**Understanding the Importance of Git Workflow: A Comprehensive Guide to GitFlow**

In the ever-evolving landscape of software development, efficient collaboration and version control are paramount. Git, with its powerful branching and merging capabilities, has become the de facto standard for version control in the industry. However, to harness the full potential of Git and ensure seamless collaboration among team members, adopting a structured workflow is essential. One such popular workflow is **GitFlow**. In this comprehensive guide, we delve into the significance of Git workflow, focusing on **GitFlow** and its benefits in modern software development practices.

**Why Git Workflow Matters**

**Organized Collaboration**

Git workflow provides a structured approach to collaboration, enabling teams to work cohesively on projects. By defining clear guidelines for branching, merging, and code review processes, it fosters better communication and coordination among team members

**Version Control**

With Git workflow, every change made to the codebase is tracked systematically through commits and branches. This ensures that developers have a complete history of the project's evolution, making it easier to trace back changes, revert to previous states, and identify the source of issues.

**Code Stability**

Git workflow promotes code stability by segregating development tasks into distinct branches. Features, bug fixes, and enhancements are developed in separate branches, preventing conflicts and minimizing the risk of introducing errors into the main codebase.

**Continuous Integration and Deployment (CI/CD)**

By adopting Git workflow, teams can seamlessly integrate continuous integration and deployment practices into their development pipelines. Automated testing, build processes, and deployment scripts can be triggered based on predefined branching strategies, ensuring rapid and reliable software delivery.

**Understanding GitFlow**

**TL;DR**

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

GitFlow is a branching model that defines a set of rules and conventions for managing Git repositories effectively. It was introduced by Vincent Driessen and has gained widespread adoption among development teams worldwide. The GitFlow workflow is characterized by the following key branches:

**Master Branch**

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.

**Develop Branch**

A diagram of a diagram

Description automatically generated

The develop branch serves as the integration branch for ongoing development efforts. It contains the latest changes and serves as the basis for feature development and bug fixes.

with a develop branch. A simple way to do this is for one developer to create a develop branch locally from master and push it to the server:

|  |
| --- |
| $ git checkout master  Switched to branch 'master'  Your branch is up to date with 'origin/master'.  $ git branch develop  $ git push -u origin develop  Total 0 (delta 0), reused 0 (delta 0), pack-reused 0  remote:  remote: Create pull request for develop:  remote: https://bitbucket.global.renesas.com/projects/GMANUF/repos/gym\_file\_loader/pull-requests?create&sourceBranch=refs%2Fheads%2Fdevelop  remote:  To ssh://bitbucket.global.renesas.com:7999/gmanuf/gym\_file\_loader.git  \* [new branch] develop -> develop  branch 'develop' set up to track 'origin/develop'. |

**Feature Branches**

A diagram of a computer

Description automatically generated

Feature branches are created from the develop branch for implementing new features or enhancements. Each feature is developed in isolation, allowing developers to work independently without disrupting the main codebase. 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.

Our features should always come from [jira](https://jira.global.renesas.com/projects/GYM/issues/GYM-529?filter=allopenissues) so that anyone can track all the details to understand what's being developed. To create new feature branch do as follow (You can use CLI if you prefer).

from your issue in Jira click create branch link (Under Development):

Select the repository, set the branch type to Feature, set the branch from to develop **(not master),** leave the default branch name (it will be the code and the name of the task)then click continue:  
A screenshot of a computer

Description automatically generated

After that just make sure you pull this branch to you local working space and make sure that you commit all the changes into this bucket, once everything is done you should create a pull request (PR) so that you code is merged in the develop branch, you can create the PR using the web based version of bitbucket.

**Note:** We delete the feature branch after it is merged back into develop, because if your feature causes any problem it will be fixed using a deferent branch (like hotfix or bugfix)

**Release Branches**

A screenshot of a computer

Description automatically generated

A diagram of a software development process

Description automatically generated

Release branches are created from the develop branch when preparing for a new release. They serve as staging areas for final testing, bug fixes, and versioning before deployment to production.

Once develop has acquired enough features for a release (or a predetermined release date is approaching), we create release branch from develop. Creating this branch starts the next release cycle, so no new features can be added after this point (on the release  branch)—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. A new release branch can be created using the following methods.

For all release an epic should be created on Jira and all the subtasks (like creating documentations and the bugfixes should be added to that epic)

You can follow the same process as you did when creating a feature branch (but using an epic instead) and configure as follow:

A screenshot of a computer

Description automatically generated

Once the release is ready to ship, it will get merged it into 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.

To finish a release branch, use the following methods:

|  |
| --- |
| $ git checkout master  Switched to branch 'master'  Your branch is behind 'origin/master' by 15 commits, and can be fast-forwarded.  (use "git pull" to update your local branch)  $ git merge release/GYM-686-release-v1.0.0.1  $ git push origin master  $ git tag -a v1.0.0.1 -m "Tagging release merge into master"  $ git push origin v1.0.0.1 |

This branches will be managed by the owners of the branches or with persons that they will give the responsibility.

And they will be responsible for merging the code with production

For now these are the owners of some of ours backend projects:

* Gym\_file\_loader (paerser) - [Chris Baptist](https://confluence.renesas.com/display/~chris.baptist.eb@renesas.com)
* Gym\_jobs - [Sany Singh](https://confluence.renesas.com/display/~sany.singh.xm@renesas.com) or [Paulo Safrao](https://confluence.renesas.com/display/~paulo.safrao.jx@renesas.com)
* Gym\_services - [Sany Singh](https://confluence.renesas.com/display/~sany.singh.xm@renesas.com)  or [Paulo Safrao](https://confluence.renesas.com/display/~paulo.safrao.jx@renesas.com)

**Hotfix Branches**

Hotfix branches are created from the master branch to address critical issues or bugs found in the production environment. They enable quick fixes to be applied without disrupting ongoing development efforts.

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 off of 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.

Having a dedicated line of development for bug fixes lets our team address issues without interrupting the rest of the workflow or waiting for the next release cycle. You can think of maintenance branches as ad hoc release branches that work directly with master. A hotfix branch can be created using the following methods:

For all hotfixes an task should be created on Jira

A diagram of a software project

Description automatically generated

You can follow the same process as you did when creating a feature branch and configure as follow:

A screenshot of a computer

Description automatically generated

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

|  |
| --- |
| $ git checkout master  Switched to branch 'master'  Your branch is behind 'origin/master' by 15 commits, and can be fast-forwarded.  (use "git pull" to update your local branch)  $ git merge hotfix/GYM-686-hotfix-v1.0.0.1  $ git push origin master  $ git tag -a v1.0.0.1 -m "Tagging release merge into master"  $ git push origin v1.0.0.1  $ git checkout develop  Switched to branch 'develop'  Your branch is behind 'origin/develop' by 15 commits, and can be fast-forwarded.  (use "git pull" to update your local branch)  $ git merge hotfix/GYM-686-hotfix-v1.0.0.1  $ git push origin develop |