Git Integration Design Practices

This document outlines the standard practices of maintaining the repository and branching startegies to contribute any code. this strategy follows the master-develop-feature architecture as explained below.

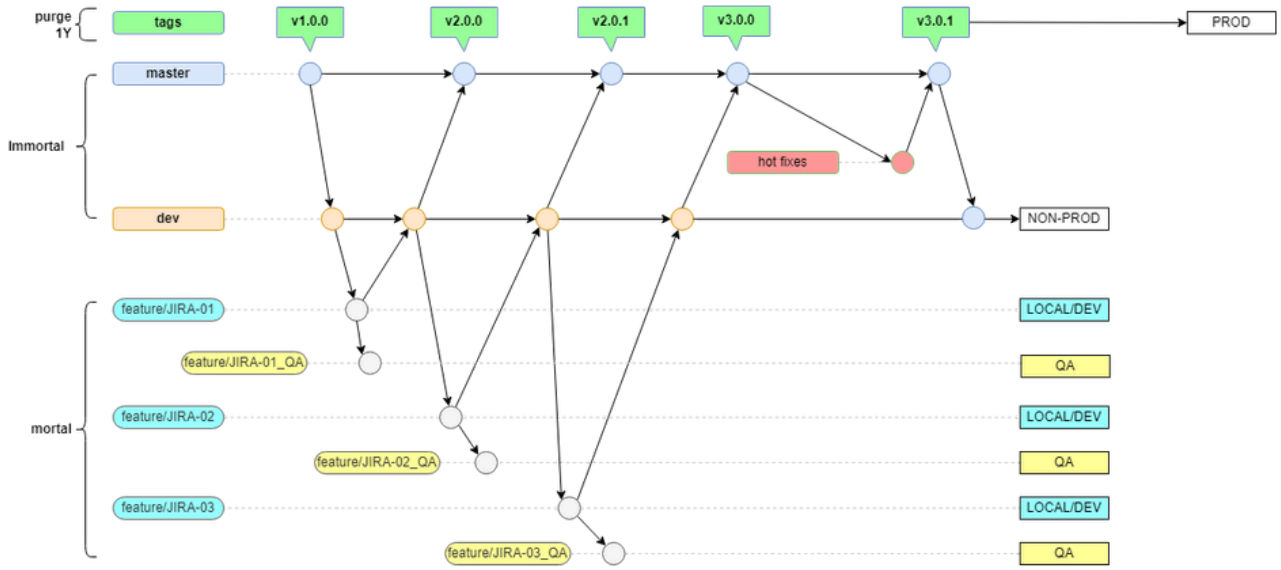
master-develop-feature branching practice

solution architecutre overview

this solution mainly consists of three leveals of branches.

- master
- develop
- features

solution architecture diagram



Branches overview

Master Branch

- master branch is the main branch for repo and it is the highest level in hierarchy
- there is only one "master" branch and it's immortal
- nobody can EDIT or CONTRIBUTE directly in master branch
- only repository owner have admin priviliges on this branch
- all the contribution must make it through merge requests from "develop" branch only
- no feature branches can merge its commits directly into master
- exception: "hot_fix" branches can be merged into master with certain approvals
  - everytime "hot_fix" branch merged into master, it should followed up by reverse merge request from master-to-develop to make develop in sync

Tags

- Tags are part of release created out of "master" branch right after the merge request completed
- Tags are used by the "PRODUCTION" applications
- PRODUCTION applications have flexibility to refer any tag from past
- Every release has it's own "Tag"
- Tags follow a naming standard with numeric identification
  - Example: v1.0.0, v2.0.0, v2.0.1 ..... etc.,
- Major releases always have an increment of higher level position
  - Example: v1.0.0; v2.0.0; v3.0.0; v4.0.0 ..... etc.,
- Minor releases like bugs, small enhancements, defects, hot fixes always have an increment of lower level position based on relativity to its preceeding release and the size of contribution
  - Example: v2.0.0; v2.1.0; v2.1.1; v2.2.0 ..... etc.,

Develop Branch

- develop branch is the second level in hierarchy and act as only child to "master"
- there is only one "develop" branch and it's immortal
- nobody can EDIT or CONTRIBUTE directly in develop branch
- only repository owner have admin priviliges on this branch
- all the contribution must make it through merge requests from "feature" branches
- feature branches can merge its commits into develop after the QA testing passed

Feature Branch

Developer feature branches

- feature branch is the third or lowest level in hierarchy and act as one of many childs to "develop"
- feature owner can EDIT or CONTRIBUTE directly in this branch for development and/or testing
  - data engineers
  - QA engineers
  - Any Others
- Repository owner have all admin related priviliges on this branch
  - delete branch
  - squash commits
  - restore, etc.,
- all the contribution in this branch must be developed and/or unit tested at LOCAL or DEV workspace environment
- feature branches can merge its commits into develop after the QA testing passed by creating merge request
- one or many feature branches can exist at same time based on Sprint activities
- must create a separate branch by Jira story
  - user story to feautre_branch is one-to-one

· feature branch naming standards:

- branche name : "feature/JIRA-1234"

QA feature branches

- QA feature branches act more like a sibling to developer feature branch
- they can be created out of another feature branch which does need a QA validation
- the QA feature bracnh contibutions can be commited back to its sibling (developer feature branch)
- most of the cases QA branches may not have code contributions in Data Engineering practices
- Exceptions:
  - it can be created out of "develop" main branch and contribute back to "develop" by merge requests
  - examples: if QA building any of their own frameworks
- one or many QA feature branches can exist at same time based on developer feature branches
- must create a separate "qa feature branch" by "developer feature branch"
  - qa_feautre_branch is one-to-one to developer_feautre_branch

· feature branch naming standards:

- branche name : "feature/JIRA-1234_QA"
- if there is any code contributions by QA team they can merge them back to it's parent "developer_feature_branch"

Hot-Fix branches

- hot-fix branches created out of "master" and they act like an instant temporary childs in hierarchy
- these can be mainly made by production support team
- all the contribution must make it through merge requests to "master"
- and these contributions must revrse-merged into "develop" right after the hot release

Merge Requests

- in this architecture, we mainly see three types of forward "merge" requests and two types of backward "merge" requests as explained below
  - forward merge requests
    - feature-to-develop
    - develop-to-master
    - hotfix-to-master
  - backward merge requests
    - master-to-develop
    - develop-to-feature

feature-to-develop

- this merge request is created to contribute any commits from feature branch to develop branch
- two approvals required to fullfill this merge request
  - QA Team
  - Team Lead
- this merge request must be created after a thourough unit testing by developer and validation by QA Team
- always check the DIFF between feature branch and "dev" branch
  - pull any commits that are ahead of your "feature" branch from "dev" branch, then only create your merge request

develop-to-master

- this merge request is created to contribute onr or many commits from dev branch to master branch
- this merge request is only executed by "release engineer"
- two or more approvals required to fullfill this merge request
  - Release Team
  - UAT Team/Code Owner
  - Team Lead
- this merge request must be created after securing the UAT approvals from (UAT/SIT systems if applicable)
- Release engineer checks the DIFF between "master" branch and "dev" branch
  - ideally master branch never go ahead with "dev"
  - if find DIFF - release engineer pull and make dev collect from master
    - do this operation by creating a backword merge request

hotfix-to-master

- this merge request is created to contribute adhoc commits part of high priority production support issues
- this merge request is only executed by "release engineer" (or) an assigned production support member
- two or more approvals required to fullfill this merge request

    - Support Lead
    - Others (if applicable as per Production Support agreements)
- this merge request must be created after a proper validation of hotfix changes
- after this merge request complete, always create a followup backward merge request master-to-develop
    - this will get the "dev" branch upto date with master

master-to-develop

- this is a backward merge request to make "dev" in sync with master
- this situation mostly arise when we are contributing any commits through the hot fixes

develop-to-feature

- this is a backward merge request to make "feature" branches in sync with "dev" branch
- this is needed only incase of developer wants to catchup their feature branch with latest commits by others into "dev"
- this merge helps resolving conflicts at feature levels
- always recommended to do this before we create a feature-to-develop merge requests
    - needed only when it find DIFF between your feature branch versus "dev" branch