Branching strategy, deployment and general rules

NPOYREKA edited this page on Sep 14, 2022 · 5 revisions

Introduction

A branching strategy is a convention or set of rules that specify when branches are created. This page describes the significance of each branch type along with a few naming guidelines of branches and commits messages.

Types of branches in FinApp

The FinApp central repo contains two categories of branches – MASTER and SUPPORT. Let us understand the significance behind each branch category in FinApp.

![Branching strategy](link to the diagram)

MASTER branches

FinApp uses three different types of MASTER branches. These branches have an infinite lifetime and are ordered by recent commits as –

- main branch - The source code in this branch reflects the state of the latest delivered development changes. The developed features merged in this branch are deployed to the DEV and QA environment.

- preview branch - The source code in this branch reflects the pre-production ready state. The developed features which are merged in this branch are deployed to the EDU environment.
- release branch - The source code in this branch reflects the production-ready state. Each time updates are merged into a release branch; this is a new 'production release' by definition.

SUPPORT branches

Our development model uses a variety of supporting branches to facilitate parallel development between team members. Unlike the MASTER branches, these branches have a limited life since they will be removed eventually. The different types of supporting branches used in FinApp are - feature, defect, and hotfix branches. Each of these branches has a specific purpose and is bound to strict rules as to which branches may be their originating branch and which branches must be their merge targets.

- feature - These branches are used to develop new features for the upcoming or distant future release. Depending on the requirement, they are branched off from main and preview and must be merged into their respective base branches. A deployment activity is triggered automatically for QA or EDU environment with every merge.
- defect - These branches are used to fix bugs belonging to certain features on QA or EDU. They are branched off from main and preview depending on the environment in which the defect is observed and must be merged into their respective base branches. A deployment activity is triggered automatically for QA or EDU environment with every merge.
- hotfix - These branches are used to fix bugs belonging to certain features in production. They are branched off from release and must be merged into the same branch. With every merge, a deployment activity is to be triggered manually for the Production environment.

Deployment activity

- A PR request into the main branch auto triggers Jenkins-dev-build-auto build (and deploy artifacts to the dev environment once FE and BE code split happens) and is considered a pre-requisite to approve the PR. This pipeline should also run a check for SonarQube Quality Gate and unit/integration tests and update the status in the PR. PR approval and merge become conditional to these 3 checks.
- A merge into the main branch auto triggers *eu-geft-dev-build* which builds and then deploys to the dev environment.

- This is followed by an auto-trigger of *eu-geft-qa-build* which deploys to the QA environment. Note that *eu-geft-qa* does not re-build artifacts; it simply deploys artifacts built already by *eu-geft-dev-build*.
- A merge into preview branch auto triggers *eu-geft-edu-build* which builds and then deploys artifacts into the EDU environment.
- A merge into release branch – requires manual trigger of *eu-geft-uat-build* which builds and then deploys artifacts into the UAT environment.
- Manually triggering the *eu-geft-prod-build* pipeline, results in deployment into the production environment. Note that *eu-geft-prod-build* does not re-build artifacts; it simply deploys artifacts built already by *eu-geft-uat-build*.

We also have an additional job *eu-geft-dev-branch-build* that builds and deploys to the dev environment. This pipeline is meant to be run by developers for their local development branch to ensure their changes build successfully before merging into any Master (aka long-lived) branch.

Branch naming conventions

1. Add branch type - like feature, defect, or hotfix, all in lowercase letters. With the help of branch type, it's easier to identify the purpose of the git branch.
2. Add ticket number - A task can either be a user story or a defect. A ticket number uniquely identifies a particular task.
3. Add environment details – Add the abbreviation of the environment for which the feature/defect branch is being created.
4. Use hyphens and slash separators – Without adding the separators, the names become more challenging to read, creating confusion for the team.
5. Add a brief title – Add a brief concise title to the branch name highlighting the context of the task. The title needs to be short and informative.

The following are some examples adopted by FinApp –

1. For feature branch, *feature/US12345678/QA/develop-IT-about-you-section* or *feature/US12345678/EDU/update-first-name-label-in-IT-about-you-section*
2. For feature branch with tasks, *feature/US12345678/task/TA12345678/QA/develop-IT-about-you-section* or *feature/US12345678/task/TA12345678/EDU/update-first-name-label-in-IT-about-you-section*

3. For defect branch, *defect/DE12345678/QA/fix-unable-to-save-IT-about-you-section* or *defect/DE12345678/EDU/fix-unable-to-save-IT-about-you-section*
4. For hotfix branch, *hotfix/US12345678/update-copy-on-GB-eligibility-screen*

Commit message naming conventions

1. Make atomic commits i.e., each commit should focus on one specific purpose.
2. Each commit should contain a ticket number.
3. Keep them concise. They should start with a single line that's no more than about 70 characters.
4. If you feel the need to add more details, add descriptions to your commit messages.
5. Avoid using special characters.
6. Try and use assertive sentences in the past tense.

The following are some examples for reference:

1. Use 'Added unit test cases for GB About you section' instead of 'Add unit test cases for GB about you section'
2. Use 'Updated translations of IT Document upload section' instead of 'Update translations of IT Document upload section'
3. Use 'Fixed build failure on QA' instead of 'Fix build failure on QA'.

The final commit message will look something like this - DE00000: Fixed build failure due to failing unit test case

    - When do we deploy next?
    - Runbooks
        - Production deploy
    - Team Directory
    - Team norms
    - How to navigate to the FinApp web portal
    - How to use test data spreadsheets
    - Architecture
    - Useful Troubleshooting Links and Queries
        - Dynatrace
        - Legacy Document (Dynatrace and Splunk)
    - Historical Troubleshooting
    - GEFT Splunk
    - How to create or fix a postman test for submitApplication for US with a complex user
    - How to view the site using test data
    - What and how do we test
- APIGEE Migration
    - Finapp US
        - Apigee Migration of Finapp US APIs
        - Finapp US Migration to SCAC APIGEE endpoints
    - Finapp EU
        - Apigee migration of Finapp EU APIs

        - Finapp EU Migration to SCAC APIGEE endpoints
        - Finapp EU Migration to SAP APIGEE endpoints
        - Finapp EU Migration to IAL APIGEE endpoints
        - Finapp EU Migration to FCM APIGEE endpoints

- Identity V2 Migration
    - Identity V2 Migration
- Strapi content authoring
    - Initial proposal
    - Technical spike
    - Integration process adopted by FinApp
    - Strapi API Data structure
    - Local static contents structure
- FMA
    - Add new url to allow list for FMA redirect target URL
- Ford Design System (FDS)
    - Grid
    - CSS and Iconography
    - Components
- Architecture
    - Environment mapping across Ford E-commerce ecosystem
    - Jenkins pipeline and deployment steps
    - Ideas for Refreshed Branching Strategies
    - Code: Style, Rules, & Norms
    - Code Smells
    - Branching strategy, deployment and general rules
    - IntelliJ Code Styling
    - PCF CLI
    - Architectural Decision Records

- Platform Roadmap
- Testing
  - Test Orders/Carts
    - Generate a test order via Showroom
  - Postman Collections
    - Creating or Fixing a US SubmitApplication Test
  - End to End testing
    - Cypress Automation
  - React
    - Wrapped in act(...) error
  - What, and how, do we test? (WIP)
- OneTrust Cookie Management CCPA
- Spikes
  - Smoke Test Framework
- Onboarding
  - Front-end
  - Backend
  - DevOps
  - QA
  - Team Glossary
  - Plus/Delta
  - APIC Onboarding
  - PCF Onboarding
  - APIs in FinApp
  - Auth Agent Implementation

    - FIM Group Setup
    - Dynatrace Onboarding
    - Vanity URL
    - Useful People
- External Contacts
    - Analytics Team
    - Auth Agent
- Important Links
    - Links with Special Access
- Git
- Incident Reports
    - INC000003801081 - ALD returns intermittent HTTP 500 for continueCreditApp
    - 5.5.0 Deployment Retrospective
    - 5.5.0 Deployment- Bank Validation Retrospective
    - 5.5 Deployment BankValidation issue
- 42 Crunch
- Jenkins and Build
    - Creating and Managing Jenkins Jobs
        - Jenkins Job DSL
        - Jenkins Pipelines and Libraries
        - Shared Jenkins Libraries
    - Legacy
        - Setting up Jenkins Pipeline & Deployments
- Misc
    - FCE Network outage fallback purposal
    - Ideas For Refreshed Branching Strategies
    - Wrapped in act(...) error message

- Improve the Wiki
- Legacy
  - Synchronize Commits from VFCA GPRS to GEFT Repos
  - GEFT Go Live
- Adobe Insights
  - Investigation into dropping customers April 2023

Clone this wiki locally

```
https://github.ford.com/global-ecommerce-finance-tech/region-agnostic-geft.wiki.git
```