# Introduction

**What is Github?**

GitHub is a developer platform that allows developers to create, store, manage and share their code. It uses Git software, providing the distributed version control of Git plus access control, bug tracking, software feature requests, task management and continuous integration for every project.

GitHub offers a wide range of features that facilitate software development and project management. **Here are some of its key features:**

1. **Version Control**: GitHub is built around Git, a distributed version control system, allowing developers to track changes to their codebase, manage different versions, and collaborate seamlessly.

2. **Repository Hosting**: Users can create repositories to store their code and related resources. Repositories can be public or private, and they can contain any type of file, including documentation, images, and more.

3. **Collaboration Tools**: GitHub provides a range of tools to facilitate collaboration among developers. These include pull requests, which allow contributors to propose changes and discuss them before merging into the main codebase, and issues, which can be used to track bugs, feature requests, and other tasks.

4. **Branching and Merging**: Developers can create branches to work on new features or experiments without affecting the main codebase. GitHub provides tools for merging changes between branches and resolving conflicts that may arise.

5. **Code Review**: GitHub supports code review workflows, allowing developers to review each other's code, provide feedback, and ensure code quality before merging changes into the main branch.

6. **Continuous Integration/Continuous Deployment (CI/CD)**: GitHub integrates with various CI/CD services, allowing developers to automate build, test, and deployment processes. This helps maintain code quality and streamline the release process.

7. **Project Management**: GitHub includes project boards, which enable teams to organize and prioritize tasks using customizable columns and cards. Users can create tasks, assign them to team members, and track their progress.

8. **Wiki and Documentation**: Repositories on GitHub can include wikis for documentation and other resources. This allows teams to maintain documentation alongside their codebase and keep it up to date.

9. **Social Features**: GitHub has social features such as following other users, starring repositories, and forking repositories (creating a copy of a repository under one's own GitHub account). These features encourage collaboration and knowledge sharing within the developer community.

10. **API and Integrations**: GitHub provides a comprehensive API that allows developers to integrate GitHub functionality into their own applications and workflows. It also integrates with various third-party services and development tools.

Overall, GitHub's feature set makes it a versatile platform for version control, collaboration, and project management, widely used by individual developers, open-source projects, and large enterprises alike.

**Github code versioning and management:**

A screenshot of a video game

Description automatically generated

As shown in above figure,

* Multiple developers (Dev1, Dev2 and Dev3) can work on respective features (Feature1, Feature2 and Feature3) development.
* Once, the respective development, unit testing and QA testing are completed for the features developed by the developers, then the respective features code will be committed.
* Now, a pull request will be raised for each feature to merge the code into development branch which will be reviewed and approved by approver, after which the features get merged into development branch.
* After this, a Release branch v1 will be created from development branch which has all features included in it and will be tested by QA team for the expected results as per the features.
* Once, QA testing is successful on Release branch, it will be pushed to QA environment, code will be executed and will be taking signoff from Business users once expected results are observed.
* After the Business user’s signoff on QA, Release branch will be pushed to prod environment, code will be executed and expected results will be checked by End users and signoff will be provided for the whole deployment.
* After the End user’s signoff in production, Release branch which is present in prod will be merged to Master branch which has the base code which is working in production always.
* If there are any deviations in the expected results in any of QA or Prod environment, code will be rolled back to previous version which was active in the respective environment.

Above explained is the process which is followed usually in any of the organizations which use Github for their code management/versioning.

**Scenarios:**

**Use Case-1 From the features Feature 1 (here after referred as F1), Feature 2 (here after referred as F2) and Feature 3 (here after referred as F3), if F2 fails in QA how it handles?**

In this scenario, F2 will be fixed again (basis on the bug raised in QA) in Development environment, and the latest code of F2 will be committed, merged again into development branch along with F1 and F3 and then another Release branch v1.1 will be created and moved to QA for testing the features again and move it further based on the successful execution of the code.

**Use Case-2. If Developer 1 is working on both F1 and F4 and F4 is a priority, how is it handled?**

In this scenario, only the features that are needed for deploying into further environments will be merged into the development branch and then Release branch will be created and process further.

**Use Case-3 In Release v1, we have F1, F2, F3 and there was high priority for F4 and only F4 should go to prod, how it handles?**

In this scenario, F4 can be taken as hotfix/priority deployment and a separate Release branch v2 will be created for this which has only F4 and will be processed further.

As discussed over the call , we share the below information over the zoom chat:

1. GCS knowledge base article to setup the Github actions workflow .

[Automated Deployment of IICS Assets- CI/CD using Informatica API's](https://knowledge.informatica.com/s/article/Automated-Deployment-of-IICS-Assets-CI-CD-using-Informatica-API-s?language=en_US&type=external)

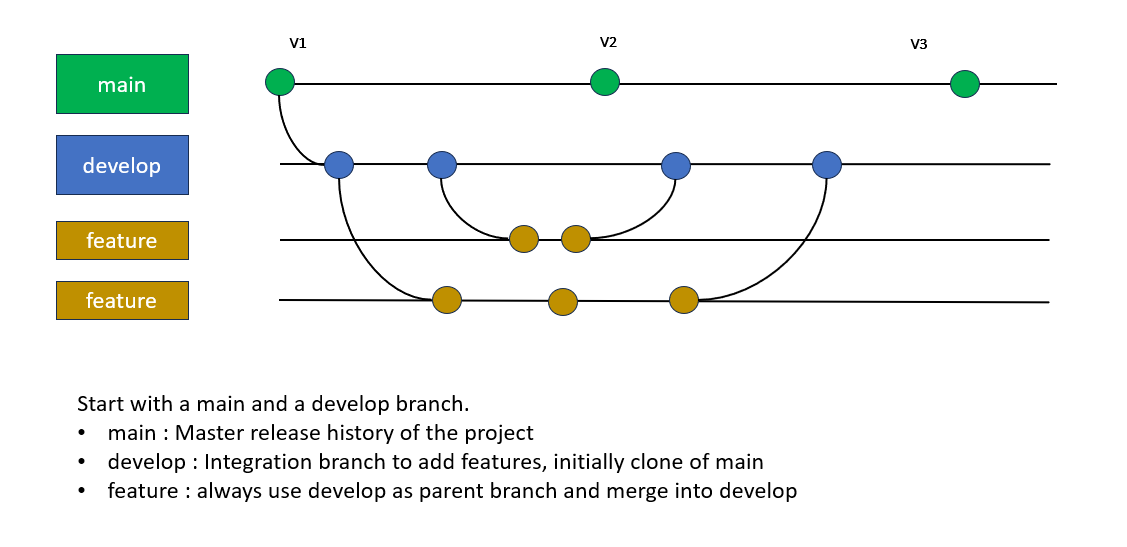
1. Clarification on April 2024 release changes on versioning.
2. Additional questions on how IICS CICD process.

# Understanding Versioning and release set-up

## Types of Branches

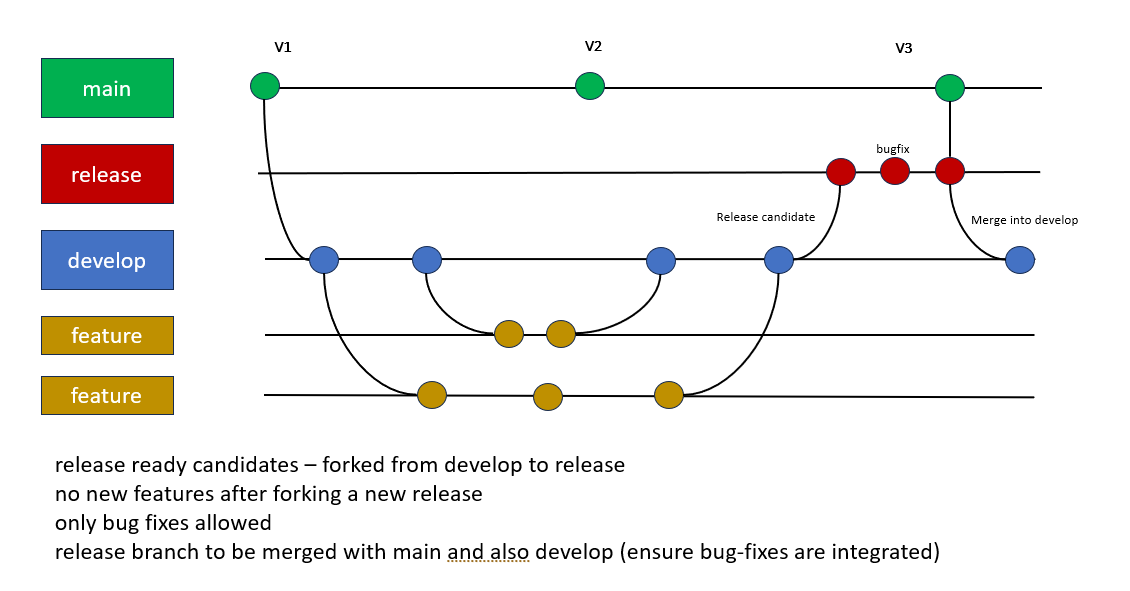
1. main
2. develop
3. feature
4. release
5. hotfix

## Continuous Integration

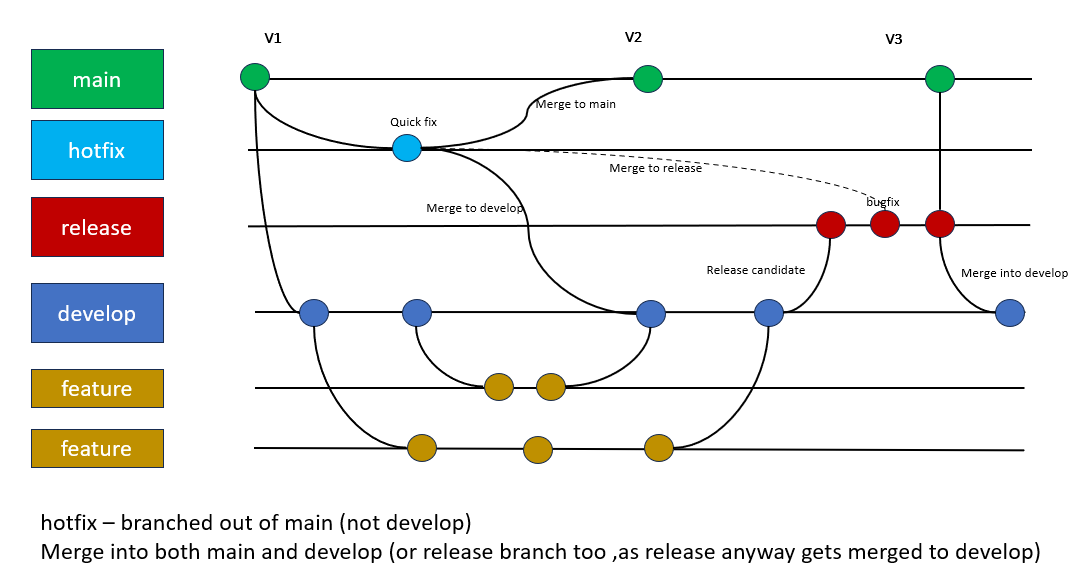


## Continuous Deployment

### Release



### Hotfix



## Knowhow

*Q1: When will develop, branch be merged to main branch.*

The develop branch is typically merged into the main branch when the code in the develop branch reaches a stable point and is ready for release. There are different merge options namely Manual Merge (most preferred and conventional), Fast-forward merge.

*Q2: How to selectively merge prioritized features to the develop branch?*

Work on Each Feature Separately. Merge Specific Features into Develop. Since you’re merging specific features, any features not explicitly merged will remain in their respective feature branches.

*Q3: How to reset the develop branch – to ignore certain feature commits?*

Identify the commit hash of the feature and reset the develop branch to the prior commit point.

# GitHub Actions

## Define the Deployment

Each organization could customize the deployment strategy. It involves defining the each of the components below and defining the dependencies to push the code across environments typically involving Dev 🡪 QA 🡪 UAT 🡪 Prod.

## Components

### Workflow

### Events

### Jobs

### Actions

### Runners

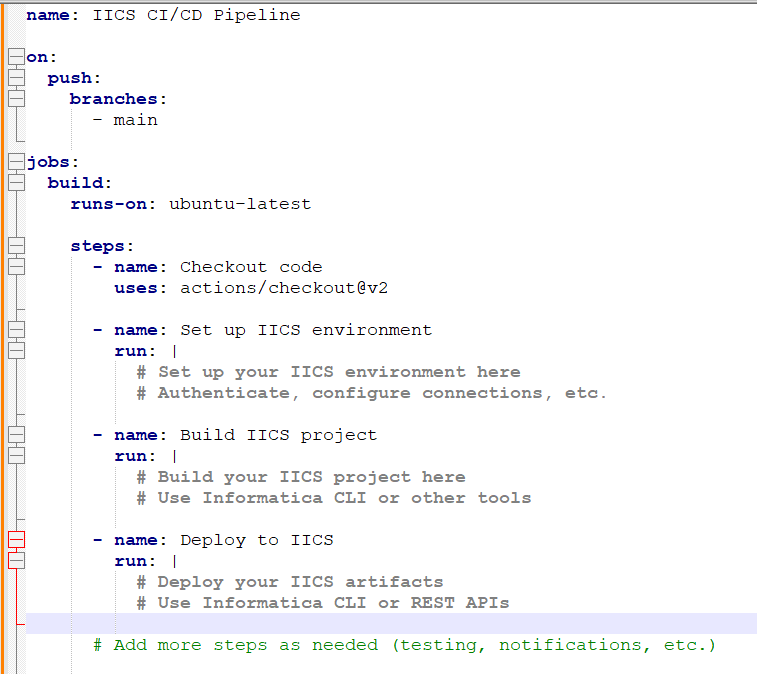
## Define the ‘YAML’

To set up a CI/CD pipeline for Informatica Intelligent Cloud Services (IICS) using GitHub Actions, you’ll need to create a YAML file that defines your workflow.

Location within IICS project directory: .github/workflows/ci-cd.yml

### Sample YAML

#### Example 1



#### Example 2

