### **Git Workflow and Branching proposal:**

This document outlines the proposed **Branching** **strategy** to enforce consistency and collaboration across the development lifecycle.

#### **2. Quick Builds for Draft Pull Requests**

* To facilitate rapid verification on real devices, the CI/CD pipeline should include a mechanism to bypass CI checks for **draft pull requests**.
* This applies to PRs raised from feature branches to the develop branch.
* Developers & other users can validate their changes quickly on real devices without waiting for the complete CI pipeline.

#### **3. Minimum Approvals for Pull Requests**

* The CI/CD pipeline/Git rules should enforce the following rules for pull requests (PRs):
  + PRs must **pass all CI checks**.
  + PRs must receive a minimum of **two approvals** from reviewers. Once development on sub-feature branches is complete then the updated feature branch should be reviewed by GIB. Feature & sub-feature branches explained in detail below.
* These conditions must be satisfied before merging PR into the develop branch.

#### **4. Commit Message Guidelines**

* The pipeline should validate that every commit message includes the branch name as a prefix. All commit messages should start with <branch\_name>: First letter capital and have an “imperative” mood.

Example:

**feature/phone-2fa: Fix the issue with API response handling.**

This ensures a clear association between commits and their corresponding tasks.

#### **5. Pre-Production Branch Naming**

* Pre-production branches should follow the naming convention: **rc-<Release\_Version>** Example: rc-1.2.0

#### **6. Hotfix Branch Naming**

* The naming convention for hotfix branches: **hotfix-<Release\_Version>** Example: hotfix-1.2.1

#### **7. Release Tagging**

* Should be able to create git tags for each release. Example: v1.2.0

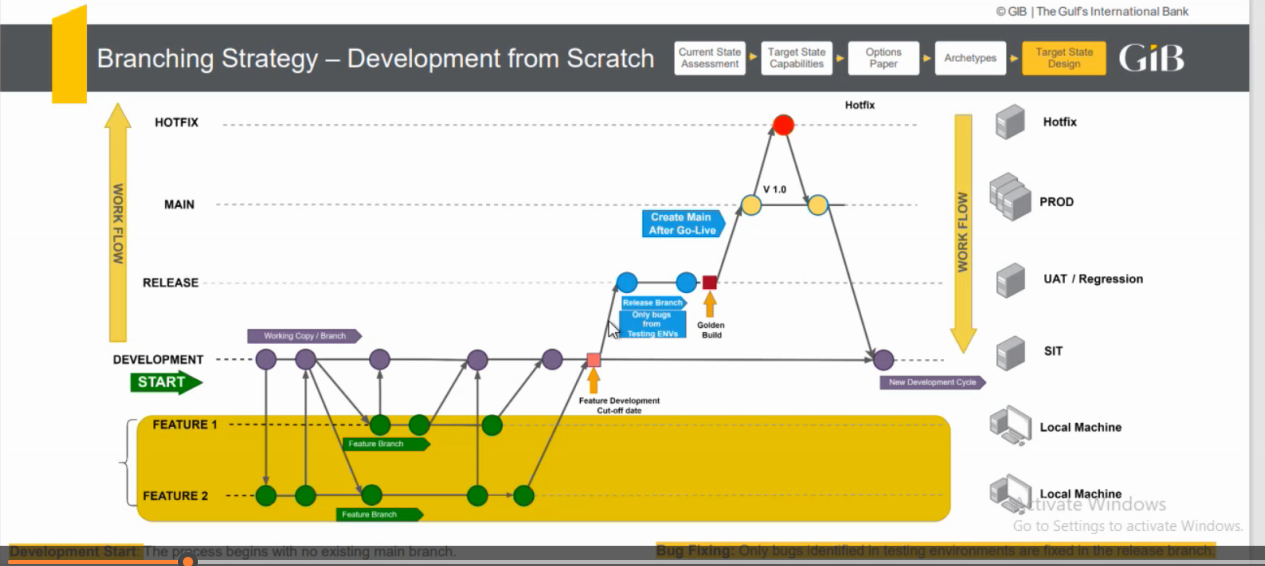
#### **8. Avoid Force Push on main branches.**

* Force push should not be allowed on the main branches like develop/release/hotfix/main

**9. General Branch Strategy shared by Devsecops team**

#### **Core Branches**

1. **main**: Code will finally be merged into main incrementally after each release or hotfix.
2. **release**: release branches per version, UAT should be performed on release branches
3. **hotfix:** any production critical issues/bug fixing
4. **develop**: Integration branch for completed features, preparing for release.
5. **SIT:** For integration tests, SIT branch should be created from develop, any bug fixes should be done on SIT. We should maintain SIT branch versions incrementally, once SIT completes it should be merged into the develop
6. **feature/use\_case**: Parent feature branch for any use case.
7. **bugfix:** For any bug fixes



### **Branching Strategy for Onboarding Feature**

#### **Task-Specific Branches**

The onboarding feature is divided into five tasks, each representing a screen or workflow:

Parent Branch: **feature/onboarding**

* **feature/onboarding-tutorial**: Onboarding Tutorial.
* **feature/onboarding-phone-2fa**: Phone Number & 2FA (via Nafath App).
* **feature/onboarding-personal-details**: Personal Details (Title, Email, Marital Status, etc.).
* **feature/onboarding-password-setup**: Password Setup.
* **feature/onboarding-account-confirmation**: Account Setup Confirmation.

**General Practices:**

1. Code should be regularly checked in the sub-feature branch (daily or max alternate days)
2. Any completed sub-feature should be merged with the parent feature promptly, thereby reducing the merge conflicts.
3. Use of mock data to work independently if needed for the given case.