COPIA

Intro to Git-Based Version Control for Industrial Automation

The Value Of Version Control For Industrial Automation

Good version control practices are essential for efficient code development. They ensure that you, your team, and your company can track changes to files over time, understand why the changes were made and by who, and revert to specific versions of code if needed. When a robust version control system is implemented correctly, teams can focus more on the development activities instead of searching for and investigating code changes. As more engineers are assigned to the project or more time passes between project activities, the benefits of proper version control increase. Less time is required to understand how and why a project arrived at its current state.

For an industrial automation developer, a sound version control system will ensure you can always answer the following questions:

- Where is the latest version of the PLC code, and can I access it without calling another developer?
- Is this version of code the same as that has been deployed onto the PLC?
- Was the latest version reviewed/approved by the proper people?
- What has changed between this file and the previous version? Who made the change and why?
- Can I control which persons can access the files at different stages of the PLC code's lifecycle?

While there are many different types of version control systems, this document focuses on Git, which dominates the software industry with over 80% market share and is used by ~100 million developers.

What is Git?

Git is a mature and actively maintained open-source tool created in 2005 by Linus Torvalds (the founder of Linux). It is now the most widely used modern version control system globally. There were several reasons why Git became the standard.

1. Git is distributed, meaning files and history are stored locally and in a central repository. This characteristic enables engineers to work without network access.
2. Git is fast. Since history is stored locally on your device, changing versions is nearly instant.
3. Git is secure. Git uses a hashing algorithm that ensures that every edit is traceable. It is impossible to change a file or directory without Git knowing.

How is Git Different from GitHub?

Git is an open-sourced, distributed version control tool. GitHub is a cloud-based platform built around Git. GitHub hosts a Git repository in the cloud, and is the most popular way people utilize Git source control. It started as a for-profit company, and Microsoft purchased the company in 2018. Other platforms built around Git include GitLab, Bitbucket (Atlassian), AWS CodeCommit, and Copia Automation. When we talk about the popularity of Git version control, we refer to the total users of the various Git providers and the open-source tool.

An important thing to note is that Git is often used via a command-line interface but has a simple graphical user interface (GUI). Most 3rd party Git providers - like GitHub and Copia, include a more comprehensive graphical user interface that simplifies Git tasks and, in the case of Copia, adds value when dealing with specific file types. (For example, Copia is tailor-built to show ladder logic and function block diagrams when working with industrial Automation files).

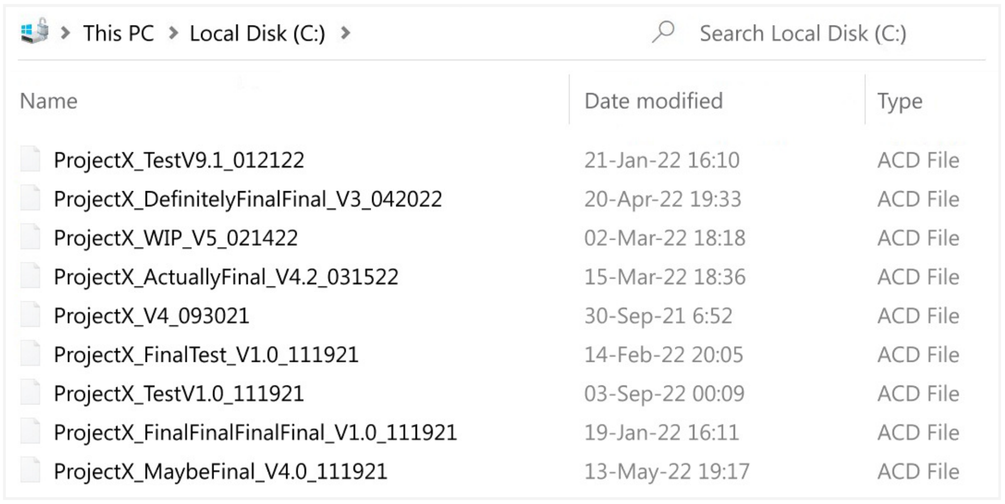
Today's Manual Process Is Prone To Errors

For an industrial automation professional who is not familiar with Git, it may be easiest to understand if you compare it to the widespread and manual practice of using an ***archive folder*** to manage industrial automation files.

The typical **Archive Folder Workflow** has the following steps:

1. The controls engineer creates project files on their local computer, using an installed Integrated Development Environment (IDE) (such as Rockwell Automation® Studio 5000 Logix Designer® or Siemens® TIA Portal). The names of the files are often a user created mix of a project name, version, and engineers initial (.i.e Mixer_DAH_V1.1)
2. Edits are stored by overwriting the previous file (Save), or copying and renaming the new files (Save As).
3. When work is completed to a significant state, the file or entire project folder may be zipped and copied to a central location for sharing and backup.
4. If another team member needs to access the files to review or make changes, they need to download them to their local hard drive and use their development environment to view and edit the files.

It is not uncommon for controls engineers to "Copy and Rename" to manage a file's history, resulting in a list of similar files distinguished by file names and modification dates.



This PC > Local Disk (C:) > Search Local Disk (C:)

Name	Date modified	Type
ProjectX_TestV9.1_012122	21-Jan-22 16:10	ACD File
ProjectX_DefinitelyFinalFinal_V3_042022	20-Apr-22 19:33	ACD File
ProjectX_WIP_V5_021422	02-Mar-22 18:18	ACD File
ProjectX_ActuallyFinal_V4.2_031522	15-Mar-22 18:36	ACD File
ProjectX_V4_093021	30-Sep-21 6:52	ACD File
ProjectX_FinalTest_V1.0_111921	14-Feb-22 20:05	ACD File
ProjectX_TestV1.0_111921	03-Sep-22 00:09	ACD File
ProjectX_FinalFinalFinalFinal_V1.0_111921	19-Jan-22 16:11	ACD File
ProjectX_MaybeFinal_V4.0_111921	13-May-22 19:17	ACD File

Prone to error, most organizations rely on file copies with file names to indicate which version is the latest.

Unfortunately, there are many problems and limitations with this workflow. For example:

- There is no inherent information on why a file was updated or changed. Additional work is needed to document and communicate changes.
- Project organization is based on manually naming files, which is prone to human error.
- There is no easy way to see the difference between file versions. Some IDEs provide this capability, but only for their specific file types.
- Local files are not backed up regularly. Work is often lost, and the latest version of a project may not be in the archive folder.
- Collaboration is limited. If teammates copy the same file from the central location and make changes, they cannot easily merge their work.
- It is difficult for managers to understand the progress made on a project since the work is usually kept on local machines.
- There is no inherent method for reviewing and approving files.
- Setting access permissions for specific files can be challenging.
- For large files, copying, pasting, and eventually uploading can take a significant amount of time. This fact may decrease the frequency at which projects are backed up.

Git alleviates these disadvantages. A basic Git-based workflow has many similarities to the **Archive Folder Workflow** workflow. Work is done locally, changes are saved and committed to the file's history, and then these committed changes are synchronized to a centralized location. There are significant benefits as Git removes the need to copy, move and rename files manually. Git stores versions of the project and provides rich context on who-what-and-why changes were made.

Git Glossary | Common Terms

Many people struggle with the Git terminology. Here is a quick glossary of terms to understand before discussing basic Git workflows:

Repository (Repo)

A "repository" or “repo” is simply a folder structure stored in Git. The "root" of a repository is the base folder you want to store. A repository is different from a typical directory only because it stores context about the changes to the root folder and all of the subfolders under the root.

Local Repository

A copy of the repository local to your workstation.

Remote Repository

A copy of the repository that is centrally located in the cloud or on a server. It is where you push changes for collaboration and backup.

Branch

A branch is a diversion from the main working project. By creating a new branch, the user can create a new version of the repo or experiment with changes that will not affect the base code. The most recent commit is considered the head of that branch. If you are happy with the changes in one branch, you can merge those changes into another branch.

Main (Master) Branch

Akin to the trunk of a tree from which all other branches start. The main branch is where the final, error-free code is stored. Every Git repository has a main branch. You can create new parallel branches off the main branch, empowering you to do work without affecting that mainline.

Clone

A copy of an existing repository. The clone can be made as a branch or downloaded to your local repository from a remote repository. You are required to create a clone to work on a repository.

Commit

A commit is a recorded change to a file or set of files. It is often thought of as a snapshot or version of your repository. Commits only create a snapshot in your local repository. To synchronize these changes to the remote repository, Push.

Pull Request (PR)

A pull request (PR) occurs when you alert others about or request a review of a change you’ve pushed to a remote repository. The changes can be discussed, reviewed, and commented on, with follow-up commits added before the changes are merged into the main branch. The intent of the pull request is to merge changes into the main branch.

Push

A push is a command used to add your commits from your local repository to a remote repository. A push is the opposite of a fetch.

Merge

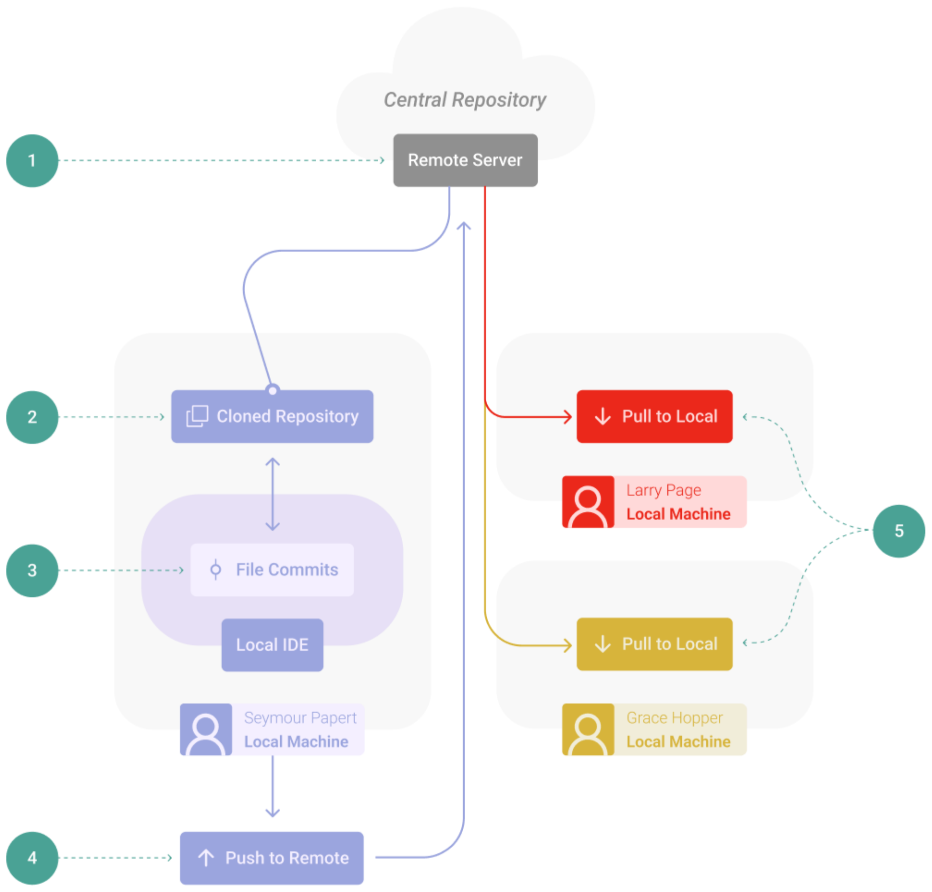
Merging combines two branches. Typically, commits made to a branch are merged into the main branch after being reviewed via a pull request. In some organizations, a project maintainer or manager is responsible for approving merges.

The Basic Git Workflow

Here are the steps of a **Basic Git Workflow** for an Industrial Automation project:

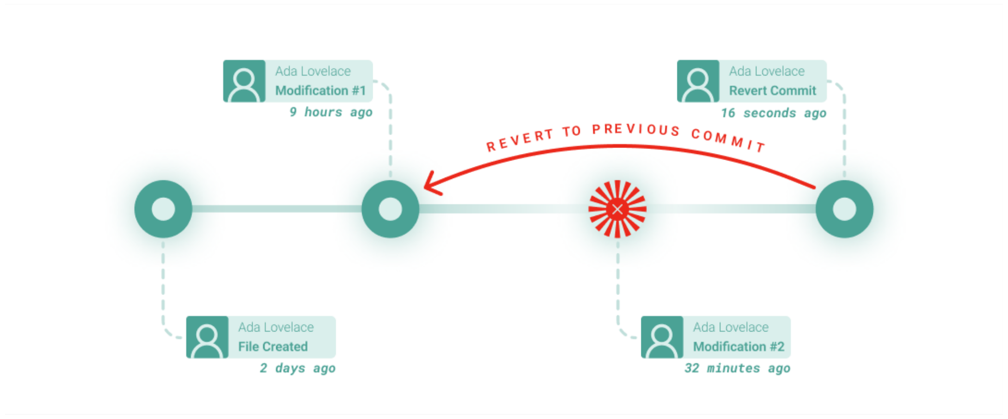
1. A central repository is created on a server. This server will be cloud-hosted for most Git providers (GitHub, Copia, etc.).
2. The repository is then cloned to the control engineer's local machine. It will appear on the local PC as a standard Microsoft® Windows® folder.
3. The engineer creates their automation files using their local IDE (such as Rockwell Automation Studio 5000 Logix Designer), saves the file in the local repository, and commits these changes to the file history when ready.
4. When a development milestone is reached, or the engineer believes it is appropriate, the engineer will push their committed changes to the central repository.
5. Meanwhile, teammates who have also cloned the central repository locally can "pull" the updated files to their local repository so that they are always working with the latest files.

You can see these steps outlined in the diagram below:



In a Git workflow, file names stay the same, and commit history is automatically tracked. A simple set of pull and push commands ensure that everyone has access and is working with the latest version.

A PLC file's history can be viewed using a Git commit graph. Notice that any previous commit can be retreived if needed.



With Git, each committed change is stored with context and can be visualized as a node along a main branch of code. A simple revert command can be used to access previous changes.

There are some subtle things worth mentioning:

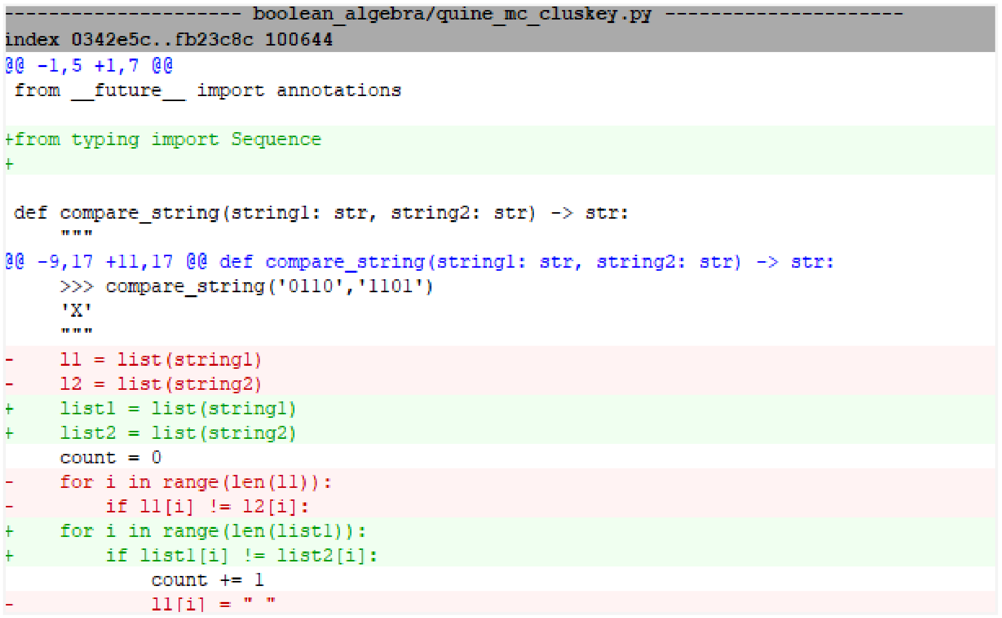
- As changes are made, file names can stay the same in Git. There is no reason for the engineer to have to use the file name to describe the state of the project (i.e. Apex_labeler_DAH_Final.ACD). Git tracks the difference in each commit for you.
- Tasks like creating and cloning repositories, committing, pushing, and pulling are fast and usually only take a few mouse clicks. The Git workflow is easy to execute.
- Git never deletes or overwrites files, so you can always access your historical work if needed. If you accidentally removed a rung and saved the file, you can simply revert to an earlier version. It's like a post-save undo!
- A persistent internet connection is not required. You can work locally and then push changes at a later time. This is quite helpful when making code changes on field-based devices.

Why Hasn't Git Been Widely Adopted by PLC Programmers (Yet)?

We mentioned that a remarkable benefit of Git version control is tracking when files change and showing what changed.

Git does this by displaying the contents of file versions and highlighting the differences in a diffing process. Diffing is relatively fast and straightforward because most traditional software development uses text-based programming languages (including Python, JavaScript, Java, C#, C, and C+ +).

An example of a Git “diff” is provided below. The red items indicate lines of deleted code, and the green items represent new code that has been added.

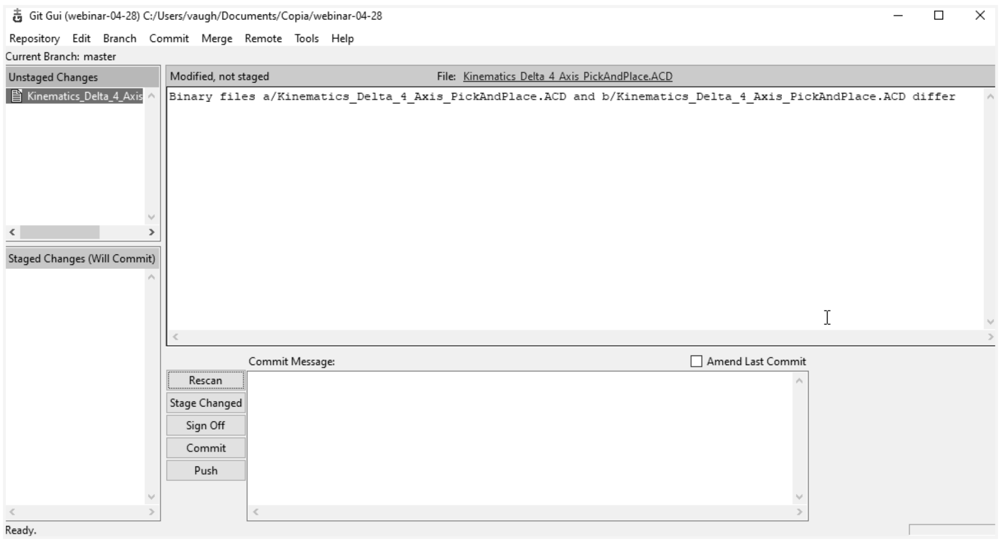


```
-------------------- boolean_algebra/quine_mc_cluskey.py --------------------
index 0342e5c..fb23c8c 100644
@@ -1,5 +1,7 @@
 from __future__ import annotations

+from typing import Sequence
+

 def compare_string(string1: str, string2: str) -> str:
     """
@@ -9,17 +11,17 @@ def compare_string(string1: str, string2: str) -> str:
     >>> compare_string('0110','1101')
     'X'
     """
-    l1 = list(string1)
-    l2 = list(string2)
+    list1 = list(string1)
+    list2 = list(string2)
     count = 0
-    for i in range(len(l1)):
-        if l1[i] != l2[i]:
+    for i in range(len(list1)):
+        if list1[i] != list2[i]:
             count += 1
-            l1[i] = " "
```

Git visually displays the changes between commits in text-based files. Deleted code is shown in red, while additions are shown in green.

Unfortunately, PLC programming evolved quite differently than traditional software programming. While there are some text-based languages for PLC Programming, most are done in visual languages like ***Ladder Logic*** and ***Function Block Diagrams***. This problem was compounded because many PLC vendors use different binary file formats. The inability of standard Git to display these languages reduced much of its value for many controls engineers.

So to be clear, for most industrial controls projects, Git can tell you when and who changed files but not show you how those files changed. The lack of this significant benefit has slowed Git adoption.


Git Gui (webinar-04-28) C:/Users/vaugh/Documents/Copia/webinar-04-28
Repository Edit Branch Commit Merge Remote Tools Help
Current Branch: master
Unstaged Changes
Kinematics_Delta_4_Axis
Staged Changes (Will Commit)
Modified, not staged
File: Kinematics_Delta_4_Axis_PickAndPlace.ACD
Commit Message:
Amend Last Commit
Rescan
Stage Changed
Sign Off
Commit
Push
Ready.

```
Binary files a/Kinematics_Delta_4_Axis_PickAndPlace.ACD and b/Kinematics_Delta_4_Axis_PickAndPlace.ACD differ
```

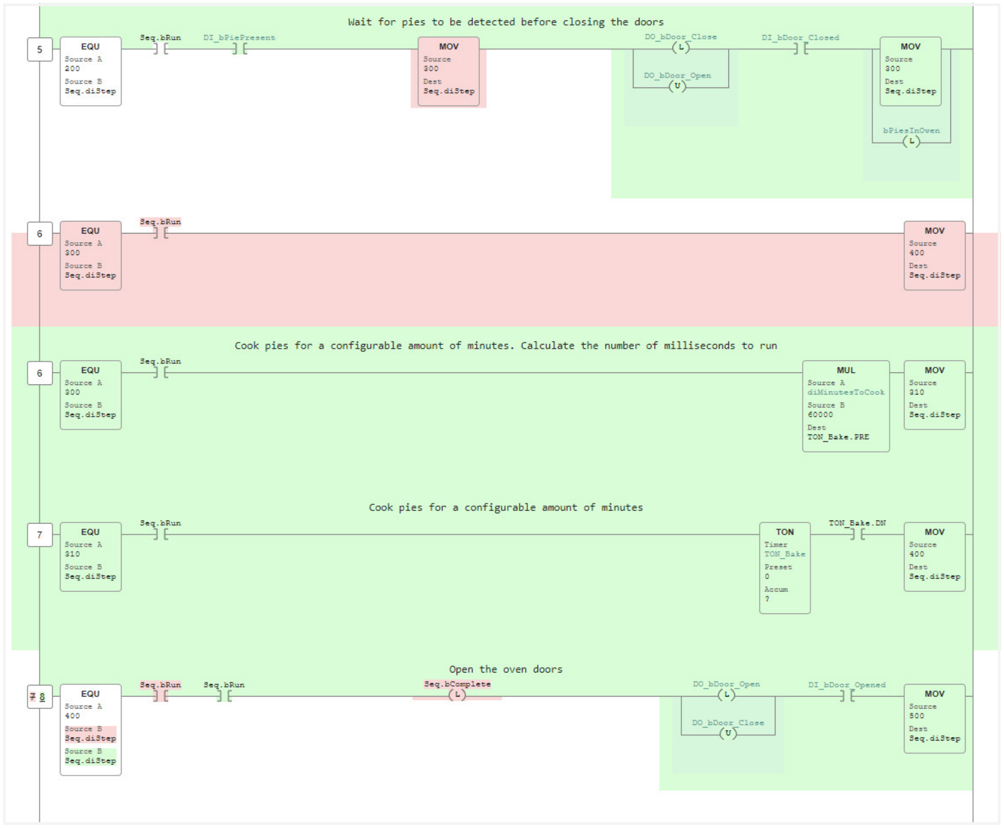
Git relates that PLC files have changed, but does not visually show differences between file states.

Copia Automation to the Rescue

Copia Automation was founded to bring modern developer tools to industrial automation professionals, unlocking the productivity gains already realized in traditional software development. They have started by solving the issue around visualizing and diffing PLC code changes when using Git-based source control.

Copia toolsets empower engineers to visualize PLC code in ladder logic, function block diagrams, and structured text languages. Copia can visualize code from Rockwell Automation® Studio 5000 Logix Designer®, Siemens® TIA Portal, ABB® Automation Builder, Beckhoff® TwinCAT®, Lenze® PLC Designer, Wago® e!COCKPIT, CODESYS®, and additional vendor support. Teams can follow a consistent workflow no matter what PLC vendor they choose.

It is essential to understand that Copia renders the PLC code in its desktop app and web browser. This capability provides incredible freedom to automation teams and accelerates code review and discussions. Consider a junior engineer developing a section of code that controls machine safety and finishes their task late in the day. With only a web link, a manager who needs to review the code can securely log in to the Copia repository from a home computer and see the latest changes directly in a web browser.



Copia visually displays changes between commits to ladder logic files. Deleted rungs are displayed in red, and additions are shown in green.

Advanced Git Workflows to Unlock Greater Value

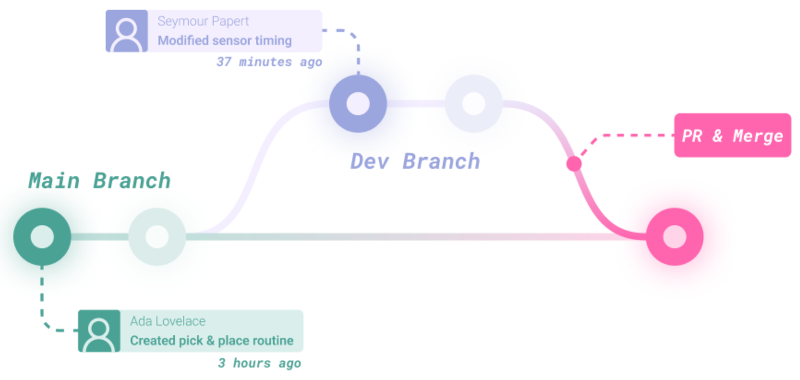
Although we described a basic Git workflow, Git supports advanced workflows that add more control and improve collaboration. These workflows are centered around concepts known as ***Branching*** and ***Merging***.

An easy way to understand the concept of branching is to envision that every development project has a main branch where the final error-free code is stored. Every commit represents a vetted change that purposely improves the code.

Git allows you to create a parallel branch from the main branch when adding new features and creating bug fixes. This branch enables you to make changes without disturbing the main branch. If your code changes are successful, you can merge the changes back into the master branch; if they are not successful, you can delete them.

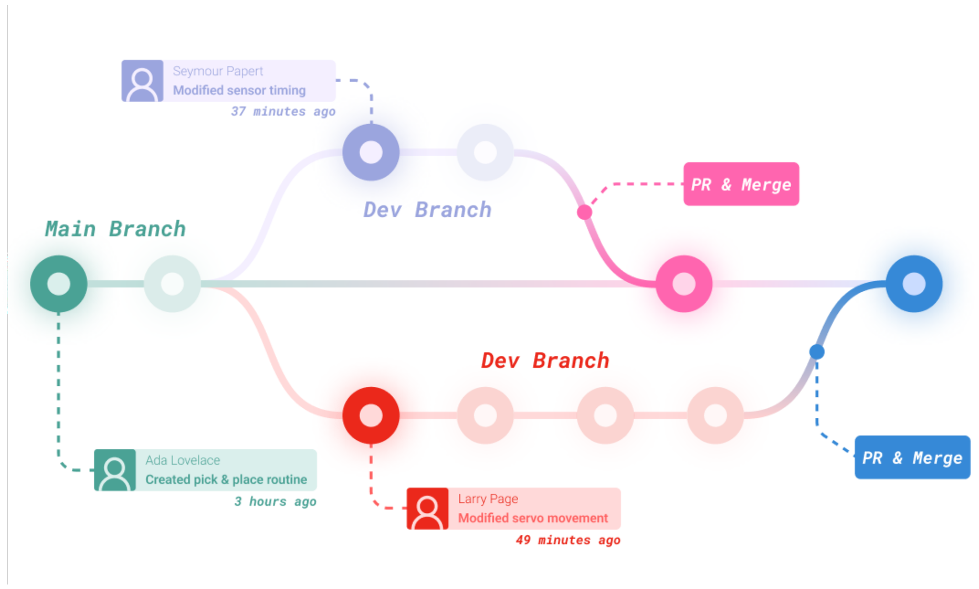
Using this branching and merging process gives your team greater control over when the main branch is changed. For example, rules can be developed and enforced so only project leads can merge code into the main branch, ensuring that all changes are reviewed and approved.

Another benefit of this workflow is that it keeps the project development history clean and easy to understand. Significant changes to the production code are documented in the main branch, while work in progress is tracked and stored within development branches.



Git branching can ensure that any work in progress (development branches) can be completed and reviewed before merging with the production code (main branch).

One of the most powerful aspects of Branching and Merging is the ability for multiple developers to work on the same project simultaneously. Each developer can create individual development branches, and when their work is complete, they can use the merge command to stitch their work together into the main branch. Branch and merging allow you to add more engineers to a job to meet tight deadlines.



Development branches can exist simultaneously, allowing multiple engineers to work together on the same code. Merging can combine all work together into the main branch.

You might ask, *"What happens if two engineers change the same line of code and then merge - and which change will be accepted?"*

Git handles such situations with tools to resolve merge conflicts. The project lead can see both changes and choose the better one. This ability is another reason why visualizing differences between commits is so essential during the code review process.

What is a Pull Request?

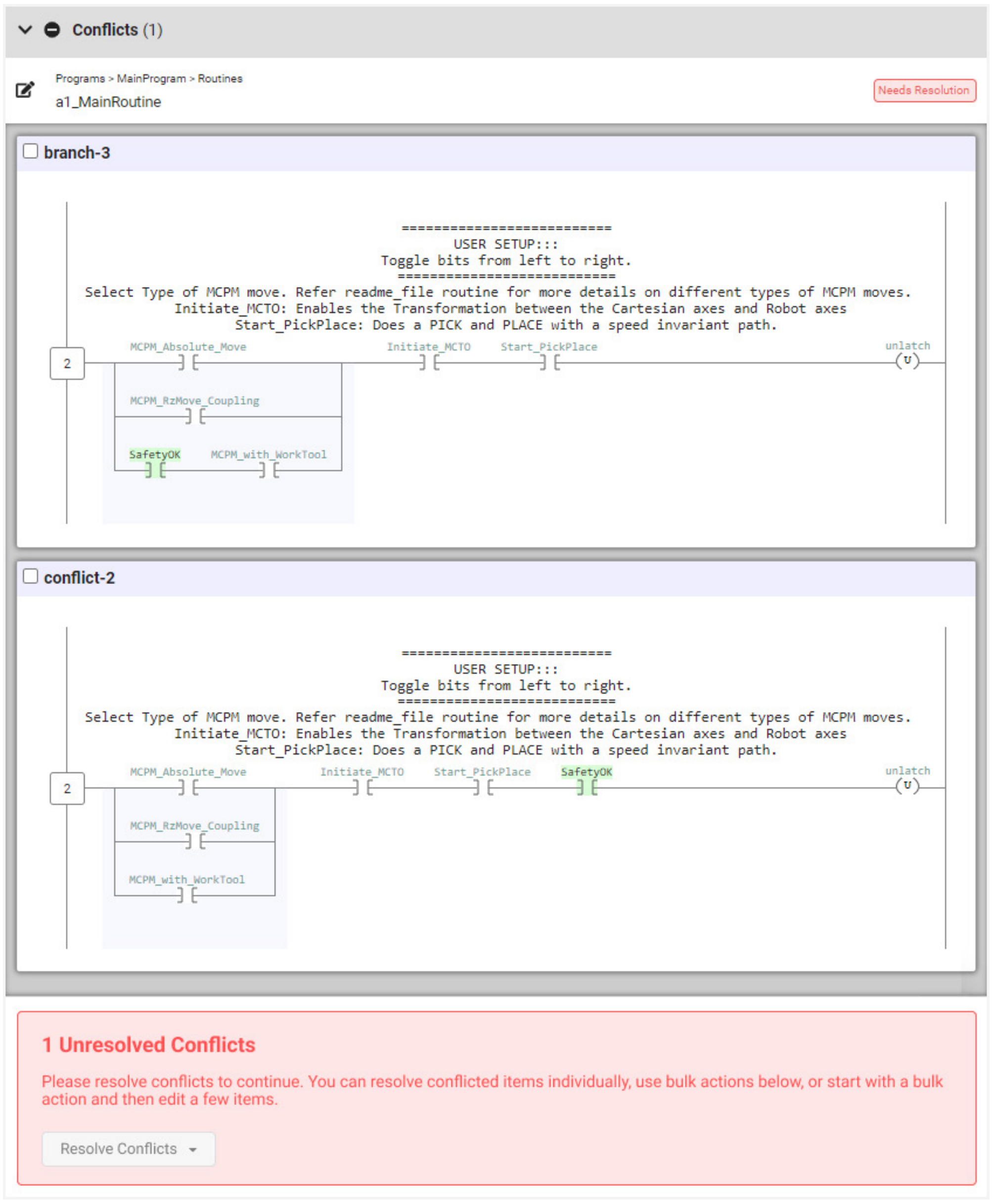
A service call with a PLC programmer is needed after machine commissioning. The programmer can create a branch and work on code. Upon completion, they can notify the project lead that the branch is ready for review and merge. To do this, they create a ***pull request***.

A pull request occurs when you alert others about or request a review of a change you've pushed to a remote repository. The changes can be discussed, reviewed, and commented on, with follow-up commits added before the changes are merged into the main branch. The intent of the pull request is to call for a merge into the main branch.

Branching and Merging, Tailor-Built for Industrial Automation

We have stated that Copia Automation provides specific tools to visualize ladder logic. The same visualization is used when handling merge conflicts. Copia will display the rungs in questions and prompt the decision-maker to choose which change is preferred if multiple engineers change the exact area of code.

Copia understands that thorough code review, careful pull request approval, and thoughtful merge conflict resolution are essential for producing the highest quality code. Copia allows teams to add comments at the rung level to discuss and document decisions during a pull request. This context, captured throughout a project's lifecycle, can be utilized to train new programmers, ensure consistency, and identify opportunities to improve.


Conflicts (1)

Programs > MainProgram > Routines
a1_MainRoutine

Needs Resolution

branch-3

```
==========================
USER SETUP:::
Toggle bits from left to right.
============================
Select Type of MCPM move. Refer readme_file routine for more details on different types of MCPM moves.
Initiate_MCTO: Enables the Transformation between the Cartesian axes and Robot axes
Start_PickPlace: Does a PICK and PLACE with a speed invariant path.
```

conflict-2

```
==========================
USER SETUP:::
Toggle bits from left to right.
============================
Select Type of MCPM move. Refer readme_file routine for more details on different types of MCPM moves.
Initiate_MCTO: Enables the Transformation between the Cartesian axes and Robot axes
Start_PickPlace: Does a PICK and PLACE with a speed invariant path.
```

1 Unresolved Conflicts

Please resolve conflicts to continue. You can resolve conflicted items individually, use bulk actions below, or start with a bulk action and then edit a few items.

If a conflict is detected during a merge, Copia will display the conflicting code and prompt the user to choose a resolution.

How Git-Based Version Control Improves Business

Hopefully, at this point, you have a clear understanding of Git-based version control and the features that are meaningful to industrial automation professionals, specifically PLC programmers. When we talk about the value to a business, the primary benefit is centered around employee productivity and shortening product timelines. With Git-based source control implemented, all individuals will spend less time searching for the files and investigating how files differ during their lifecycle. That time savings can be reinvested in high-value work developing innovative and high-quality code.

Branching and merging enable multiple team members to work on the same automation project simultaneously. This practice has the potential to be a substantial competitive advantage for a company and ensure tight project deadlines are met.

The increased collaboration of Git-based version control also enables the business to utilize their most skilled people more efficiently. Senior control engineers can quickly review more junior engineers' work continuously via a web app and document their feedback to help accelerate team training.

Improved code quality is another significant benefit of a Git-based version control system. Using visual diffing capabilities allows errors to be detected more readily. Formal pull request procedures ensure that only authorized people can change the production code.

Finally, a solid Git-based version control system can save a business thousands when dealing with unexpected operational problems. If a major incident disrupts manufacturing, the last good version of the code can always be found quickly and used to restore service.

Summary and Next Steps

Git is the ubiquitous source control solution for software development, and its use has accelerated the speed at which code is developed and deployed. It is proven to shorten development timelines, increase quality, and maximize operational uptime.

While the visual languages and proprietary formats of PLC code have kept industrial automation developers from realizing the same gains, Copia Automation has made tremendous strides to remove these challenges. Now, ladder logic and function block diagrams are supported and can be visualized outside of their development environments. While saving and storing code is slightly different from the traditional archive folder workflow, Git-based source control for industrial automation projects is easy to learn and worth the benefits.

It's easy to get started with Git. You can find more information and choose to download the open-source version of Git from https://git-scm.com or try a free version of GitHub at https://github.com/. If you are considering Git for your PLC programming projects, we strongly recommend requesting a demo of Copia Automation at www.copia.io.

COPIA

copia.io