# **Contents**

1. Git branching and merging strategy

2. Branching Structure

2.1 DEV Branch

2.2 SIT/UAT Branch

2.3 Production Branch

3. Feature Branches

4. Raise a Pull Request with Squash Commit

4.1 Raise a Pull Request

4.2 Choose the Squash Merge Option when the PR is ready to merge.

5. Merging Process: Step-by-Step Guide

5.1. Developing a Feature

5.2. Testing the Feature in Dev

5.3. Finalizing the Feature in SIT

5.3.1 Ensure Your Feature Branch is Up to Date

5.3.2 Creating a Bug Fix on any error/failure.

5.4. Finalizing the Feature in UAT

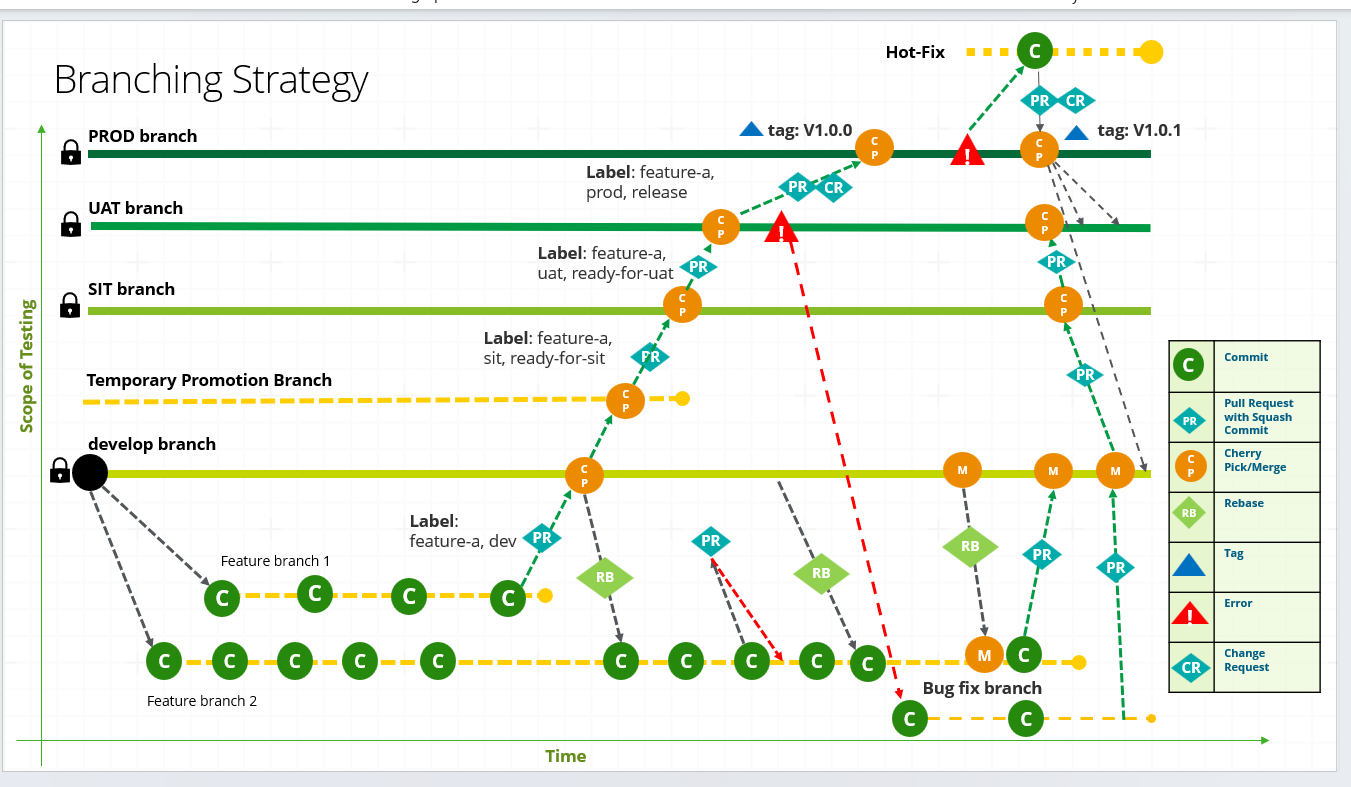
6.Releasing to Production

7.Adding labels when raising PR’s and tagging in Production

7.1 Tagging in Production

This document illustrates the branching structure/strategy that we are following, where the features start from DEV, get tested in DEV, and are eventually merged into Production.

Main branches that we make use of are DEV(Develop), SIT, UAT and Production (PROD).



**2.Branching Structure**

**2.1 Dev Branch:**

All deployments to the development environment are made from the DEV branch, which is also the most actively updated branch.

**2.2** **SIT Branch:**

SIT branch is for fully developed and tested features, which also includes the tests done by QA. No unfinished or partially tested features should be present in this branch.

**2.3 UAT Branch:**

UAT is for fully developed and tested features. This branch is a staging ground where features are checked and finalized before they are promoted to production. No unfinished or partially tested features should be present in this branch.

**2.4** **Production Branch:**

Production branch holds the code that is currently deployed in the production environment. Only fully stable and tested features reach this branch.

**3. Feature Branches**

Developers create feature branches for each individual task or feature they are working on. These branches are isolated from the main development branches until the feature is ready to be merged.

**4.Raise a Pull Request with Squash Commit**

Raising a Pull Request with a squash commit means that all your commits in a feature branch will be combined or squashed into a single commit when merged into the target branch.

**Commands**

* git checkout -b feature/my-feature
* git push origin feature/my-feature à Make the required changes and commit the changes.

**4.1 Raise a Pull Request**

* Go to your repository on Git hosting service.
* Click on Compare & pull request or “New Merge Request.”
* Select your feature branch as the source and the target branch.

**4.2 Choose the Squash Merge Option when the PR is ready to merge.**

* Click the Squash and merge option.
* Specify the commit message if needed.
* Confirm and merge.

**5. Merging Process: Step-by-Step Guide**

**5.1. Developing a Feature**

Start from DEV: Create a feature branch off the DEV branch. This ensures that your feature builds upon stable code that has already been tested.

**Commands:**

* **git checkout dev**
* **git checkout -b feature/dev**

**Note:** If there are multiple feature branches and if you want to cherry pick a Feature from Multiple Features follow the below procedure

* **git log à** Use git log to find the commit hash for the feature you want to cherry-pick.
* **git checkout <target-branch> à** Move to the branch where you want to apply the feature.
* **git cherry-pick <commit id1> <commit id2> à** Cherry-pick multiple commits
* **git push origin <target-branch> à** Push the changes to the remote repository.

**5.2. Testing the Feature in Dev**

Once we are done with the development of the feature, we will need merge it into the dev branch for testing. This step allows frontend engineers to start consuming the APIs or feature and helps identify any integration issues early on.

**Commands:**

* **git checkout dev**
* **git merge feature/dev**

**5.3. Finalizing the Feature in SIT**

After the feature has been tested in the dev branch, the next step is to promote it to the SIT branch. This branch should only include fully tested and completed features.

**Commands:**

* **git checkout SIT**
* **git merge feature/dev**

The SIT branch is then used for final user acceptance testing (UAT), as a pre-production checks. If any issues arise, they can be fixed here before moving to production.

**5.3.1 Ensure Your Feature Branch is Up to Date**

Create a Bug Fix Branch from the Feature Branch

**Commands:**

* **git checkout feature/SIT**
* **git pull origin feature/SIT**

**5.3.2 Creating a Bug Fix on any error/failure.**

**Commands:**

* **git checkout -b bugfix/feature-fix**

The above command creates a new bug fix branch and switches to a new branch named bugfix/feature-fix based on the feature/SIT branch. Post which we can edit the files and commit the changes and push back to the bug fix branch by running the below commands.

**Commands:**

* **git add .**
* **git commit -m "<commit message>”**
* **git push origin bugfix/feature-fix**

**Note:** If required create a pull request from bugfix/feature-fix into feature/SIT or the (desired branch) and merge back the changes.

**5.4. Finalizing the Feature in UAT**

Steps for UAT would be similar to SIT.

**6.Releasing to Production**

Once the feature passes all necessary tests and is deemed stable, it’s time to promote it to the production branch.

**Commands:**

* **git checkout master**
* **git merge UAT**

**7.Adding labels when raising PR’s and tagging in Production**

* When creating a PR, search for Labels.
* Click on “Labels” and select the required labels (e.g., bugfix, feature etc).

**7.1 Tagging in Production**

**Commands:**

**git tag <tag version> -m <message>**

* **git tag -a v1.0 -m "< tag comments if required>"**
* **git push origin v1.0**

**8. Creating Hotfix Branch for Production**

A hotfix branch is created to address the issues encountered in production.

**Commands:**

* **git checkout main**
* **git pull origin main**
* **git checkout -b hotfix/prod**

Once the hot fix branch is created make the required changes, commit and push the code back to production by following the below

**Commands:**

* **git add .**
* **git commit -m "<message>”**
* **git push origin hotfix/prod**
* **git checkout main**
* **git pull origin main**
* **git merge hotfix/prod**
* **git push origin main**

Merge back Hotfix into Dev, SIT, and UAT

**Commands:**

* **git checkout dev**
* **git pull origin dev**
* **git merge hotfix/prod**
* **git push origin dev**

**Note:** Follow the above commands for SIT and UAT to the code to be in sync

**9. Add repo for ETL**

Maintain a dedicated repository for the ETL application, as it is a distinct component within the architecture. This repository should include separate branches for each environment— DEV, SIT, UAT, and Production—to ensure clear separation of code and streamline the deployment process.

**10. Branching Strategy for Multiple DEV Environments**

If there are multiple DEV environments like DEV1, DEV2, etc the branching strategy supports parallel development

**10.1 Environment-Specific DEV Branches**

* Create separate branches out of dev branch for each DEV environment:
  + dev1
  + dev2
  + dev3

**10.2 Feature Branches**

* Feature branches are created from the respective DEV environment

Example: feature/dev1 from dev1

**10.3 Merging and Promotion**

* Once features are tested in a DEV environment, changes are merged into the dev branch then promoted to sit, uat, and finally main or can be merged directly from each dev1 branch to SIT.

**10.4 Sample Branch Structure**

main  
 |── uat  
 │  
 |── sit  
 │  
 |── dev  
 │ |── dev1  
 │ │ ── feature/dev1  
 │ |── dev2  
 │ │ ── feature/dev2  
 │ |── dev3  
 │ ── feature/dev3

* Similar structure continues for SIT environments aswell.

**11. Commit message standards**

All commit messages should follow the below structure

* **[Epic Number] | [Feature Number] | [Story] | [Commit Description]**

**Branching Strategy in Developer Point of View**

Reference Diagram on how the DEV team works on the code changes

* feature/\* → DEV → SIT → UAT → main/Production
* To avoid conflicts developers pulls the latest changes from DEV branch before working on feature branch.

**Commands**

* git checkout dev
* git pull origin dev
* git checkout -b <feature/\*>
* After executing the above commands, dev team starts working/adds the respective code changes by checking in to the feature branch and finally commits the changes

**Commands**

* git status --> To check if the changes made to the files are reflected
* git add . --> Stages all the changes
* git commit -m "<commit message" --> Commit the changes
* git push origin <feature/\*> --> Push the changes back to the feature
* Creates the PR from Feature branch and merges back to dev branch (i.e.) For example, goes to the GIT repo and then raises a PR <feature/\*> to dev.
* Once the PR is raised the changes from feature branch is merged to the dev branch

**Commands**

* git checkout dev
* git pull origin dev
* git merge <feature/\*>
* git push origin dev
* Once the code changes are complete for DEV the changes will be moved to SIT

**Commands**

* git checkout sit
* git pull origin sit
* git merge dev
* git push origin sit
* Once the code changes are complete for SIT the changes will be moved to UAT

**Commands**

* git checkout uat
* git pull origin uat
* git merge sit
* git push origin uat

Once the code changes are complete and a BUG is being identified in the UAT a Bug Fix branch would be created and the team fixes the issue in the code post which the changes would be merged back to DEV, SIT, UAT.

* Changes are moved to Production

**Commands**

* git checkout main
* git pull origin main
* git merge uat
* git push origin main