|  |
| --- |
| **“Expert Cloud Consulting” -**  **Git & GitHub branching strategies**  09.September.2025  Contributed by M Bindu Sri Santhi  Approved by Rushi wagh(In Review)  Expert Cloud Consulting  Office #333, Gera Imperium Rise,  Hinjewadi Phase-II Rd, Pune, India – 411057 |

“Expert Cloud Consulting”

Git & GitHub branching strategies

### 1.0 Contents

Contents

[1.0 Contents 1](#_Toc208309531)

[2.0 General Information: 2](#_Toc208309532)

[3.0 Document Overview: 4](#_Toc208309533)

[4.0 Steps / Procedure 5](#_Toc208309534)

[5.0 Annexure - I [ Roman Numbers] | [ heading of content] 11](#_Toc208309535)

****

### 

### 2.0 General Information:

#### 2.1 Document

|  |  |
| --- | --- |
| **Ticket(s) Name** | **Url** |
| Branchig strategies | <https://medium.com/@sreekanth.thummala/choosing-the-right-git-branching-strategy-a-comparative-analysis-f5e635443423> |

#### 2.2 Document Purpose

The document covers creating a GitHub organization and repository, setting up multiple branches (main, develop, QA, and UAT), and demonstrating collaborative development using feature branches. It also includes handling merge conflicts, raising pull requests, and ensuring a smooth flow of code from development to testing and staging branches

#### 2.3 Document Revisions

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Date** | **Version** | **Contributor(s)** | **Approver(s)** | **Section(s)** | **Change(s)** |
| 09/Sep/2025 | 1.0 | M Bindu | Rushi wagh | All Sections | New Document Created |
|  |  |  |  |  |  |

#### 

#### 2.4 Document References

The following artifacts are referenced within this document. Please refer to the original documents for additional information.

|  |  |  |
| --- | --- | --- |
| **Date** | **Document** | **Filename / Url** |
| 2021 | Branching strategies | <https://medium.com/@sreekanth.thummala/choosing-the-right-git-branching-strategy-a-comparative-analysis-f5e635443423> |
| 2020 | Merge conflicts | <https://www.youtube.com/watch?v=SJLgI9BfhPY> |

### 

### 3.0 Document Overview:

This document explains the implementation of a Git branching strategy to simulate a real DevOps workflow. It covers creating a GitHub organization and repository, setting up branches (main, develop, QA, UAT), working with feature branches, resolving merge conflicts, and merging changes. The goal is to demonstrate collaboration, version control, and release management practices in a structured way.

### 

### 4.0 Steps / Procedure

Git is a free and open-source distributed version control system (VCS). It is a local software that tracks changes in source code during software development. Git enables multiple developers to work on the same project simultaneously without overwriting each other's work by managing different versions of files and allowing for collaboration, branching, merging, and reverting to previous states of a project.

Git Branching Strategy:

Common commands used with git:

**Basic**

git init Initialize a new Git repository

git clone <repo> Clone a repository from remote

git status Show working tree status

git add <file> Stage file(s) for commit

git commit -m "msg" Commit staged changes with a message

git log View commit history/logs

git diff Show changes not yet staged/committed

git branch List all branches

git branch <name> Create a new branch

git checkout <branch> Switch to a branch

git merge <branch> Merge a branch into the current branch

git rebase <branch> Reapply commits on top of another branch

git remote -v Show configured remote repositories

git push origin <branch> Push local branch to remote

git pull origin <branch> Pull latest changes from remote branch

#### 4.1 : Creating GitHub Organization

**Team Collaboration** – An organization lets multiple people (developers, testers, DevOps, managers) work together under a common space.

**Centralized Management** – Instead of repos belonging to just one person’s account, they belong to the organization → safer if someone leaves the team.

**Access Control** – You can give different roles (Admin, Maintainer, Developer, Read-only) to team members.

**Project Separation** – You can group multiple repositories (frontend, backend, infra, CI/CD, docs) under one organization.

**Professional Setup** – Companies prefer an organization account (e.g., my-company) instead of using a developer’s personal GitHub.

How to create organization :

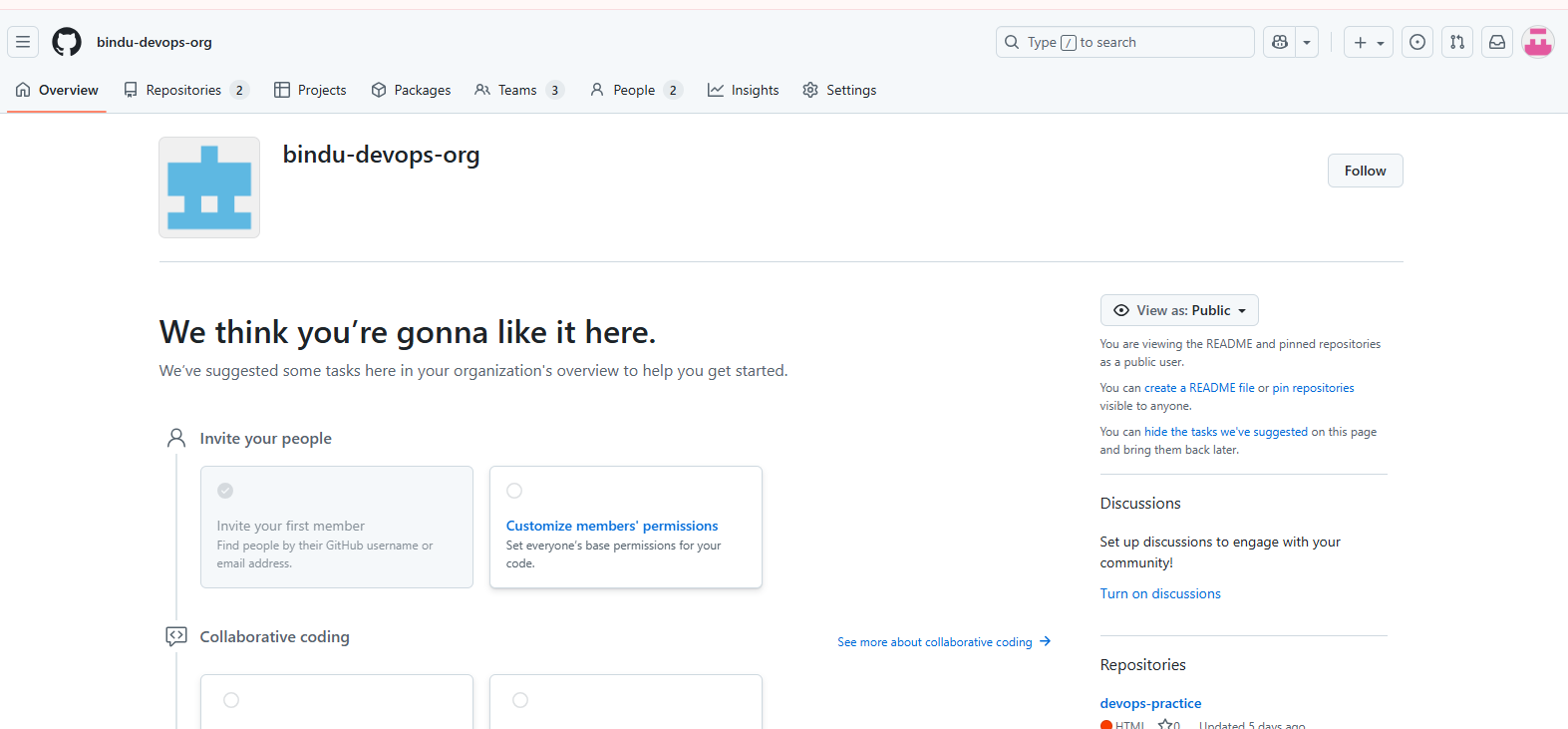
A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

Created GitHub Organization as bindu-devops-org



created the organization bindu-devops-org so that:

* All your practice repos (devops-practice) can stay in one place.
* We can simulate how real companies manage projects.
* Later, if teammates join, you can just add them with proper roles.

#### 4.2: Setting Up Repository

Created Repository as devops-practise

A screenshot of a computer

AI-generated content may be incorrect.

#### 4.3: : Added teams :

Added development teams, QA team, UAT team

A screenshot of a computer

AI-generated content may be incorrect.

#### 4.4: Added members:

Added members

A screenshot of a computer

AI-generated content may be incorrect.

##### 4.4.1: Created Branches:

To create branch git checkout -b develop

Created few branches

Main - Production-ready code

Develop- Integration of all features. Used by devs.

QA- Code is ready for QA testing.

UAT- code is ready for User Acceptance Testing

Feature1- login page

A computer screen with yellow text

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**Rule sets:**

To maintain proper workflow and avoid mistakes, we applied rulesets on branches in GitHub

A screenshot of a computer

AI-generated content may be incorrect.

We applied branch protection rules (rulesets) on all main branches (main, UAT, QA, develop) to enforce pull requests, code reviews, and prevent direct pushes. This ensures code quality and simulates real project workflow

**Why Rulesets Were Applied?**

* To prevent accidental commits to important branches.
* To enforce code review before merging.
* To maintain quality control across environments (Dev → QA → UAT → Main).
* To simulate real project workflow in a team.

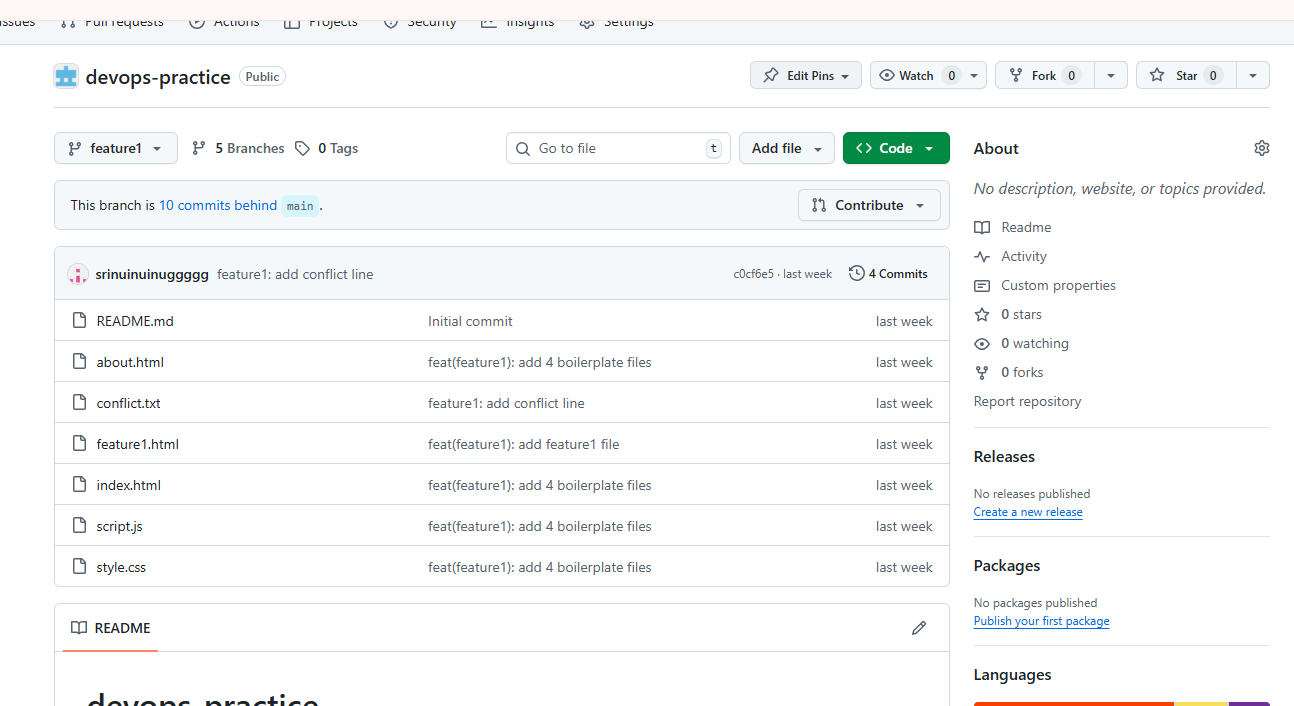
##### 4.4.2: Added Boilerplate code:

Added boilerplate code in feature1 branch:

A screenshot of a computer

AI-generated content may be incorrect.

Added about.html, index.html, script.js, style.css files





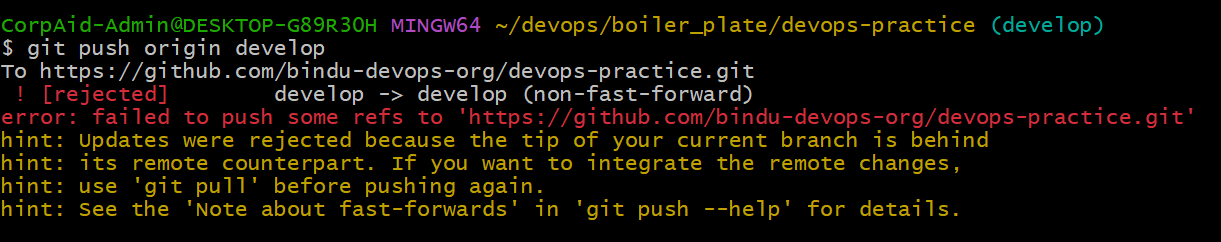
Added 4 files in feature1 branch

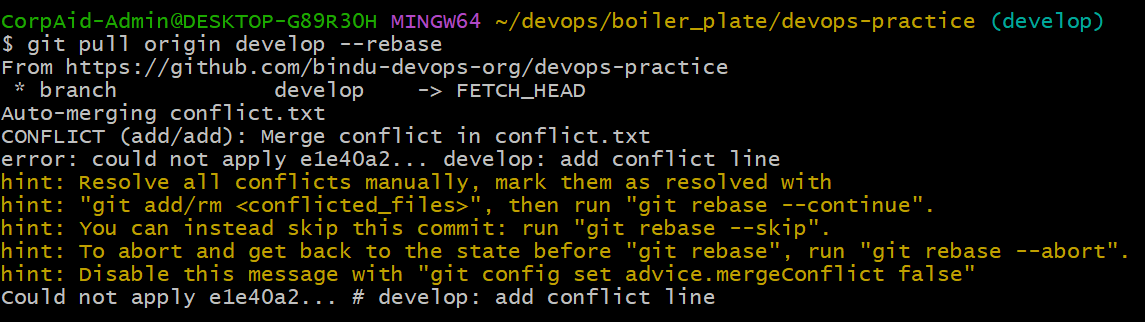
A screen shot of a computer

AI-generated content may be incorrect.

#### 4.5:Merge conflict solved

I tried to push develop branch but I got failed





This step pulls the latest remote changes and tries to reapply your local commits.

**Conflict Appears**:

During rebase, Git showed conflicts (<<<<<<< HEAD … >>>>>>>).

A computer screen with white text

AI-generated content may be incorrect.

feat(feature1): add login page code

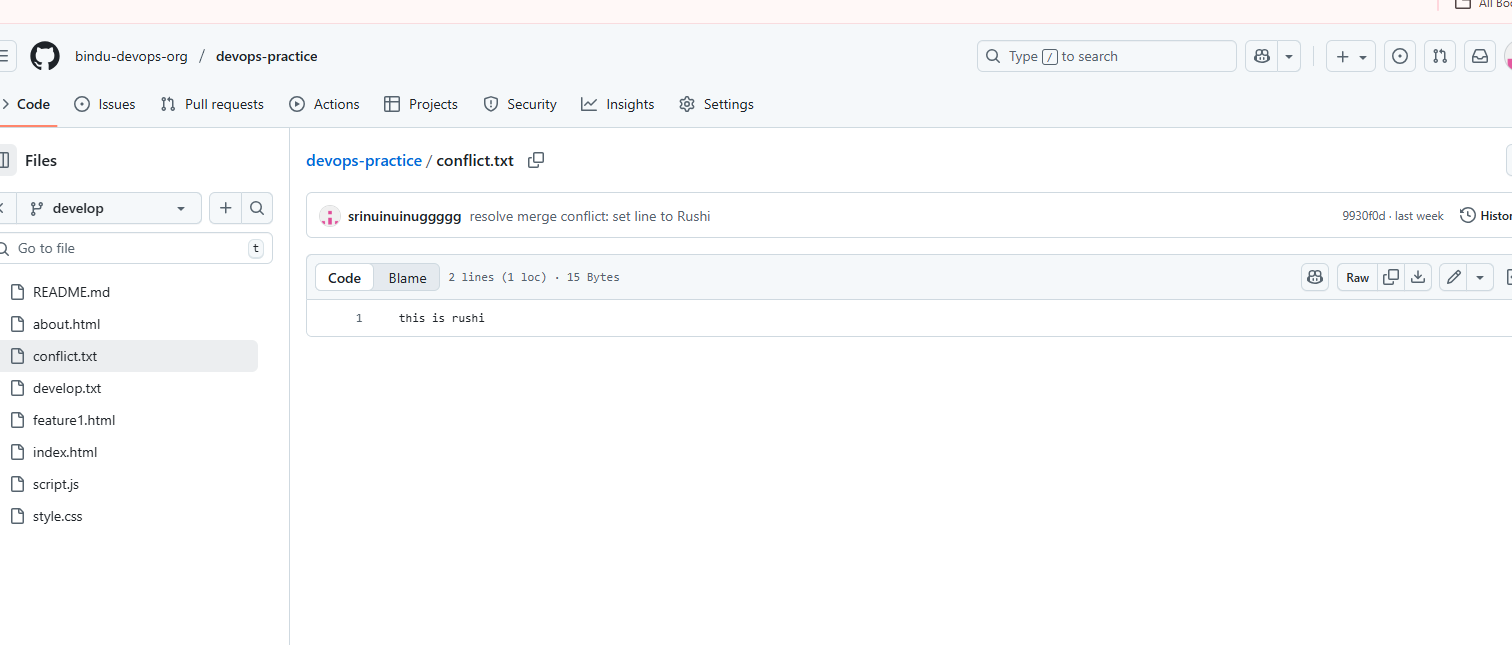
develop: add conflict line

then A PR was created from feature1 to develop.

Merge pull request #1 from bindu-devops-org/feature1

Since both feature1 and develop edited the same line, GitHub flagged a conflict.

resolve merge conflict: set line to Rushi



Successfully merged conflict

Pull request from feature1 from develop :

A screenshot of a chat

AI-generated content may be incorrect.

#### 4.6: . QA and UAT Branch:

* Pull request from develop to QA

A screenshot of a computer

AI-generated content may be incorrect.

Pull request from develop to QA, it successfully merged and closed

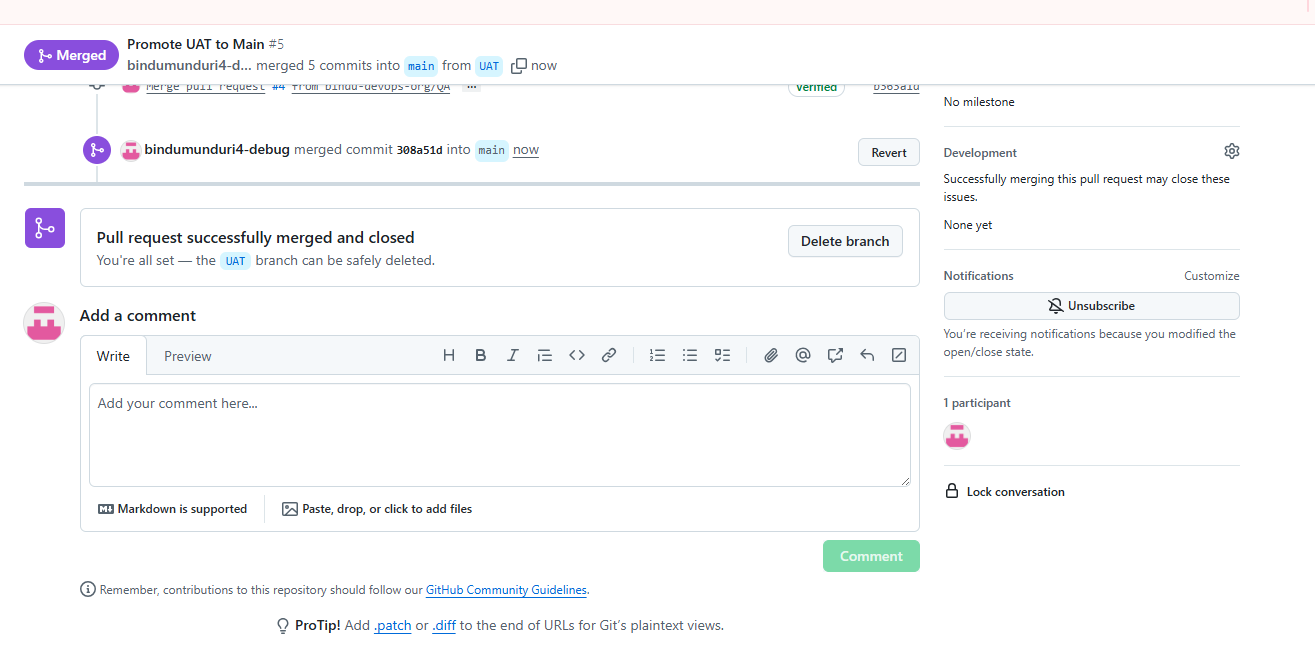
* Pull request from QA to UAT

A screenshot of a computer

AI-generated content may be incorrect.

Pull request from QA to UAT , it successfully merged and closed

* Pull request from UAT to main



pull request from UAT to main, it successfully merged and closed

4.7: . Graph log:

A screen shot of a computer program

AI-generated content may be incorrect.

Succesfully merge conflict resolved

Explanation:

**The very beginning**:

c5b2975 → Initial commit

**Main branch setup**

3db9590 chore: add README file

🡪First real file added (README)

🡪This commit is shared by **main, QA, UAT**

**Develop branch starts**

e3355e7 feat(develop): add develop base file

🡪created develop branch and added its base file.

860ca3c add something file

🡪Another commit added on develop

**Parallel work on feature1**

67330c4 develop: add conflict line

🡪change in develop that later conflicts with feature1.

5c1ff0c feat(feature1): add feature1 file

🡪Work started in feature1 branch.

fad9f38 feat(feature1): add 4 boilerplate files

🡪More work done in feature1

**Conflict area**

c0cf6e5 feature1: add conflict line

🡪feature1 added a conflicting line

9930f0d resolve merge conflict: set line to Rushi

🡪When merging, Git found a conflict between develop and feature1 fixed it manually (“set line to Rushi”)

**Merge PR**

10b9822 Merge pull request #1 from bindu-devops-org/feature1

🡪GitHub PR merged feature1 back into develop.

**New commits on develop after merge**

e1e40a2 develop: add conflict line

63fc894 feature1: add conflict line

🡪Additional commits on develop (after merging feature1).

Explanation:

 started with develop.

 created feature1 and worked (boilerplate + feature file + conflict line).

 Meanwhile, develop also got a change in the same file.

 When merging feature1 → develop, Git found a conflict.

 resolved it (set line to Rushi) and merged successfully.

 Then more commits were added on develop

### 

### 5.0 Annexure

Information / content of Annexure - I and so on…

[ SOPs are used to document many types of processes, while runbooks are typically used to document IT-specific tasks.

SOPs may also include contextual information and multiple processes, while runbooks typically cover one process. ]