CI/CD Pipelines and Terminologies (GitHub & Matillion DPC)

This document explains Continuous Integration and Continuous Deployment (CI/CD) pipelines, their components, and key terminologies with respect to GitHub and Matillion. It also outlines how environments (Dev, Test, Prod) fit into the workflow.

1. What is CI/CD?

CI/CD stands for Continuous Integration and Continuous Deployment (or Delivery). It is a modern DevOps practice used to automate the process of building, testing, and deploying applications. The main goals are:
- Faster delivery of features
- Early detection of bugs
- Reliable deployments across environments (Dev → Test → Prod)

2. CI/CD Pipeline Components

A pipeline typically has these stages:

- Source Code Management: Code is stored in GitHub repositories.

- Build: Code is compiled and packaged.

- Test: Automated tests run to validate code quality.

- Deploy: Application is deployed to environments (Dev, Test, Prod).

- Monitor: System health and logs are monitored after deployment.

3. Git/GitHub Terminologies

- Commit: Saving changes to the local repository with a message describing the changes.

A commit captures a snapshot of currently staged changes to the Git integration project.

- Push: Uploading local commits to the remote repository on GitHub.

- Pull: Fetching and merging changes from the remote repository to the local repository.

- Pull Request (PR): A request to merge changes from one branch into another (usually feature → main).

- Merge: Combining changes from one branch into another branch.

- Branch: A separate line of development (e.g., feature, bugfix, dev_env, test_env, prod_env).

- Compare Changes: Reviewing differences between two commits or branches before merging.

4. Matillion in CI/CD

Matillion is used for orchestrating and automating deployments. It integrates with GitHub to deploy applications into different environments. Its purpose is to standardize deployment steps and reduce human errors.

When and Why We Use Matillion:

- Dev Environment (Dev_env): For developers to test new features. Matillion deploys builds automatically after each commit or merge.

- Test Environment (Test_env): Used by QA teams for testing. Matillion ensures only tested code moves here.

- Prod Environment (Prod_env): The live system. Matillion enforces approval workflows (manual review, pull request merges) before deploying here.

When using the Git version control feature in Matillion ETL, it'll help to understand the underlying architecture and concomitant technical terms. There are six components involved:

1. A Matillion ETL project. The project is the top-level structure containing jobs and other collateral within Matillion ETL. Each project is isolated, and user access can be granted or denied on a per-project basis.

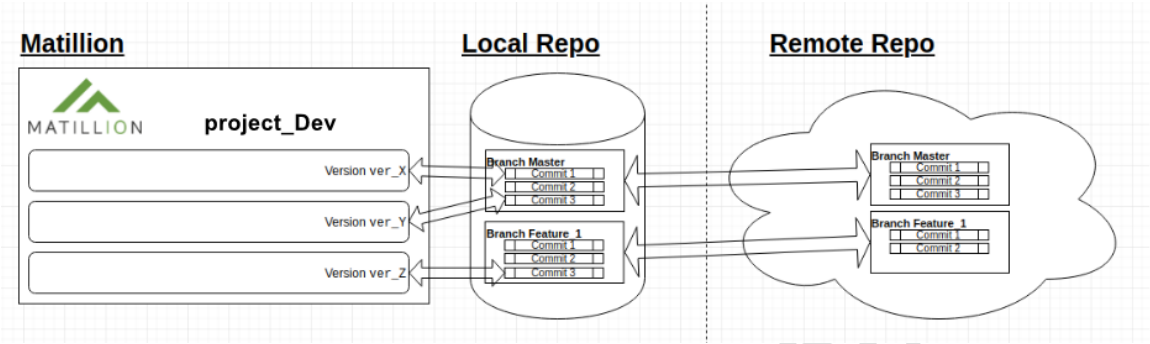
2. A Matillion ETL version. A project can contain more than one version. When used with Git, think of a version as an independent working area. Each version points to a single Git commit in the local Git repository.

3. The local Git repository (repo). The local repository stores files on the Matillion ETL instance's filesystem, which is created automatically when a project is Git-enabled.

4. The remote Git repository (repo). A self-hosted or cloud-hosted Git repository that is *external* to Matillion ETL, and which you must set up *prior* to Git-enablement in Matillion ETL. Users can push local repository commits to their remote repository. Users can also fetch newer commits from the remote repository into the local repository.

5. Commit. A commit is a point-in-time copy of a Matillion ETL version, typically with collateral stored in the version, such as orchestration and transformation jobs.

6. Branch. A branch is a collection of one or more commits in Git. Typically, a Git project will have a master branch, from which other branches will be created to develop and test code. A branch model typically allows users to develop new code without adding questionable code to the master branch. Once the code has undergone testing, it can be merged safely into the master branch.



The above diagram includes an example project, project_Dev. Within this project are three separate versions. Each of these versions might, for example, belong to an individual developer in a development team.

Shown to the right of the project is the local repository, which contains two branches. There is the master branch (**Branch Master**), and an additional branch (**Branch Feature_1**). Within each of these branches are three commits, and the diagram shows via the shorter white arrows which version is pointing at which commit.

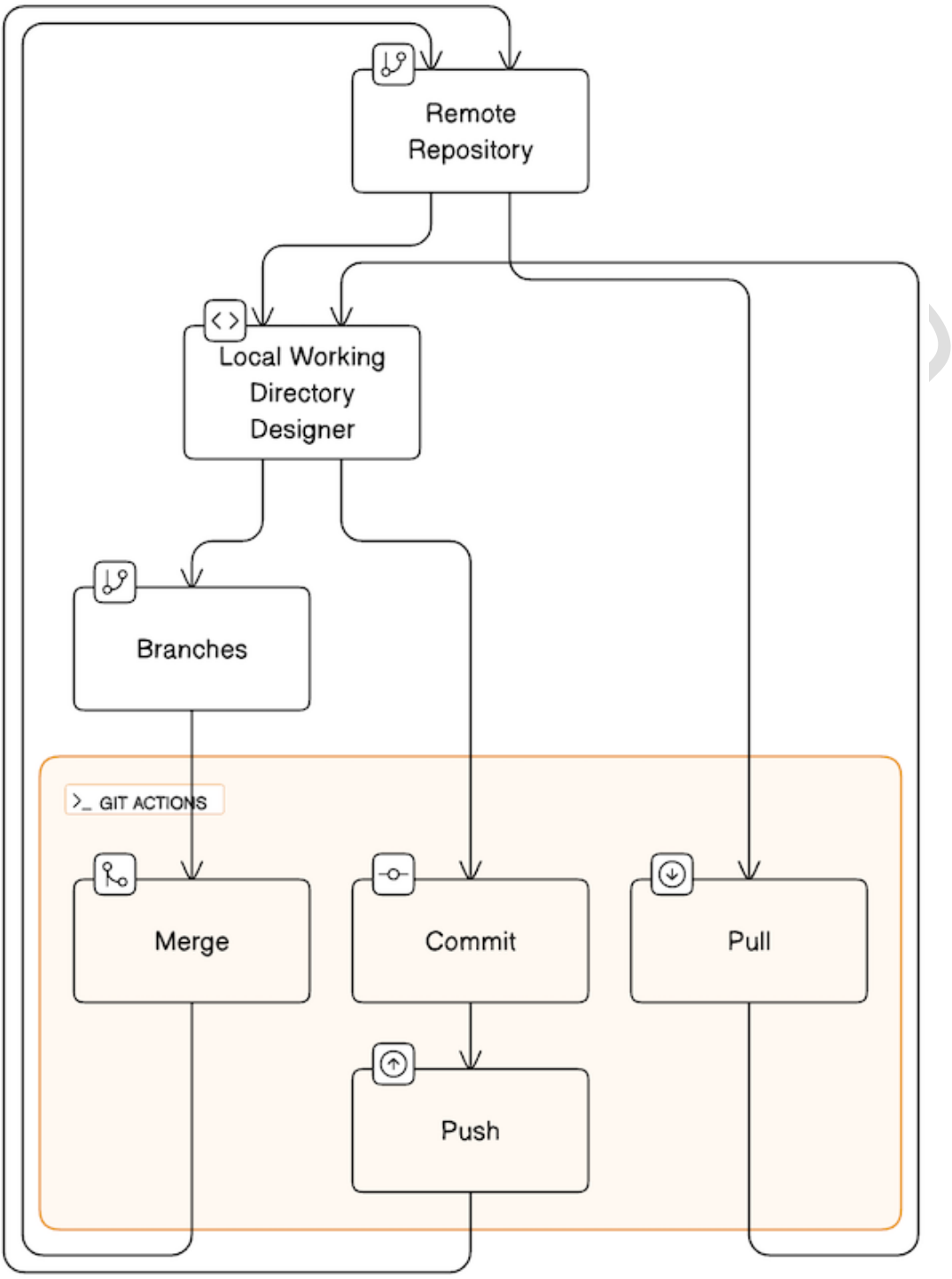
Shown to the right of the local repository is the remote repository, which contains a backup copy of local repository. However, note that the remote repository is missing Commit 3 from Branch Feature_1. This simply means that the local repository's changes require a push to the remote repository, at which point Commit 3 of Branch Feature_1, which is being developed in Version ver_z, will be backed up in the cloud-hosted remote repository.

Note

Matillion ETL's Git integration feature does not support multi-factor authentication (MFA) at this time.

5. Example Workflow

1. Developer makes code changes locally.
2. Developer commits changes with a clear message.
3. Developer pushes changes to GitHub.
4. Developer creates a Pull Request to merge feature branch → dev_env branch.
5. Code is reviewed and merged.
6. Matillion automatically deploys the updated dev_env branch into the Dev environment.
7. After testing, changes are merged into test_env branch → Matillion deploys to Test.
8. Once approved, changes are merged into prod_env → Matillion deploys to Production.



6. Summary

In summary, GitHub handles version control and collaboration, while Matillion automates deployments into environments (Dev, Test, Prod). CI/CD pipelines connect these steps to ensure code is continuously built, tested, and deployed reliably.