Mobility Practice

GIT Repository Usage Guidelines

Version 1.0

1. Introduction

1.1. Getting Started

The proposed workflow is just an abstract idea of a Git workflow. This means it dictates what kind of branches to set up and how to merge them together. Inspiration for this workflow is Gitflow Workflow.

1.2. Scope

This document defines the overall GIT workflow standard including a strict branching model designed around the project release. Defined workflow assigns specific roles to different branches and defines the branch interaction.

1.3. Audience

This document is targeted to all the developers/members within Xebia Mobility Practice. It is expected that all developers/members will follow the same standard in every project inside Xebia Mobility Practice.

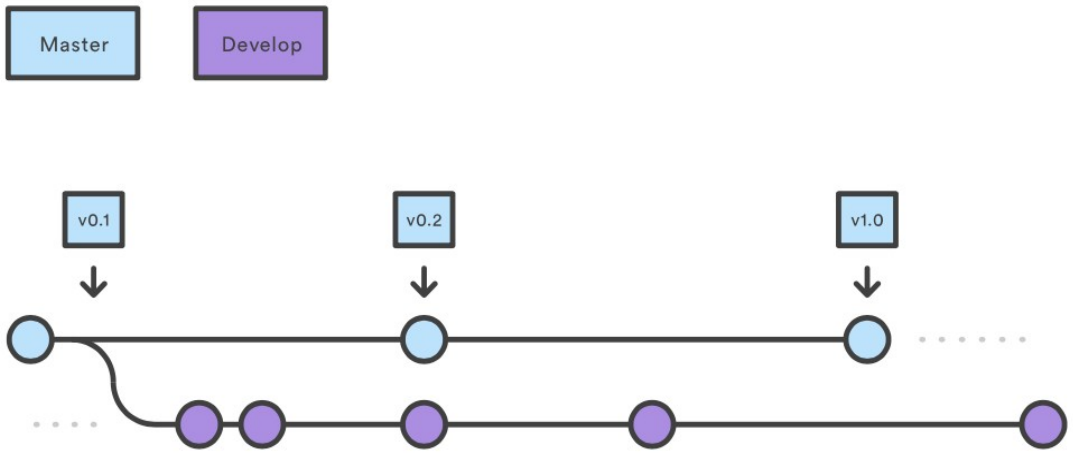
1.4. Objective

This document defines a standard process of using any GIT Server to store codebase/files and access them through any GIT Client.

2. Branch Workflows

2.1. Develop and Master Branches

Instead of a single master branch, this workflow uses two branches to record the history of the project. The master branch stores the official release history, and the develop branch serves as an integration branch for features. It's also convenient to tag all commits in the master branch with a version number.



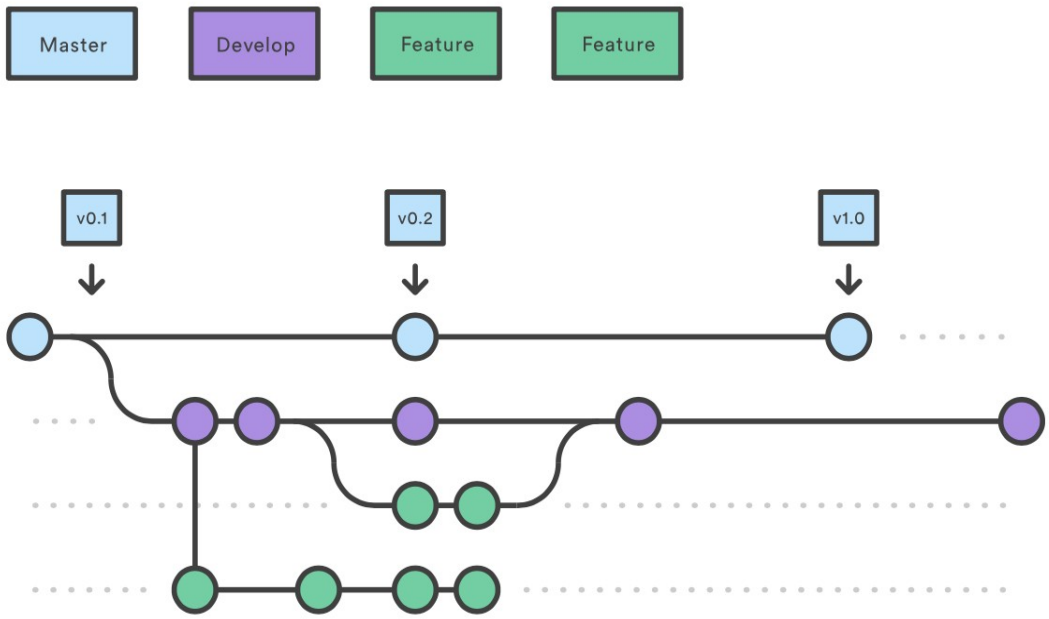
The first step is to complement the default master with a develop branch. A simple way to do this is for one developer to create an empty develop branch locally and push it to the GIT server.

This branch will contain the complete history of the project, whereas master will contain an abridged version.

Other developers should now clone the central repository and create a tracking branch for develop.

2.2. Feature Branches

Each new feature should reside in its own branch, which can be pushed to the central repository for backup/collaboration. But, instead of branching off of master, feature branches use develop as their parent branch. When a feature is complete, it gets merged back into develop. Features should never interact directly with master. Feature branches are generally created from the latest develop branch.



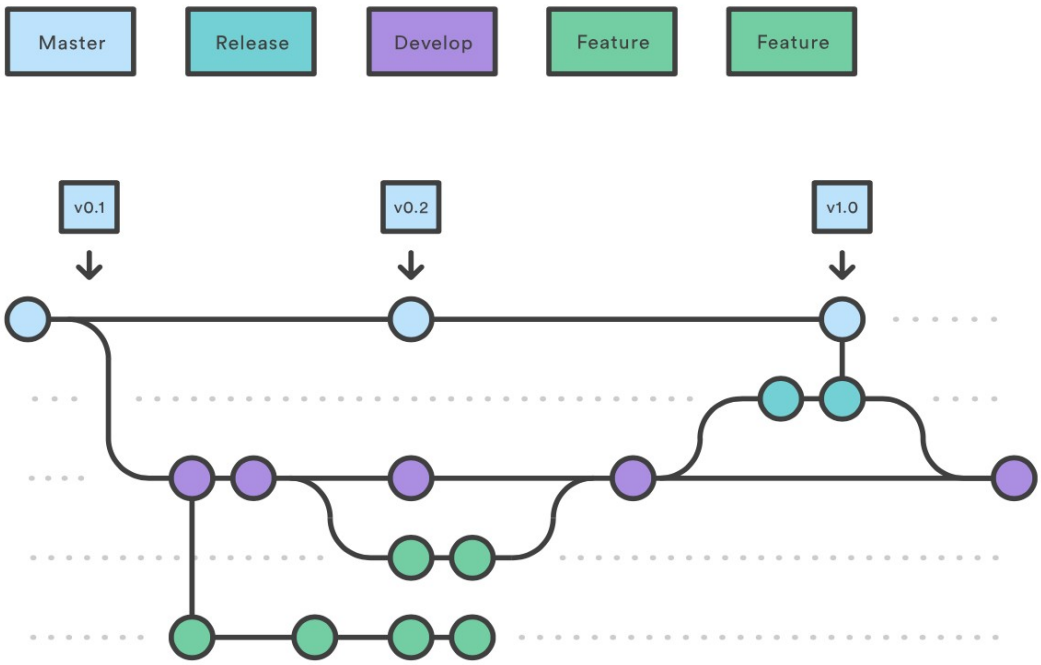
2.2.1. Finishing a feature branch

When the development work is completed on the feature, the next step is to merge the feature branch into develop.

2.3. Release Branches

Once develop has acquired enough features for a release (or a predetermined release date is approaching), you fork a release branch off from develop. Creating this branch starts the next release cycle, so no new features can be added after this point—only bug fixes, documentation generation, and other release-oriented tasks should go in this branch.

Once it's ready to ship, the release branch gets merged into master and tagged with a version number. In addition, it should be merged back into develop, which may have progressed since the release was initiated.



Using a dedicated branch to prepare releases makes it possible for one team to polish the current release while another team continues working on features for the next release. It also creates well- defined phases of development (e.g., it's easy to say, "This week we're preparing for version 4.0," and to actually see it in the structure of the repository).

Making release branches is another straightforward branching operation. Like feature branches, release branches are based on the develop branch.

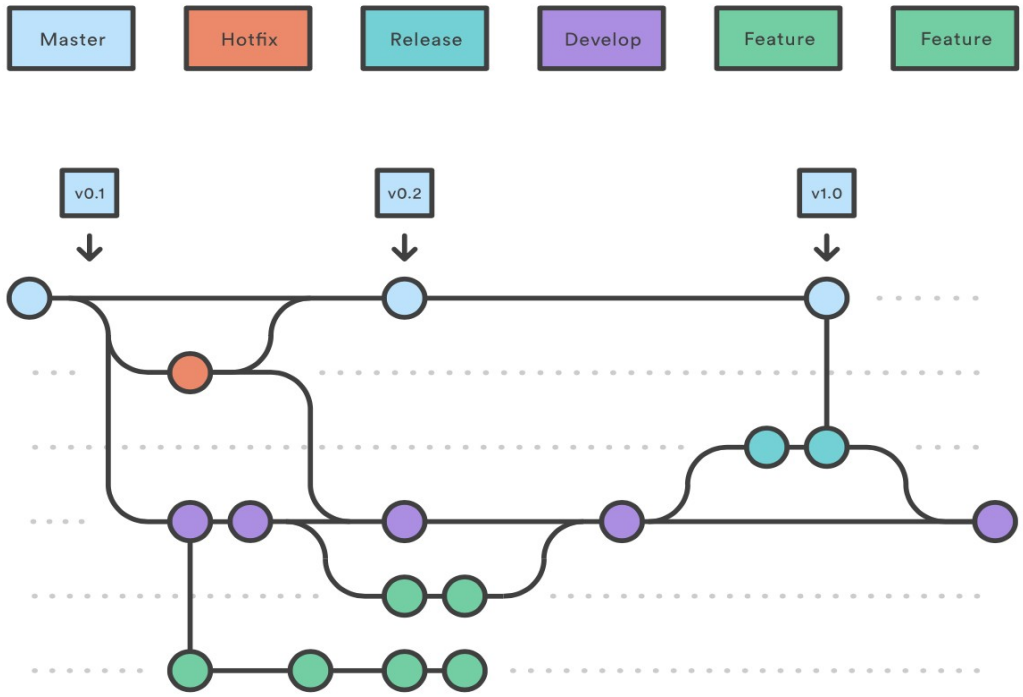
Once the release is ready to ship, it will get merged into master and develop, then the release branch will be deleted.

It's important to merge back into develop because critical updates may have been added to the release branch and they need to be accessible to new features.

Code review may be ideally carried out in this branch through a pull request.

2.4. Hotfix Branches

Maintenance or Hotfix branches are used to quickly patch production releases. Hotfix branches are a lot like release branches and feature branches except they're based on master instead of develop.



This is the only branch that should fork directly from master.

As soon as the fix is complete, it should be merged into both master and develop (or the current release branch), and master should be tagged with an updated version number. Then the Hotfix branch should be deleted.

Having a dedicated line of development for bug fixes lets your team address issues without interrupting the rest of the workflow or waiting for the next release cycle

2.5. Key Takeaways

Some key takeaways about Gitflow are:

- The workflow is great for a release-based software workflow.
- This offers a dedicated channel for hotfixes to production.

The overall flow is:

1. A develop branch is created from master
2. A release branch is created from develop
3. Feature branches are created from develop
4. When a feature is complete it is merged into the develop branch
5. When the release branch is done it is merged into develop and master
6. If an issue in master is detected a hotfix branch is created from master
7. Once the hotfix is complete it is merged to both develop and master

3. Conclusions

This workflow should be strictly maintained across all the GIT repositories, irrespective of every application (new or insourced), developed/maintained by Xebia Mobility Practice.

4. References

https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow