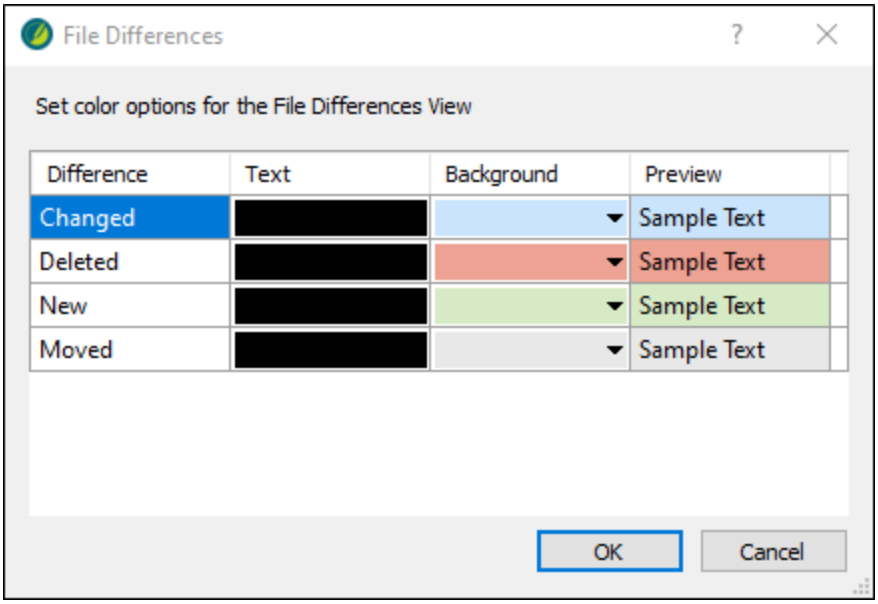
12. In the Commit of Conflict Resolution on Branch dialog, click **Commit**.



13. In the Select Remote for Synchronize dialog, click **OK**.



NOTE If you want to set the shaded colors for the file differences, select the Options button. This opens the File Differences dialog where you can change the colors.



Reverting Commits

Reverting a commit is usually only necessary if we merge something into the release branch but then are told the feature won't be included in that release of the product after all, or if we accidentally merge into the wrong branch. So we need to back those changes out of the branch.

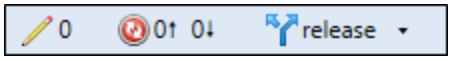
Git actually gives you the choice of resetting or reverting a commit:

- Reset The reset method is easier and cleaner, but Git recommends that you do a revert instead. A reset means you have to go around to other writers to track down whether any of them have the stray commits and then have each writer reset them.
- Revert A revert makes a new commit that undoes the previous commit. One benefit is that if other authors pulled from the branch before you backed out the commits, then those changes are out there with the other writers. A revert will undo those stray commits that other writers have out there when they pull again with the revert commit. A downside of the revert method is that it can mess up your history a bit.

Our first choice will be to do a revert when possible. If you need to do a reset, you'll have to use either Visual Studio or Git Bash.

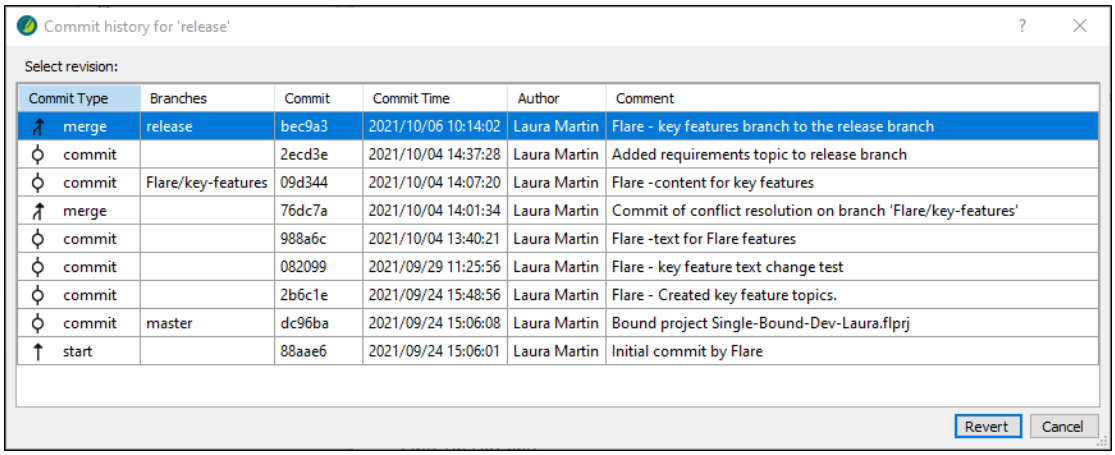
How to Revert a Commit

1. From the Source Control Explorer, open the Branches page. See "Viewing Branches" on page 31.
2. Select the release branch (if not already selected); or the branch containing the commit you want to revert. The name should be listed as the current branch in the Branches page and you should see it in the lower-right toolbar of the interface. We want to remove (or revert back) a committed feature branch from the release branch.

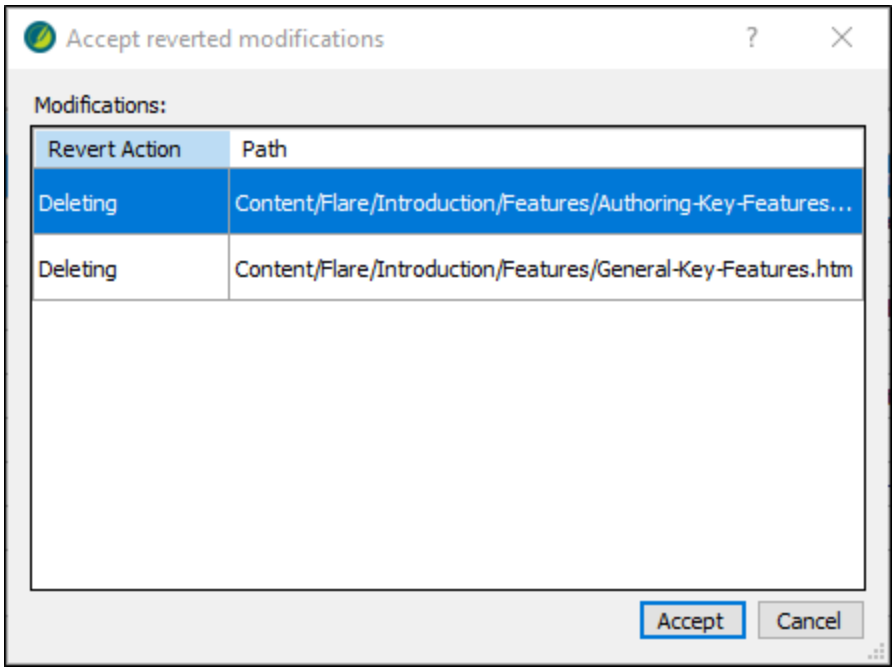


3. From the Source Control ribbon, select Branch History.

4. In the Commit history for 'release' dialog, select the commit to remove, and select **Revert**.



5. In the Accept reverted modifications dialog, ensure you want to revert the actions for that commit, and select **Accept**.



If you view the file(s) in that branch, notice the changes are no longer there. Any committed changes that were completed before or after that commit are still in the project.

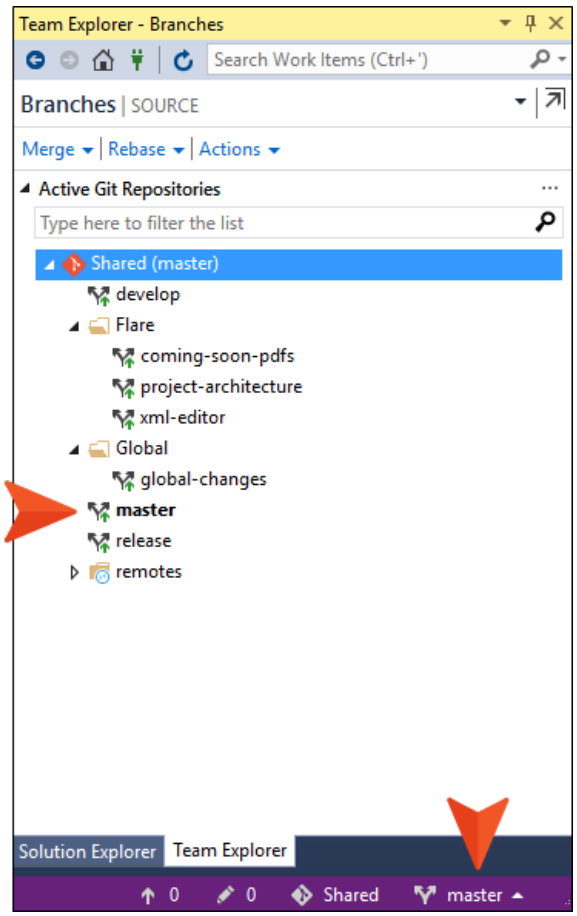
Using Hot Fixes to Update Old Outputs

If we need to fix published output for an older release, we can create a “hot fix” branch for it, based on the appropriate place in the master branch. Then make the changes in Flare and republish.

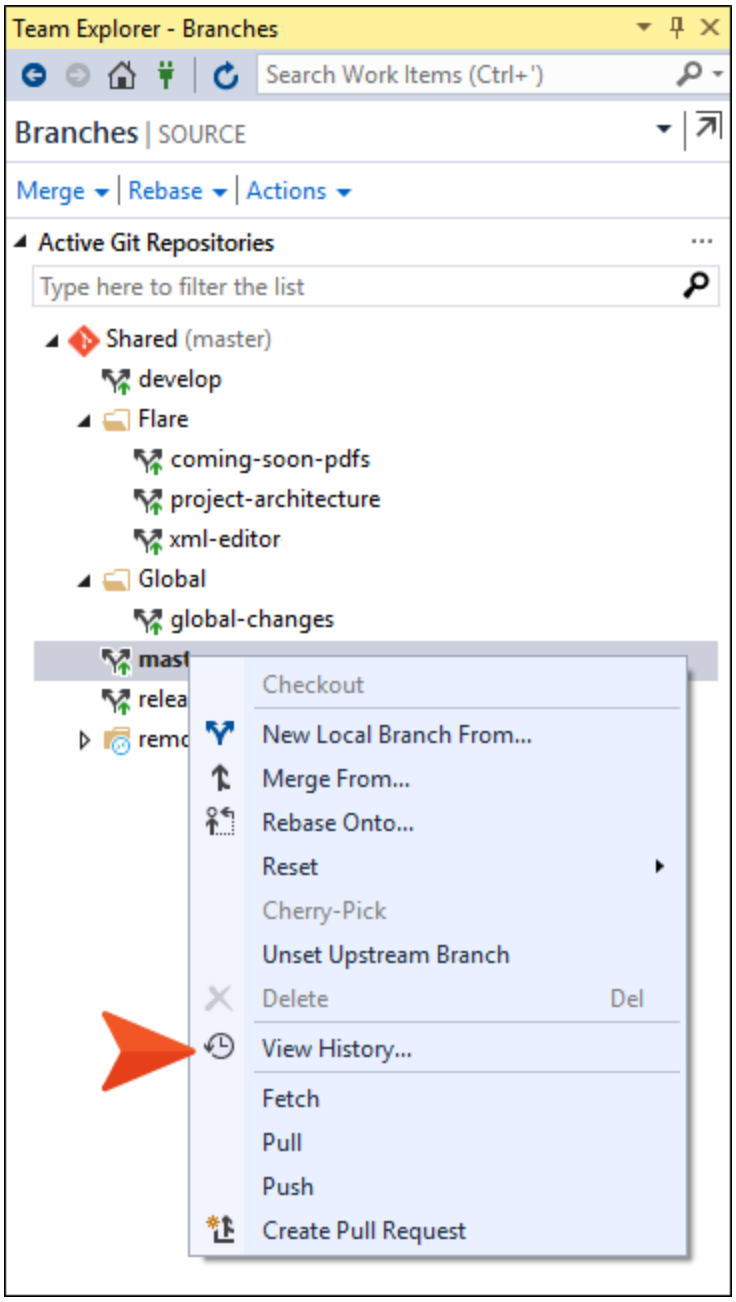
How to Use a Hot Fix to Update Old Outputs

1. In Visual Studio, open the Branches view in the Team Explorer. See "Viewing Branches" on page 31.
2. Double-click the *local* master branch to select it.

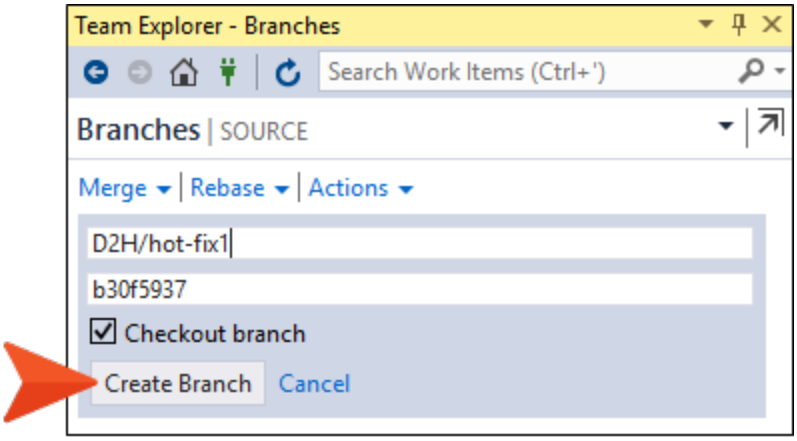
   After this, the name should be in bold and you should also see it at the bottom of the interface.



3. Right-click the master branch and select View History.



4. In the History window on the left, locate the most appropriate commit that you want to base the new "hot fix" branch on. In most cases, this commit will have a custom tag on it that we created when merging the release branch to master. Right-click the commit row and select New Branch.
5. In the area at the top of the Team Explorer on the right, enter a name for the new branch (e.g., D2H/hot-fix1). The commit ID is already entered in the field below this, indicating that it will be the basis for the new branch.
6. Click Create Branch.



7. The branch now exists locally, but still needs to be pushed so that it has a remote counterpart that is available to you and other writers. To do this, right-click the new branch and select Push Branch.

   That branch can now be seen with the other remote branches in the Team Explorer.
8. With the new "hot fix" branch checked out in the Team Explorer window, open the project and make the necessary changes.
9. When all changes from all writers are completed in that branch, make sure everything is synchronized (pulled, committed, pushed) for that branch.
10. Merge the "hot fix" branch into the master branch as well as the release branch.
11. Check out the master branch, then build and publish the updated output.
12. Once you are sure that the "hot fix" branch is no longer needed, you can delete both the local and remote branches. See "Deleting Old Feature Branches" on the next page.

Deleting Old Feature Branches

A couple of days after the final product release, delete feature branches that will no longer be used. We usually coordinate this so that each writer is deleting the necessary branches both locally and remotely.

It is also a good idea to periodically delete the develop branch and create a new one.

How to Delete a Branch

1. From the Source Control Explorer **Home** page, select **Branches**. See "Viewing Branches" on page 31.
2. Make sure the branch you want to delete is not selected as the active branch.
3. Choose the branch you want to remove, and select **Delete Branch**.

> NOTE If you want to remove a branch only from your computer, just delete the local branch. If you want to remove a branch altogether, delete both the local and remote branches.