Git is one of the best version control available now. It is so flexible. You can create your own workflow strategies using git. The main well-known workflows for git are.

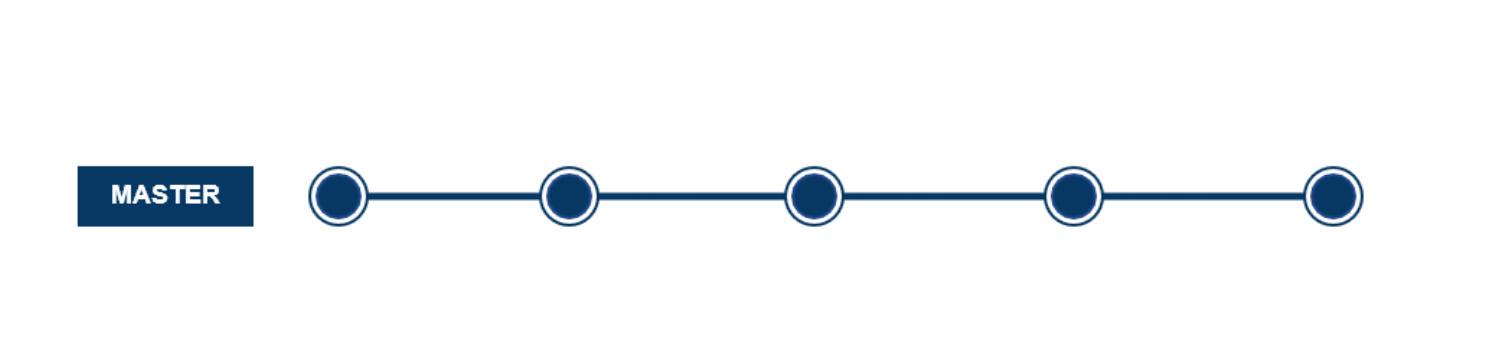
* Basic Workflow
* Feature Branch Workflow
* Git Flow
* Gitlab flow

**Basic Workflow**

This is the basic workflow. There is one central repository. What this means is that each developer will clone the repo, work locally on the code, make a commit with various changes, and then finally put it in the central repository for other developers to take and put in their own individual work.

**While the basic workflow is a fantastic method for creating and developing a simple website**, it doesn’t do well with any complications. If two developers need to work on two separate functionalities inside one project, then the Basic workflow is no longer an ideal method for the team.

*For example, one of the developers has just finished their functionality and wants to release it. They cannot release it if the second feature from the second developer is incomplete. Because both of them are working on the same branch.*If they want to make the release anyway then this will result in so many unfavorable issues for the developers, not to mention the project at hand.



The above diagram shows an example of a simple workflow. Here it is only creating one branch which is the master branch. All the developers are adding their commits to the master branch. So the branch will become stable once all the users finish their commits.

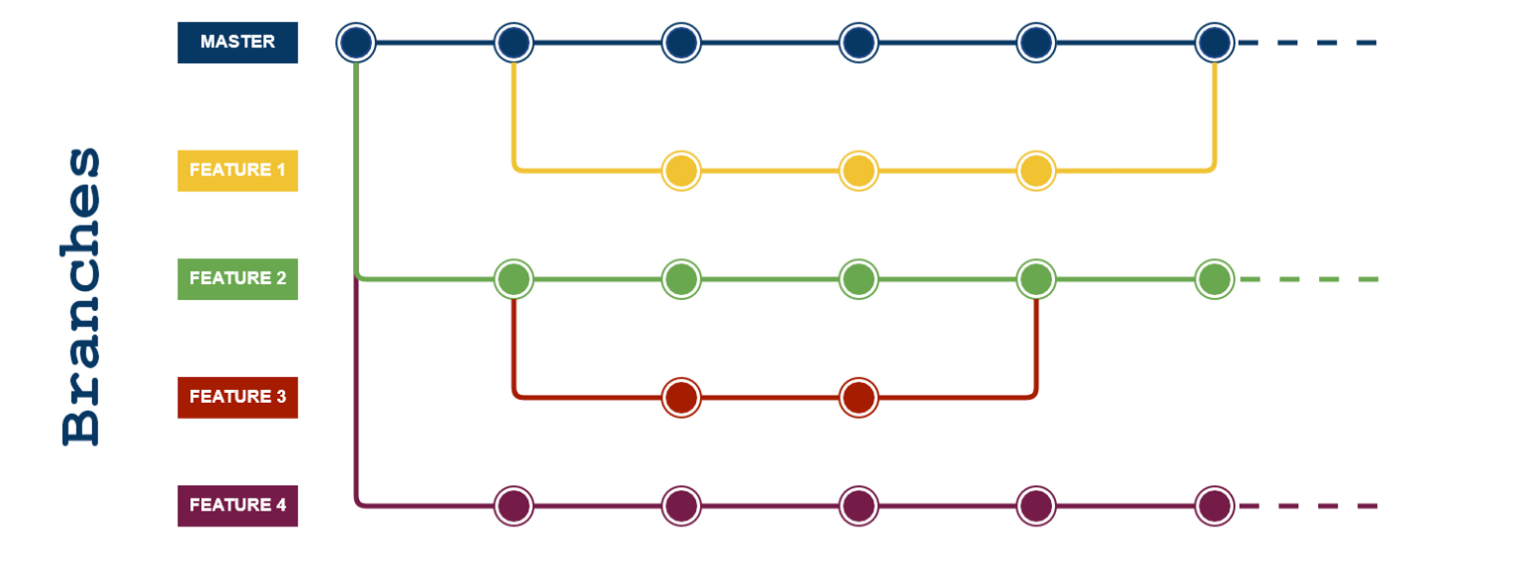
The above-mentioned issue can resolve by the next workflow.

**Feature Branch Workflow**

The Feature Branch Workflow assumes a central repository and master represents the official project history. **Instead of committing directly on the master branch, developers can create a new branch every time they start work on a new feature.** Feature branches should have descriptive names.

The suggested way to utilize this type of workflow would follow a hierarchal method that distinguishes between various levels of staff. **Before a branch is merged to master, it needs to be verified and checked for errors**. Junior developers can create a merge request and assign it to one of the senior developers. **The Seniors in turn can review the code and leave any necessary comments.** If everything seems satisfactory to move forward, the request is accepted and the branch is merged.

Here below explains the diagrammatic explanation about feature branch workflow.



1. Anything in the master branch is deployable and is stable.
2. To work on something new, create a branch off from master and given a descriptive name(ie: ui-upgrade).
3. Also, you can create branches from the existing feature branch.
4. Commit to that branch locally and regularly push your work to the same-named branch on the server.
5. When you need feedback or help, or you think the branch is ready for merging, open a pull request.
6. After someone else has reviewed and signed off on the feature, you can merge it into master.
7. Once it is merged and pushed to master, Your feature is ready for deployment.

## Advantages

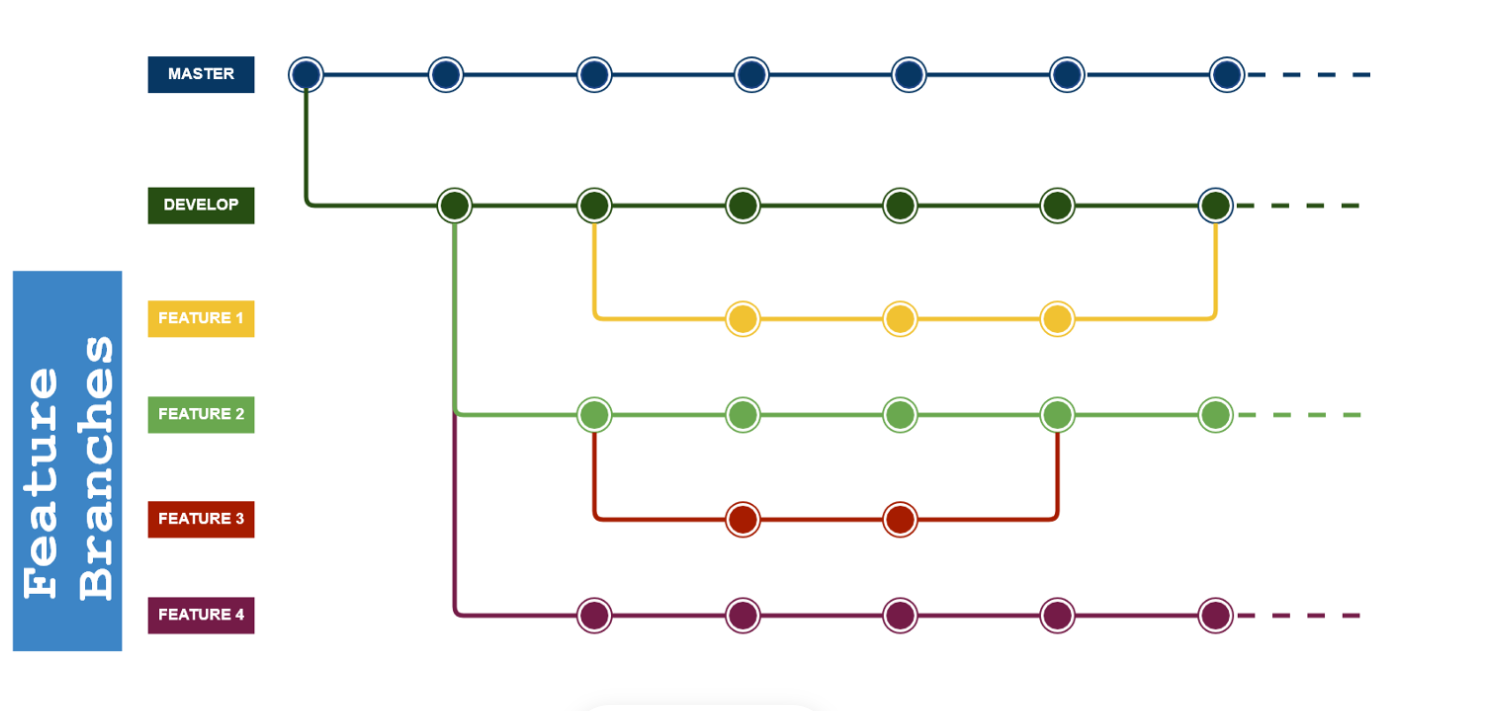
* it is friendly for the [Continuous Delivery](https://medium.com/javarevisited/7-best-courses-to-learn-jenkins-and-ci-cd-for-devops-engineers-and-software-developers-df2de8fe38f3) and [Continuous Integration](https://medium.com/javarevisited/13-best-courses-to-learn-devops-for-senior-developers-in-2020-a2997ff7c33c)
* A simpler alternative to Git Flow
* It is ideal when it needs to maintain a single version in production

## Disadvantages

* The production code can become unstable most easily
* Are not adequate when it needs the release plans
* It doesn’t resolve anything about deploy, environments, releases, and issues
* It isn’t recommended when multiple versions in production are needed.

# Git Flow

The Git Flow is the most known workflow on this list. It is almost similar to the feature branch workflow. But the difference is the developers are creating branches from the develop branch and it is a branch of master branch. [Developers](https://medium.com/javarevisited/top-10-pluralsight-courses-to-learn-programming-and-software-development-during-covid-19-stay-at-30b7d8a4f88f) are not allowed to create branches directly from master branch. This flow eliminates buggy code from the master branch.



* master — this branch contains production code. All development code is merged into master in sometime.
* develop — this branch contains pre-production code. When the features are finished then they are merged into develop.

During the development cycle, a variety of supporting branches are used such as feature, releases, bugfixes..etc

## Advantages

* Git Flow is used by a lot of distributed, open-source teams that have varying skill levels. The project maintainers can review and approve every line of code going into releases.
* Git Flow can work well for a traditional release model, where releases are done in terms of months and weeks.
* Git Flow also works well when dealing with an established product or multiple versions in production.

## Disadvantages

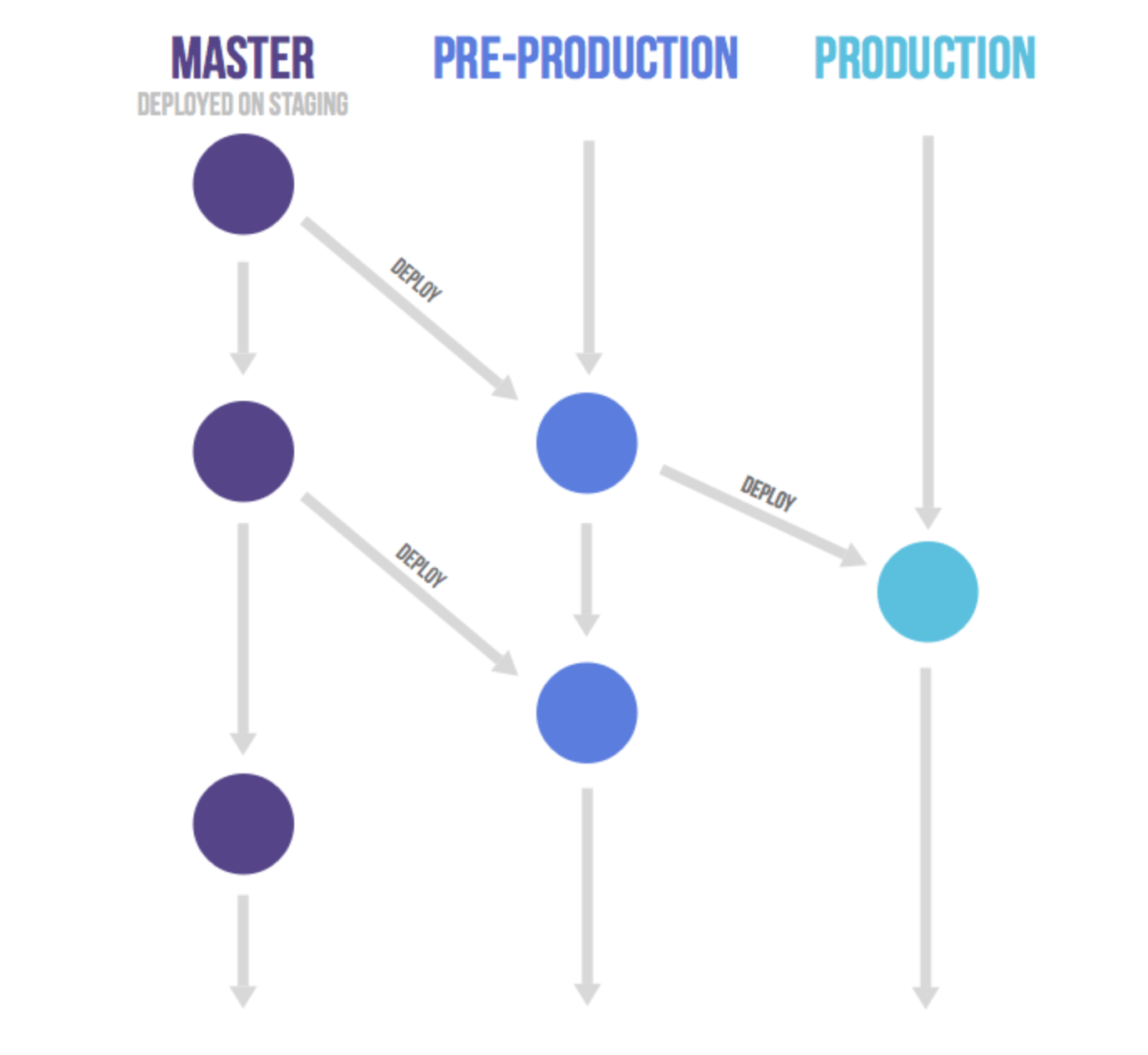
* Git Flow can slow things down when having to look at large pull requests if you are trying to iterate quickly.
* Releases are hard to do more than once a week.
* Larger features can spend days merging and resolving conflicts (merge hell) and force multiple testing cycles.
* The Project history is full of merge commits and makes it hard to see the actual work.
* Can be problematic in [Continuous Integration](https://javarevisited.blogspot.com/2018/09/top-5-jenkins-courses-for-java-and-DevOps-Programmers.html#axzz6cRYpiwdu) or Continuous Delivery scenarios.

# Gitlab flow

The most difference between GitLab Flow and Git Flow are the environment branches having in GitLab Flow (e.g. staging and production) because there will be a project that isn’t able to deploy to production every time you merge a feature branch.

**GitLab Flow** is kind of an extension to Git Flow accompanied by a set of guidelines and best practices that aim to further standardize the process.

In the Git Flow, the high-level branch is master. But here the high-level branch become production. The staging is deployed from the master branch. In between the master and the production branch, there are different branches are available for uat and pre-production.



Aside from promoting ready to deploy master branch and feature branches (same as Git Flow) it introduces three other kinds of branches:

1. Production branch
2. Environment branches: uat, pre-production, production
3. Release branches: 1-5-stable, 1-6-stable

# forking workflow

The Forking Workflow is fundamentally different from other popular Git workflows. **Instead of using a single server-side repository to act as the “central” codebase, it gives every developer their own server-side repository.**

This means that each contributor has not one, but two Git repositories: a private local one and a public server-side one. **The Forking Workflow is most often seen in public open-source projects.**

The main advantage of the Forking Workflow is that contributions can be integrated without the need for everybody to push to a single central repository. Developers push to their own server-side repositories, and only the project maintainer can push to the official repository. This allows the maintainer to accept commits from any developer without giving them write access to the official codebase.

The following is a step-by-step example of this workflow.

1. A developer ‘forks’ an ‘official’ server-side repository. This creates their own server-side copy.
2. The new server-side copy is cloned to their local system.
3. A Git remote path for the ‘official’ repository is added to the local clone.
4. A new local feature branch is created.
5. The developer makes changes on the new branch.
6. New commits are created for the changes.
7. The branch gets pushed to the developer’s own server-side copy.
8. The developer opens a pull request from the new branch to the ‘official’ repository.
9. The pull request gets approved for merge and is merged into the original server-side repository.