BROADCOM®
MAINFRAME SOFTWARE

Endevor®
Bridging Your Enterprise Git Server with Endevor

White Paper

Table of Contents

Chapter 1: Introduction

In many companies, a new generation of developers is finding its place to work alongside experienced mainframe developers. Have you been faced with the challenges stemming from the different backgrounds, habits, and needs of the two groups? Established mainframe developers who are familiar with traditional mainframe tools such as Endevor® and are content working on the green screen stand apart from the young developers coming out of university or previous projects. These young developers are familiar with development centers on Git, and modern IDEs such as Visual Studio Code or Eclipse Che. Imagine their surprise when they first encounter the world of mainframe, and they are asked to edit their code using ISPF.

Figure 1: Version Control
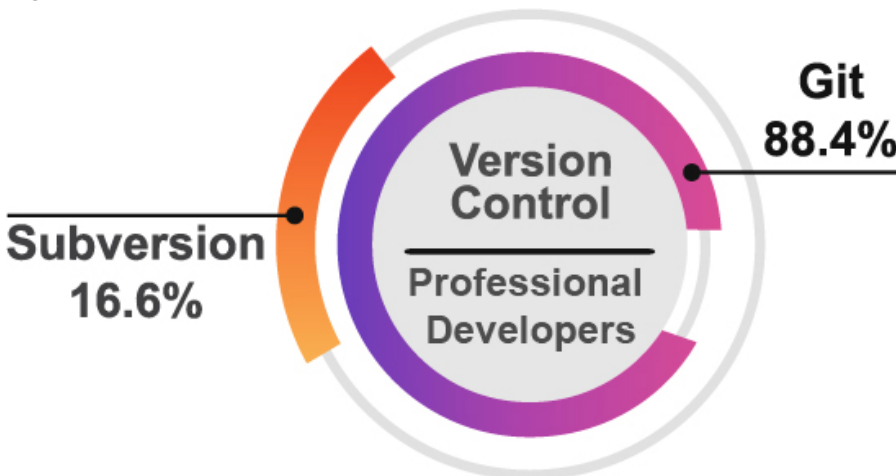
The Broadcom strategy in this dichotomy is to enable choice, rather than force a change. The goal is to merge the two worlds and connect the current silos of mainframe and distributed application development. Enabling the modern developer experience on the mainframe means three things:

- Git for collaboration
- Freedom of choice regarding the IDE for code editing
- Command line interface (CLI) to access the mainframe

Even though CLI might not appear to be the first choice for most modern developers, it has been regaining popularity thanks to its cross-platform flexibility and ability to be used in scripts. Popular tools such as Git rely on CLI, and a console is available in most modern IDEs. Achieving cross-platform consistency means that the same set of tools can be used no matter the target platform. A consistent set of tools should make life easier for developers spanning distributed applications and mainframe applications, frontend and backend.

This approach delivers the following benefits:

- Attract the new generation of developers to mainframe (easy on-board, common skill-set, defacto standard collaboration practices)
- Support common practices for end-to-end enterprise application development (mobile to mainframe)
- Offer a similar experience across all platforms (distributed, cloud, mobile, mainframe)
- Leverage popular, proven collaboration practices
- Protect investment in mainframe software asset management and lifecycle automation

Chapter 2: Design Thinking

When designing solutions, Broadcom teams want to understand the users and their everyday challenges, expectations, frustrations, and desires. Looking at the individual profiles of people working in Broadcom customer organizations, and observing patterns has enabled the creation and definition of specific personas. These personas fall under two categories of users—developers and tool administrators.

2.1 Developers

The primary developer personas are Michelle and Rob. They are set apart in their background and experience on the mainframe. Michelle is a bright young developer who knows modern languages such as TypeScript, Java, .Net, and C++. She also knows some COBOL, and has enough knowledge of Endevor to work efficiently. From her perspective, she does not understand why mainframe user interface evolution stopped at the green screen. Even in the case of modern mainframe UIs, the user experience is still inspired to a great extent by the green screen. Her previous experience involved Git and modern IDEs. Her colleague, Rob, has been working with Endevor for ages. His interaction with ISPF is so smooth that he does not want to even hear about any modern UIs. He knows what works best for him and is rather skeptical about new solutions claiming an improved user experience.

Persona - Michelle
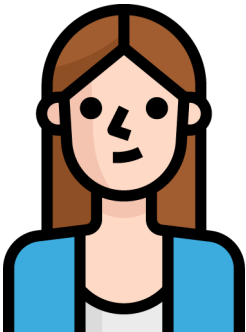
Profile – Application Developer

- Programs in TypeScript, Java, Net, C#, C++, SQL.
- **Uses Git.**
- Uses Eclipse Che, Visual Studio Code, and other modern IDEs.
- **Not interested in working with mainframe tools or mainframe environments.**
- Interested in building new apps, extending existing applications, and modernization.

Persona - Rob

Profile - Application Developer – Experienced

- Has extensive COBOL and/or PL/I programming.
- Understands mainframe database application concepts.
- Writes code to create software applications or update existing software applications.
- Enjoys working with **traditional mainframe tools**.

2.2 Tool Administrators

Carl the Endevor administrator, manages and maintains the development environment for Rob. He understands the specifics of that particular Endevor implementation. He will naturally want to protect the investment put into Endevor processes over the years because he can appreciate the complexity of potential conversion to a new system. Conversely, Todd, the senior operations engineer, does not have as much insight into mainframe development tools and believes moving all application development under Git is the way to go.

Persona - Carl
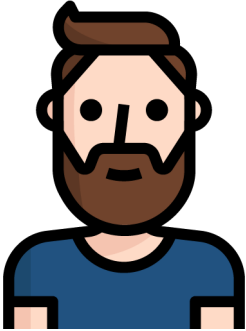
Profile - Endevor Administrator

- **Ensures that Endevor is running at the most current release level.**
- Creates and manages software development build and lifecycle automation.
- Participates in product upgrade and product maintenance to some extent.

Persona - Todd
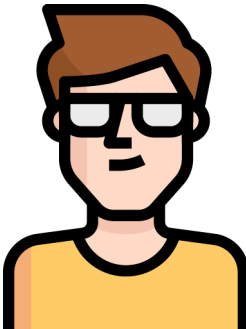
Profile - Senior Operations Engineer

- Aligns Git setup with development processes.
- **Helps to move engineering teams into Git.**
- **Creates and initializes Git repositories.**

2.3 Michelle versus Rob – Todd versus Carl

Different species of IT professionals from different backgrounds with different approaches, yet they need to work together on common projects. The challenge is to find common ground for all of them, so that they can be as productive and comfortable in their work environment as possible.

The concept of personas is a constant reminder of who we are working for. During a recent validation exercise with Endevor customers, more than 80% confirmed that the current challenge they are facing is how to onboard young talent and have them smoothly collaborate with existing mainframe teams. So, the question we are looking at is, "How can we help Michelle and Rob to work together and be efficient?" Related to that question, "How can we support Todd and Carl in their efforts to provide an ideal development environment to their programmers, both modern and traditional?"

The following sections describe the solution Broadcom offers along with Endevor.

Chapter 3: Endevor Bridge for Git

The Endevor Bridge for Git is a part of the Endevor Integrations for Enterprise DevOps package. To enable developers to benefit from the advanced team collaboration capabilities, Bridge for Git connects Git to Endevor through an Enterprise Git Server, such as GitHub, Bitbucket, GitLab, or Azure DevOps. Only code that is pushed from the local Git repository to the Enterprise Git repository is recognized and propagated to Endevor by the integration.

In this integration, the branching and merging capabilities of Git complement Endevor purely as source control and a collaboration tool. For Rob, Endevor keeps the role of source control, and serves as a build machine and life cycle automation system for the entire application. The user experience remains exactly the same for both Rob on Endevor and for Michelle in Git—including their related tools.

3.1 Git versus Endevor – Differences and Challenges

Main takeaways:
- *Git and Endevor are different in several ways:*
  - *Data organization – hierarchy, partitioning, and size.*
  - *Location of application files and elements:*
    - *Git has all files available in the working directory so developer builds can be performed.*
    - *Endevor only has the elements that are being modified in the working stage—the Endevor map is used to find the other elements needed at build time.*

Git and Endevor in many aspects are inherently different. There is not a direct way to link them one to one. Some differences stem from the underlying operating system, others are due to the SCM design. Before comparing Git and Endevor, let us start with a quick terminology clarification for the purpose of this document.

Endevor manages business applications. These applications can often be independent from one another. In other cases, applications do not have clear boundaries and are melted or blended together. This document uses the word *application* with the assumption that there is a clear understanding of the application *boundaries*, such as the set of all artifacts that make the application including source code, binaries, JCL/Rexx scripts, and so on. The granularity of what is meant by an application depends on the environment.

Now, back to Git and Endevor. In the following section we will first outline the main differences between the two systems, and then review the recommended approaches

Figure 2: Git Repository
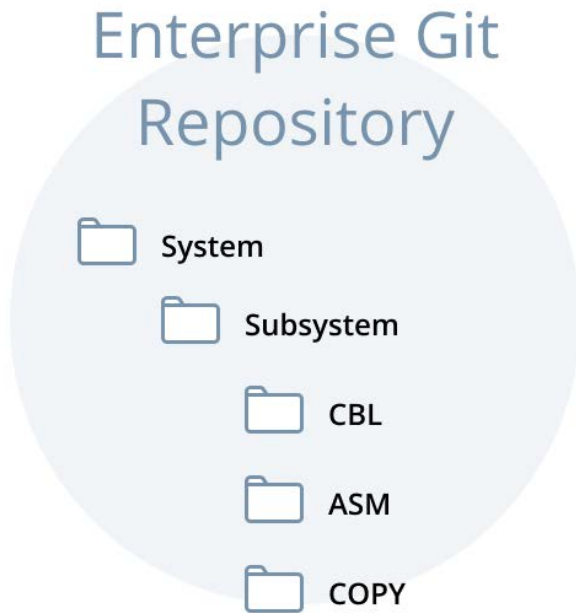
The file structure in Git allows for an arbitrary organization into projects with files in a folder hierarchy of virtually infinite complexity and depth of folders, sub-folders, sub-folders of sub-folders, and so on. Alternatively, inventories in Endevor are organized in a much flatter structure that uses systems, subsystems, and these elements are further defined by types. As for the size of projects, usually a Git project can range from a few files to a few thousand, and in some cases it can scale up to 50K files. In Endevor, there is a vast range in the size of systems and subsystems, which can contain hundreds of thousands of elements. It is important to understand that copying Endevor elements to Git will maintain the Endevor-like flat structure. This needs to be considered when deciding on a synchronization strategy for a particular Endevor application with respect to the number of elements.

Another thing to consider is how application code is distributed across systems and subsystems in Endevor. In simple cases, one application corresponds to a system, and subsystems correspond either to individual components, or to releases. However, depending on particular Endevor implementation, it is common to have one application spanning multiple systems and subsystems, or the other way round—one system and subsystem being shared among multiple applications, or even a combination of both. In this last case, file naming conventions may be used to distinguish which elements belong to which application. These complex situations must be analyzed before synchronizing with Git.

In an enterprise Git server it is natural to have all files related to an application or a component available in one or more repositories for Michelle to clone and work on locally. All files are then present in her local working directory so that it is possible to perform a build at any time. In Endevor, the user experience is not that different—Rob can also access all the elements that he needs to work with. However, the elements are not all physically located in one place. The elements that are not currently being modified, are placed in the last environment and stage of the Endevor life-cycle, often named Production. Whenever Rob wants to edit an element, he retrieves it to the working stage—let's call it the Development environment stage for the purpose of this document. As the modified element progresses through the life-cycle, it gets promoted from Development to later stages. For example, the element goes from QA and to Production using Endevor packages. As for building, Endevor performs a build of each element after every change. The build processors are able to search through the Endevor map for any dependencies required for generating the particular element. Hence, it is not necessary to have all elements in one location. We use this concept in the following sections when outlining the recommended solution.

3.2 Endevor Map and SDLC

It is important to have an understanding of the concept of the Endevor map and the software development life-cycle (SDLC). The Endevor map consists of any number of environments and stages that are linked together to create a path where software changes can progress from a development work area to a production deployment.

In the following example, there are three environments: Dev, QA, and Prod. Each Endevor environment usually has two active stages (S1 and S2), of which one is an entry stage (marked with a little door icon in the following figure). The thin yellow arrow shows how the life-cycle is defined. In this case, it follows all the stages one after another up until QA S2. Changes from QA S2 are then promoted directly to Prod S2. Prod S1 is dedicated to hot-fix changes, which are allowed to skip the regular life-cycle path and are promoted directly to Prod S2.

Figure 3: Example Environment
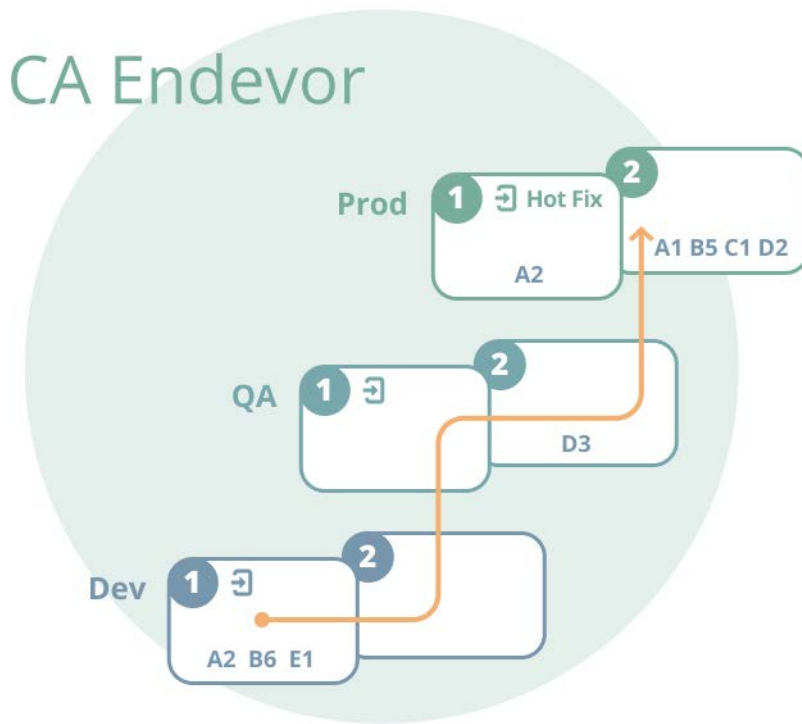
The figure also shows how elements can be distributed along the Endevor life-cycle map:

■ Version 1 of element C (C1) is only present in Prod S2. If Rob wants to modify this element, he must retrieve it from Prod S2, modify it, and then check it in at Dev S1

■ E1 is a new element that Rob has created and added to Dev S1. It is not present in Prod S2.

■ Element D2 has been modified and has been promoted, and its version 3 (D3) is now in QA S2 waiting for final testing to complete before it can be promoted to Prod S2.

■ Element B5 is currently being modified in Dev S1 as B6. When Rob completes his changes, B6 is promoted through the life-cycle as unit testing and integration testing are successfully completed.

■ Element A1 is currently being modified in the Development environment Dev S1. At the same time, it is also modified as a hot-fix in Prod S1. These two versions of element A are both labeled A2, however they differ in content. Let's say that the version from hot-fix stage Prod S1 is the first promoted to Prod S2. In this case the version at Dev S1 must base its changes on the new A2 version coming from Prod S1 (this corresponds to the Git concept for merge). After that the element coming from Dev S1 becomes A3.

NOTE: The Endevor map can include elements from multiple systems and subsystems.

3.3 Synchronizing Endevor with Git

Main takeaways:

- *The Endevor to Git Synchronization using the Bridge for Git involves the following actions:*
  - *Creating Endevor Connections*
  - *A branch in a Git repository is associated to a location in Endevor defined by environment, stage, system, and subsystem*
  - *A Git branch is associated to a Endevor entry stage for editable branches, or any stage for read-only branches*
  - *There are multiple mapping types:*
    - *Simple mapping*
    - *Basic multi-branch mapping*
    - *Advanced multi-branch mapping*
  - *There are two synchronization options:*
    - *Synchronize all stages in Endevor map*
    - *Synchronize only the elements from the work environment*

To enable Rob and Michelle to collaborate on a project, Carl and Todd need to work together to connect Endevor and Git, associating one Enterprise Git repository to a location in Endevor. This association is done by creating a Git-Endevor mapping using the Endevor Bridge for Git. Based on the structure of the selected Endevor location and on the team's development processes, one of the following mapping types will be used.

3.3.1 Creating Endevor Connections

An Endevor Connection consists of a base URL and a configuration of Endevor Web Services. Administrators such as Carl and Todd must work together and set up the correct Endevor Connections so that Mapping Administrators use the correct configuration of Endevor when mapping an inventory.

Additionally, both Mapping Administrators and Developers (or anyone who works with elements in mappings) need to ensure that they provide their credentials for the particular Endevor Connection. Their credentials are used for the synchronization of changes from Git to Endevor.

NOTE: When a Mapping Administrator creates a mapping, their credentials are used to synchronize changes from Endevor to Git. If you do not want to use individual credentials for this communication from Endevor to Git, you can use certificates for such connections.

3.3.2 Simple Mapping – Single System with a Single Branch

Simple mapping associates one enterprise Git repository to one Endevor system and subsystem. A designated Git branch, usually the main branch, is associated with one single stage in the Endevor life-cycle. The selected stage must be an entry stage in case the mapping is intended to allow Git to Endevor synchronization. If the mapping is read-only where changes will never be propagated from Git to Endevor, any Endevor stage can be used, entry or non-entry. Examples of where read-only mappings might be useful include troubleshooting application bugs at a given stage or cherry picking changes coming from a different development stream higher in the map.

It is possible to create more branches in the Git repository after the repository initialization, but with this mapping type, only one branch will be synchronized with Endevor.

NOTE: In this document the Bridge for Git always drives the synchronization through the Development entry stage.

Figure 4: Simple Mapping – Single System with a Single Branch
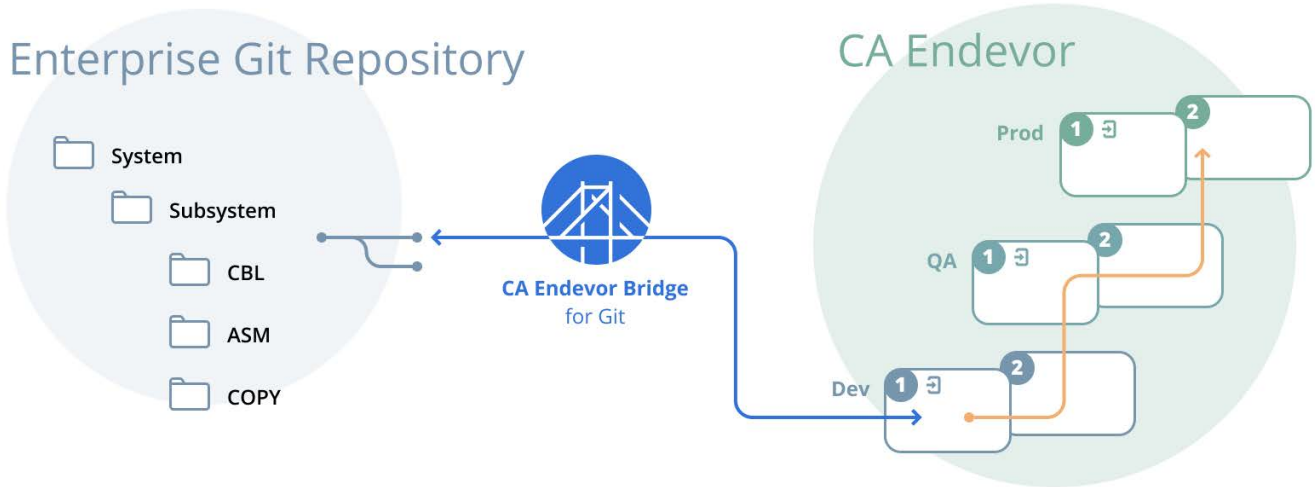
3.3.3 Basic Multi-Branch Mapping – Multi-System Definition per Repository

Basic multi-branch mapping associates one enterprise Git repository to any number of Endevor systems and subsystems. Multiple synchronized branches can be created, corresponding to different Endevor environments and stages. As an example, this mapping type can be used by teams who have different life-cycles for regular new value development as opposed to hotfix changes. One synchronized branch can be associated to the Development entry stage for the regular development life-cycle, and another synchronized branch for hotfixes can be linked directly to a pre-prod stage, such as Prod 1 in our example.

In this mapping type, the selected systems and subsystems are identical for all synchronized branches.

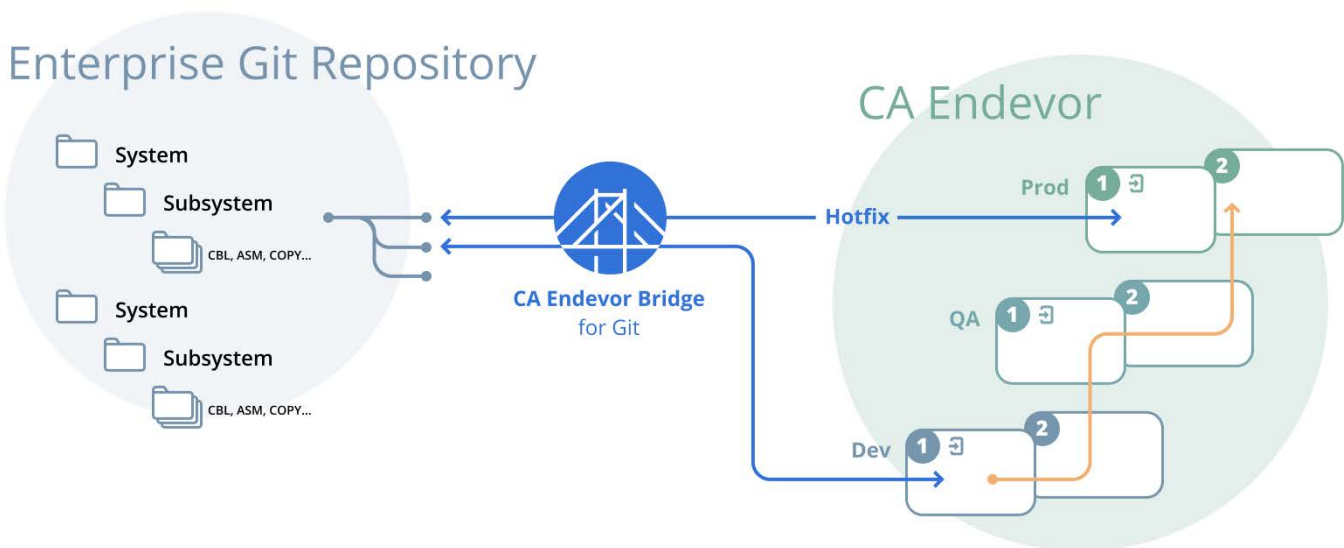**Figure 5: Basic Multi-Branch Mapping – Multi-System Definition per Repository**

3.3.4 Advanced Multi-Branch Mapping – Multi-System Definition per Branch

Advanced multi-branch mapping associates one enterprise Git repository to any number of Endevor systems and subsystems. Multiple synchronized branches can be created, corresponding to different Endevor environments, stages, systems and subsystems. This mapping type can be used for any complex use case not covered by the previous mapping types, typically when a specific setup is required for each branch. As an example, teams who want to synchronize sandboxes (a Endevor sandbox is defined as a subsystem) to branches will benefit from advanced multi-branching.

This mapping type is the most flexible in terms of branch configuration. Some synchronized branches might only differ in environment and stage (similar to the previous hotfix example), while other synchronized branches will differ in the system/subsystem definition (the *advanced* branches).

NOTE: To keep the following visual simple, subsystems are only displayed in the Dev environment, but they exist in all environments, and are not necessarily the same across the lifecycle.

Figure 6: Advanced Multi-Branch Mapping – Multi-System Definition per Branch
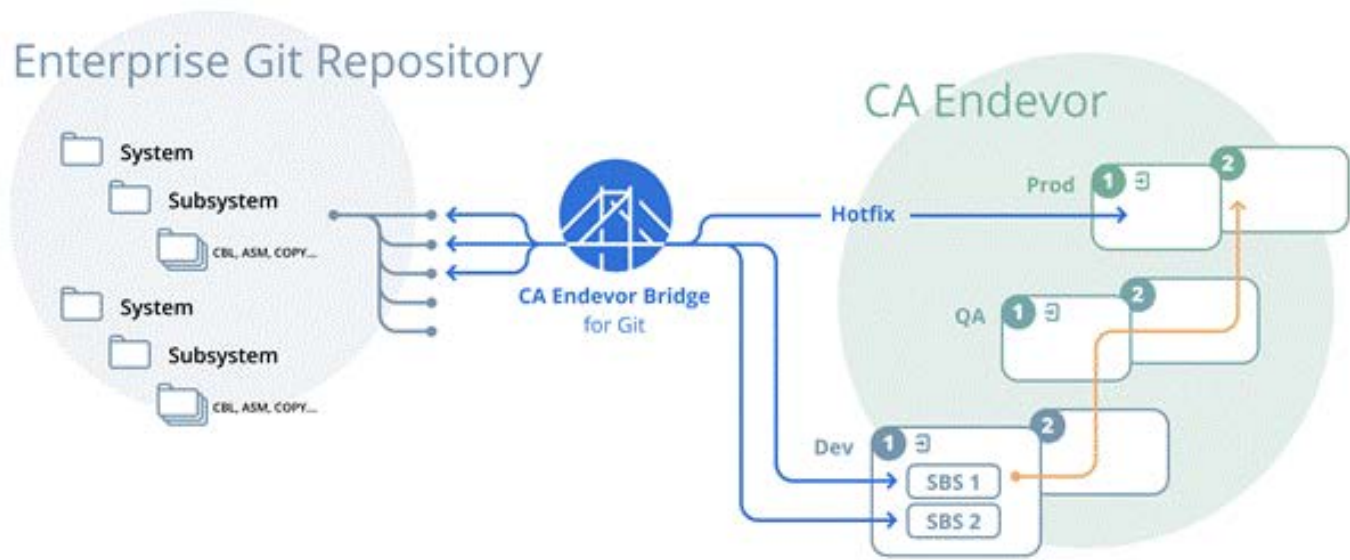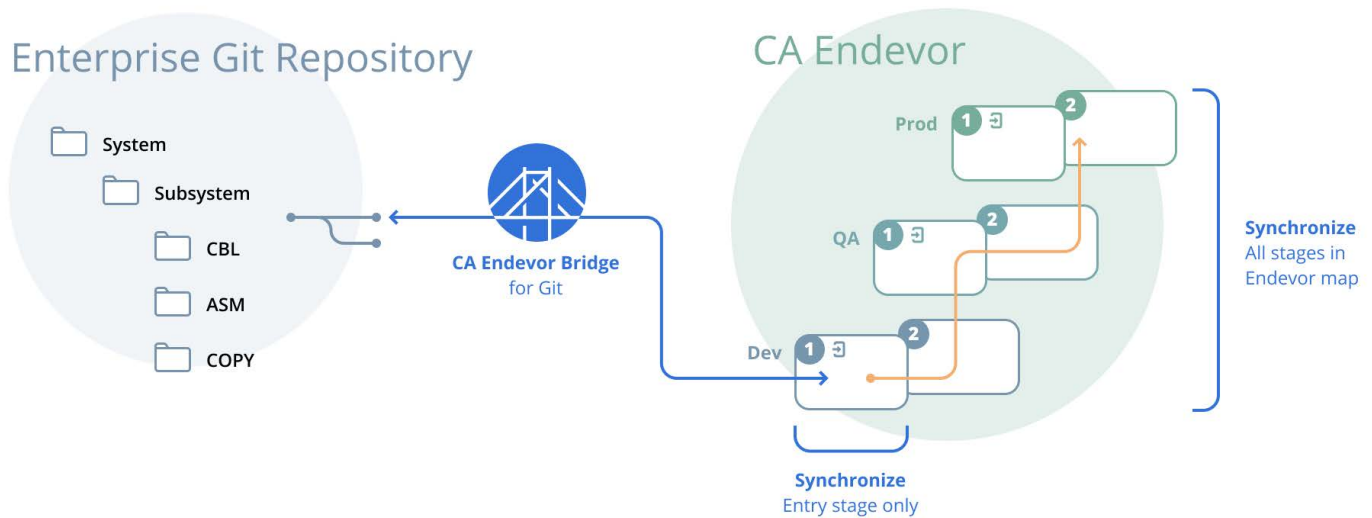

3.4 Synchronization Options

To address the differences and challenges outlined in the section Git versus Endevor – Differences and Challenges, and to enable synchronization of very large Endevor inventories, the Bridge for Git offers two options:

- Synchronize all stages in the Endevor map, or
- Synchronize only elements from the working environment

Figure 7: Bridge for Git Options

3.4.1 Option 1: Synchronize All Stages in Endevor Map

All files will be available in the Enterprise Git repository:

This first option is suitable for applications having a reasonable number of Endevor elements that all can be synchronized in the enterprise Git repository, and consequently made available in the local working directory of Michelle. Regardless of the life-cycle stage the elements in the Endevor map are placed in, they are all synchronized in an enterprise Git repository through the Development entry stage. The elements will be placed in folders in the repository organized by type.

3.4.2 Option 2: Synchronize Only Elements from Work Environment

Only elements from Development entry stage will be in the enterprise Git repository

If you do not want to have all elements synchronized over to the enterprise Git repository, number of elements in Endevor is too large. The Bridge for Git provides the ability to synchronize elements only from the Endevor Development entry stage, ignoring all the other elements located in higher stages of the Endevor map.

This option ensures synchronization of a reasonable amount of data while allowing Git users to have access to the elements they require. In general, the Development entry stage only holds around 5% or less of the total number of elements.

If Michelle needs an element that is in one of the higher stages of the Endevor map, she can get it in one of the following ways:

- **(Recommended)** Michelle can pre-empt synchronization by using the Bridge for Git plug-in for Zowe CLI to copy elements from up the map directly into the synchronized repository, with or without its dependencies.

  Example Syntax: `zowe ebg mapping add-element` *elementName* `--repo-context` *ORGANIZATION* `--repo` *REPONAME* `--type` *TYPENAME*

- **(Recommended)** Michelle does everything directly from Visual Studio Code or Eclipse Che. Using the Bridge for Git Explorer extension, she is able to see the elements that exist up the Endevor map from her work environment and then can add the required element(s) to the synchronized repo, optionally with dependencies. Once it is added to the remote synchronized repository, Michelle can pull the missing element(s) to her local working directory. When she makes changes in a subsequent commit and pushes them back to the Enterprise repo, the element(s) will be added to the Development entry stage.

■ Michelle uses the Endevor plug-in for Zowe CLI to copy the element to the Endevor Development entry stage, using the command generate with copyback.

Example Syntax: `zowe endevor generate element` *elementName* `--env` *DEV* `--sn 1 --sys` *SYSNAME* `--sub` *SUBNAME* `--type` *TYPENAME* `--copy-back -i` *ENDEVOR*

The next time a synchronization runs in the Bridge for Git, it will automatically bring this element to the synchronized repository from where Michelle can pull the elements to her local repository.

■ Michelle uses the Endevor plug-in for Eclipse to retrieve the element to a local project and select the option to immediately add it to the Endevor Development entry stage. The next time a synchronization runs in the Bridge for Git, it will automatically bring this element to the synchronized repository from where Michelle can pull the elements to her local repository.

As a general rule with Option 2 Work Elements Only synchronization, the enterprise Git repository only contains elements from the Development entry stage. On top of that rule, the Bridge for Git identifies elements which Michelle has already modified or pulled into the repository on demand, and those remain synchronized in the enterprise Git repository even if they get promoted to a higher stage in the Endevor lifecycle and disappear from the Development entry stage. This ensures that Michelle's user experience is more consistent in the sense that *her* elements are not removed by the synchronization process in the background without her knowing it. They remain in the enterprise Git repository unless she or another Git user removes them (in which case they are still present in one of the higher stages in Endevor) or unless they are removed from all Endevor stages completely.

3.4.3 Using Filters

In the Mappings, Synchronization options, and scenarios described previously, the Bridge for Git allows for a reduction in the number of synchronized elements through the use of filters that are based on element type and element name. When creating a mapping, Bridge for Git makes a request to Endevor and provides a list of the available element types for each system or subsystem that is included in the inventory of the mapping. At the same time, mapping administrators can specify names or parts of names using wildcards (standard Endevor wildcards apply, * [asterisk] and % [percent]) in an entry field for an element name.

For example, if Michelle only needs to work on COBOL files and copybooks with the name prefix of FIN, filtering by COBOL and CPY type (depending on the file extension setting in Endevor) and `FIN*` is applied. Filtering can substantially reduce the number of synchronized elements, which improves the performance of the initial synchronization process, makes the Git repository less cluttered, and provides a better user experience to Michelle.

NOTE: If a developer wishes to add an element or elements of a different type than is present in a mapping that uses filters, they can. The element will be added and synchronized in the appropriate type folder.

3.4.4 Bi-directional synchronization

In all mapping types and synchronization options, once initialized, the synchronization goes both ways between Git and Endevor.

3.4.4.1 Synchronization from Git to Endevor

This Git-Endevor forward synchronization makes Michelle's changes (done in Git) available to Rob and subsequently have them managed by Endevor. The synchronization is triggered immediately after Michelle's changes get merged into the synchronized branch in the synchronized enterprise Git repository.

3.4.4.2 Synchronization from Endevor to Git

This synchronization from Endevor to Git, also called sync-back, makes Rob's changes (done in Endevor) available to Michelle in Git. The Bridge for Git offers two mechanisms for the sync-back:

- Regular (polling) sync-back – Runs at a desired frequency, scans the relevant Endevor inventory and retrieves all new changes back to the Git repository.
    - An administrator can set a sync-back interval with a specific start date and time for the whole application.
    - Alternatively, mapping administrators can set sync-back intervals for their mappings.
- Event-based sync-back – Using the Mainframe Webhook server, actions happening in Endevor are monitored and the sync-back is immediately triggered for any new change in the synchronized inventory.

3.4.5 Configuration of Options after Initialization

After the mapping is initialized, the mapping administrator can configure several options that affect the synchronization of changed elements to Endevor.

3.4.5.1 Generate after Update

The mapping administrator can toggle `Generate after Update` on for a mapping so that elements from the mapping that are successfully synchronized to Endevor will be generated. Generate actions are performed with the same user ID, CCID, and comment as the original synchronization.

3.4.5.2 CCID, Comment, and From-location

The mapping administrator can also set expressions or string literals that are used for the CCID, Comment, and From-location for elements changes in the mapping. These values will be used instead of the default behavior, which is described in the following section.

3.4.5.2.1 Default Behavior

Bridge for Git allows developers to provide CCID and Comment values in the commit field for their changes using the "CCID=" pattern. The first eight characters following "CCID=" are used for the CCID, while other values are populated to the comment field. The first seven characters of the Comment field are reserved for the first seven characters of the git hash for a commit. The from-location value uses "Bridge for Git" plus the identifier that is specified in the application.yml.

3.4.5.3 Webhooks

Mapping administrators can also configure webhooks for a particular mapping to send information about events and their results in a mapping to a specified endpoint. One practical use case of the webhooks is to create a webhook that sends information of successful synchronization events to a Jenkins pipeline that kicks off automated testing. In this way, more of the development process is driven through the developer's actions in a Git-Endevor mapping.

3.5 Developer's Build and the Use of Work Areas

Main takeaways:

■ *An Endevor work area is needed to perform the developer build*
■ *Michelle can build her changes in her isolated work area in Endevor before committing*
■ *Developer's build prevents compilation errors from being pushed from Git to Endevor*

With Git, Michelle is used to having all files available in her local working directory. This availability enables her to build and test her changes at any time before committing them and sharing them with the rest of the team. In Endevor, a generate action can automatically run to build the given element immediately whenever an element is changed and the changes are saved. Even though the experience is different, both Michelle and Rob are used to having timely feedback on their changes. The challenge here is that in most cases, Endevor applications cannot be built outside Endevor. So how can Michelle validate her changes in Git before committing and pushing them to Endevor?

The Bridge for Git provides a developer's build. The developer's build enables Michelle to build her changes in isolation, in a designated location in Endevor, using the Endevor Bridge for Git plug-in for Zowe CLI directly from her local machine. She is able to commit after successfully building and unit testing. So, how does it work?

To benefit from this functionality, Carl has created a work area for Michelle located in a Working environment directly under the Development entry stage. Having the work area located under the Development entry stage substantially improves the performance of each build. The Developer's build uploads to the Endevor work area only those elements which Michelle actually changed, rather than the entire working directory. She then does not need to worry about dependencies, because Endevor does the work for her under cover.

The following figure is an example of a Endevor map with the required setup for the Developer's build to function correctly.

Figure 8: Example Endevor Map
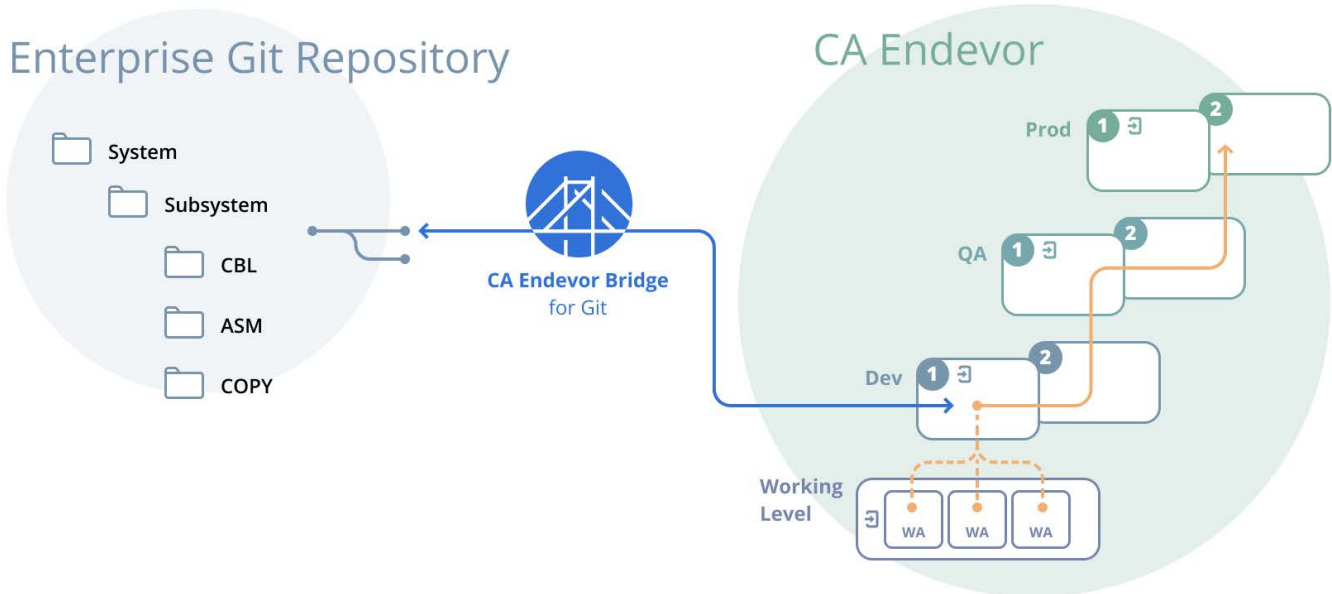
NOTE: There are a number of ways to provide work areas to developers, such as Endevor sandboxes, static work areas, or dynamic environments. If static work areas are in use, they may be shared and reserved, or unreserved to indicate their current usage. Another option is for each developer to have their own work area.

If none of the previous scenarios work due to lifecycle constraints, another possibility for developer builds is to use Endevor Team Build, which is included in the Endevor Integrations for Enterprise DevOps option.

Endevor Team Build lets you initialize a build project from the processors and code in Endevor, which you can then use to build application changes outside the traditional Endevor infrastructure. For more information about this option, see the Endevor Team Build documentation.

3.6 A Day in the Life of Michelle

Main takeaways:

■ *Michelle's one-time setup tasks:*
  – *Provide credentials for the Endevor Connection used by the mapping*
  – *Clone the enterprise Git repository*
  – *Set up the Developer's build with her Endevor work area and her Zowe CLI profile*
  – *(Optional) Set up a local pre-check helping prevent synchronization issues*
■ *Michelle's workflow:*
  1. *Create a feature branch*
  2. *Edit Code with VS Code*
  3. *Build with Endevor*
  4. *Test*
  5. *Git pull to integrate other users' changes*
  6. *Iterate the above as needed*
  7. *Git commit*
  8. *Git push with a pre-check*
  9. *Enterprise Git server: create a Pull request*

Michelle is joining Rob's team to help in a long-term project developing new features for a mainframe application. She usually works on front-end, but she also has knowledge of COBOL that she can use in this assignment.

After Todd and Carl set up the Git-Endevor mapping for the application, Michelle is directed to Bridge for Git where she adds her credentials to the Endevor Connection that is used by the mapping for synchronization. She can alternatively add her credentials through the Endevor Bridge for Git plug-in for the Zowe CLI. Michelle views the details of the mapping and the options that are set, and then she navigates to the Git repository.

This Git repository is managed in an enterprise Git server from which Michelle can clone. She creates a development branch in her local Git repository, and updates her Zowe CLI user profiles. Michelle can now edit elements in her working directory.

Michelle uses her preferred IDE, for example Visual Studio Code, and she makes the necessary changes on selected files. She can optionally set a non-default processor group for any of the elements by navigating to the `bridge.json` file in the element type folder and changing the processor group entry for the particular elements. The processor group will not only be used for the build action, but also for the generate action (if the generate after update feature is enabled for her mapping).

Now that she has edited the files and specified the correct processor group, she can now run the Developer's build using the Bridge plug-in for Zowe CLI, and she is notified of the result without having to access the green screen. If there are any issues with the compilation, she is able to consult her listings and make additional changes. She can then deploy the build results in her test environment and perform unit testing, leveraging the Endevor plugin for Zowe CLI. Once Michelle is happy and confident with what she did, she pulls the latest changes from the Enterprise Git repository, merges the changes, and reruns the build testing again if necessary.

Following these steps, Michelle is ready to commit and push to the Enterprise Git repository. In the Git commit message field she provides a CCID and comment that will be propagated to Endevor by specifying `CCID=<value> <comment>`. Alternatively, she does not enter any value and the mapping uses the expressions her mapping administrator set for the mapping. She then commits and pushes, with the push validated by a precheck. Her changes are synchronized to the enterprise Git server and there she is able to create a pull request so that her changes can be reviewed and merged in the synchronized branch. The Bridge for Git then immediately propagates her changes to Endevor and the changes are available to the rest of her team, regardless of whether they use Git or Endevor.

Figure 9: Local Repository
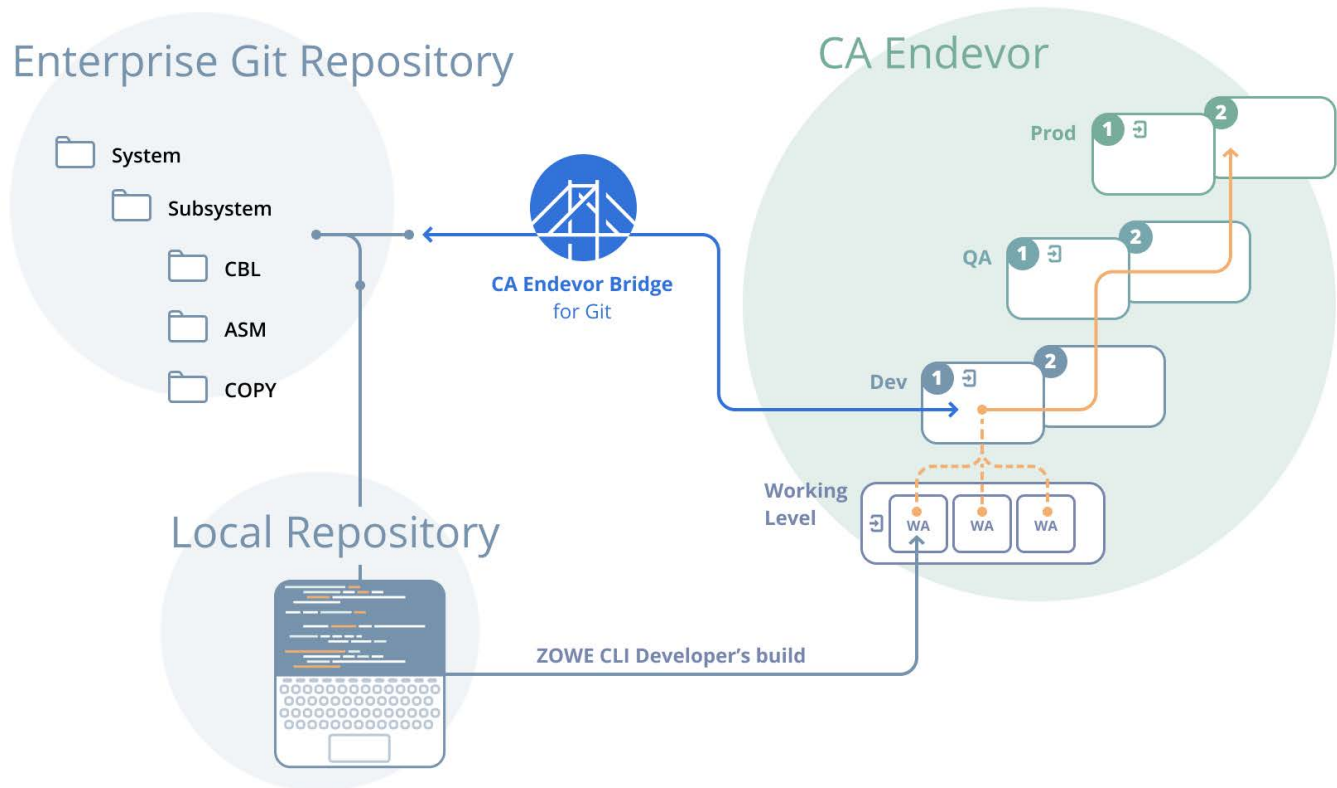
Michelle's experience working on a mainframe application using a repository synchronized with Endevor is no different from what she does when doing code changes on the front-end.

Videos:

- Michelle's one-time setup tasks
- Michelle's workflow

3.7 Out-of-Sync Changes Management

Main takeaways:

- *When Michelle's pushed changes are in conflict with Rob's changes done in Endevor or contain a user error:*
  - *Endevor is the single source of truth.*
  - *Out-of-sync changes are reverted and placed in a revert branch.*
  - *Michelle can fix any errors and/or use Git merging capability to merge with the latest changes from Endevor and push again.*

Let's now take a look at how collaboration between Rob and Michelle will work. Rob edits his code using the green screen or Eclipse UI and saves his changes directly in Endevor the way he is used to. Michelle works alongside him, in her favorite IDE, and pushes her changes in the enterprise Git repository.

Figure 10: Collaboration
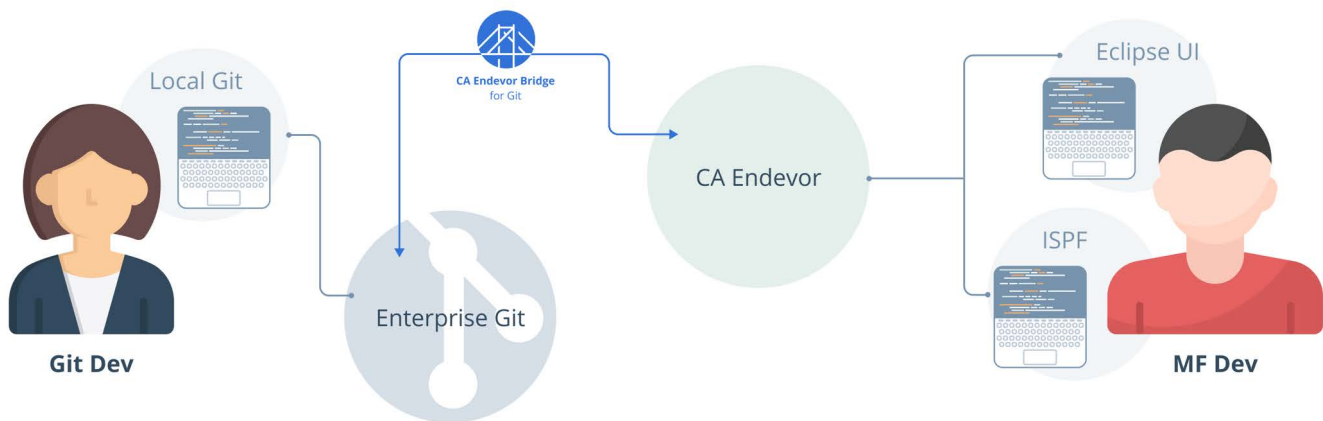
To prevent out-of-sync issues when pushing her changes to the synchronized branch, Michelle is encouraged to use common Git practices—always fetch the latest changes from the enterprise Git repository and merge her code with them before pushing. The frequency of Endevor to enterprise Git server synchronization determines how up-to-date the synchronized branch will be most of the time. The more frequent synchronization, the smaller the probability of a conflict will be.

An issue in synchronization still can occur, be it because of a conflicting change performed in Endevor after the last synchronization, or because it is rejected by Endevor due to a user error (for example, an incorrect type or record length). In these cases, Endevor remains the single source of truth. The Bridge for Git delegates the resolution to the enterprise Git server and leverages its powerful merging capability by always reverting conflicting changes back to a consistent state. A new revert branch gets created containing Michelle's changes, so that she can pull the latest version, correct anything if necessary and finally merge before pushing again.
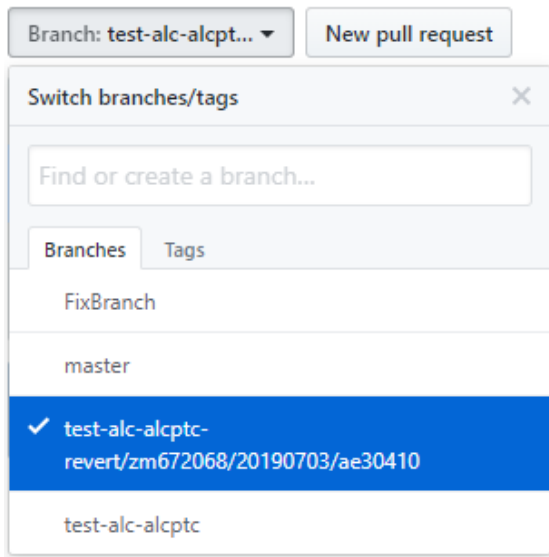
Figure 11: Revert Branch Gets Created

Figure 12: Information About Reverted Commits
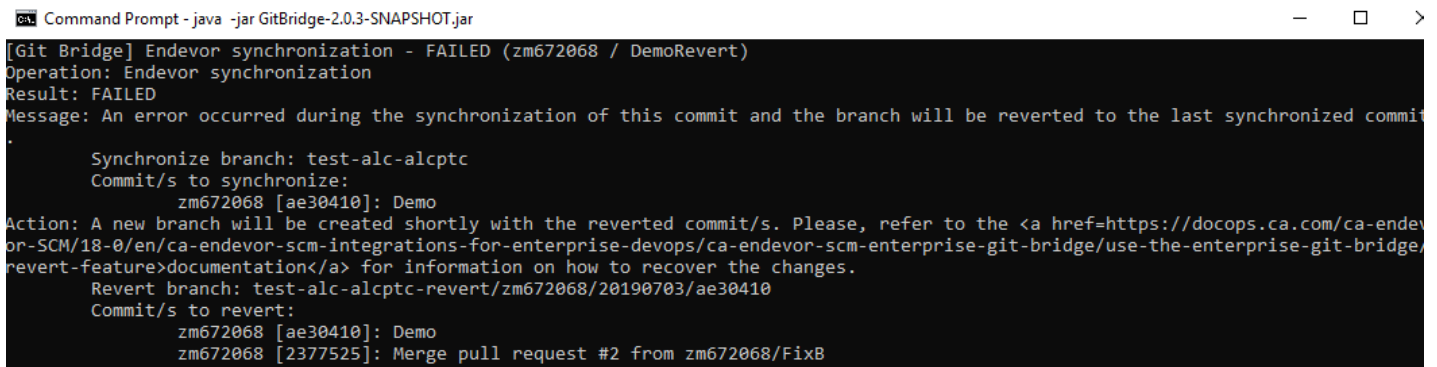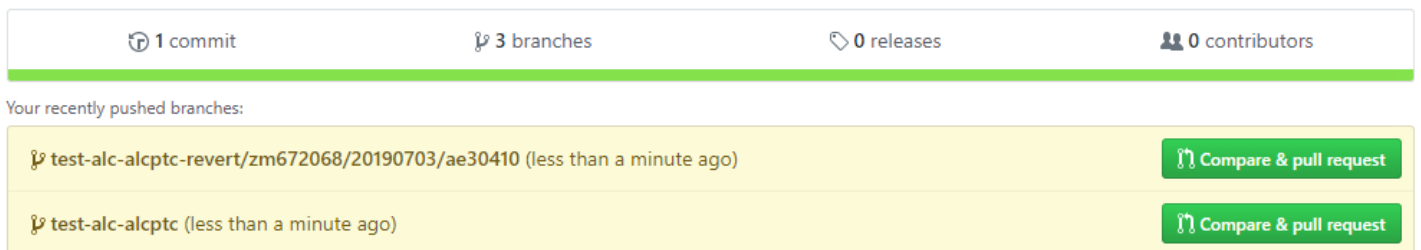

Figure 13: Squash and Merge the Commits from the Revert Branch

Chapter 4: Summary

This document outlines the general principles of the Bridge for Git. To summarize the main key points:

- Setting up the Bridge for Git does not require changes in any of the following items for Endevor:
    - The system and sub-system structure
    - The application build process
    - The lifecycle management
- The Bridge for Git protects the user experience for Rob and does not require that he changes the way he works
- The Bridge for Git enables Michelle to have the desired user experience—enjoying the benefits of team collaboration provided by the enterprise Git server.

This makes the deployment of the Bridge for Git fairly transparent and enables a very flexible adoption by teams.

Endevor Bridge for Git is a part of the Endevor Integrations for Enterprise DevOps.

If you are interested in learning more through a demo or a workshop, contact us at: gitbridge@broadcom.com.

Revision History

Endevor-SCM-WP103; June 23, 2022

- Updated 3.1 Git versus Endevor – Differences and Challenges
- Updated 3.3 Synchronizing Endevor with Git
- Updated 3.3.1 Creating Endevor Connections
- Updated 3.4.3 Using Filters
- Updated 3.4.4.2 Synchronization from Endevor to Git
- Updated 3.4.5 Configuration of Options after Initialization
- Updated 3.5 Developer's Build and the Use of Work Areas
- Updated 3.6 A Day in the Life of Michelle

Endevor-SCM-WP102; February 1, 2021

- Updated 2.2 Tool Administrators
- Updated 3.3 Synchronizing Endevor with Git
- Updated 3.4 Synchronization Options
- Updated Chapter 4, Summary

Endevor-SCM-WP101; March 20, 2020

- Updated Chapter 1, Introduction
- Updated Chapter 2, Design Thinking
- Updated Chapter 3, Endevor Bridge for Git
- Updated Chapter 4, Summary