Git branching model

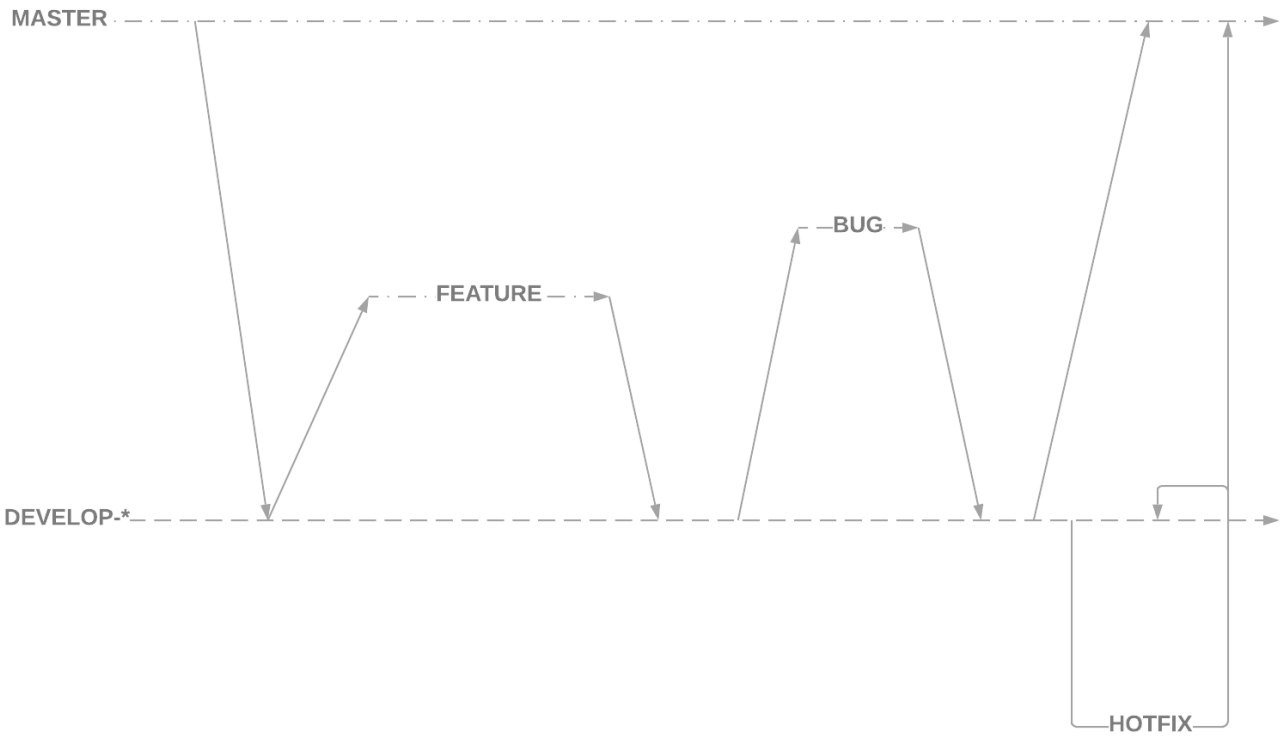
Our git branching model is based on the Git Flow - a git branching model suggested by Vicent Driessen in 2010 and the GitHub Flow (another model created by GitHub in 2011). As our big project is divided into **3 mini projects**, our workflow is based on **4 main branches** with infinite lifetime.

- **master** - this branch is for production code. All development code is merged into master sometime. In addition, anything in the master branch is deployable.
- **develop-1** this branch contains pre-production code. When the features are completed, they will merge into **develop-1** by team 1 (team Mobiusation).
- **develop-2** this branch has a similar meaning with **develop-1.** But this branch is for team 2 (team Pentagon).
- **develop-3** this branch is for team 3 (Team Rectify).

Note: you can only work on the develop branch of your team. For example, if you are in team 1, you are not allowed to submit anything on the develop branch of team 2.

During the development cycle, a variety of supporting branches are used:

- **feature-*** feature branches are used to develop new features for the upcoming releases. May branch off from **develop-*** and must merge into **develop-***. Note: all the 3 teams should concur with each other to use a common naming convention for naming tickets to avoid conflicts in creating branches as well as to make the repository more manageble.
- **hotfix-*** hotfix branches are necessary to act immediately upon an undesired status of **master**. May branch off from **master** and must merge into **master** and **develop-***.



Advantages:

- Ensures a clean state of branches at any given moment in the life cycle of project
- The branches naming follows a systematic pattern making it easier to comprehend

Disadvantages:

- Difficult to make **the Continuous Delivery and the Continuous Integration** because of the master/develop split.

Some rules we have to follow strictly when working together on the same repository:

1. No direct commits on master.
2. Test all commists, not only ones on master.
3. Review code carefully before merging from develop-* to master.

4. Fix bugs in master first and release branches second.
5. Commit messages reflect intent. Refer https://www.conventionalcommits.org/en/v1.0.0-beta.2/ to learn how to write meaningful commit messages.

References:

4 branching workflows for Git

The Git Flow

The GitHub Flow