Build and Release along with GIT Strategies

Introduction:

Our team would like to follow-up good practice of build,deployment, development and release management process. As a part of this, we would like to use GitLab, Jenkins and python Boto:(using Python to automate AWS services) for production ready branches.

GIT Strategies:

We have hosted our repos at gitlab and it holds two kinds of branches:

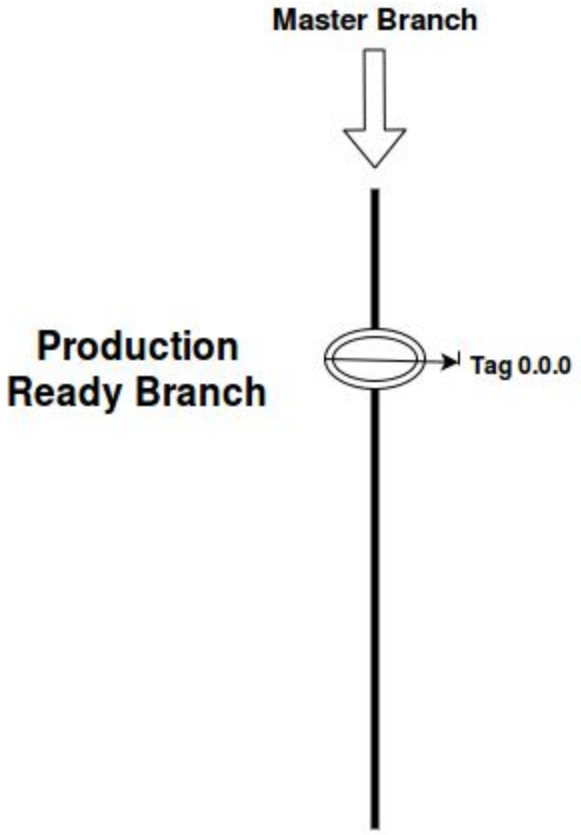
1. Main Branches (Permanent)
2. Supportive Branches (Temporary)

The main branches:

- Master

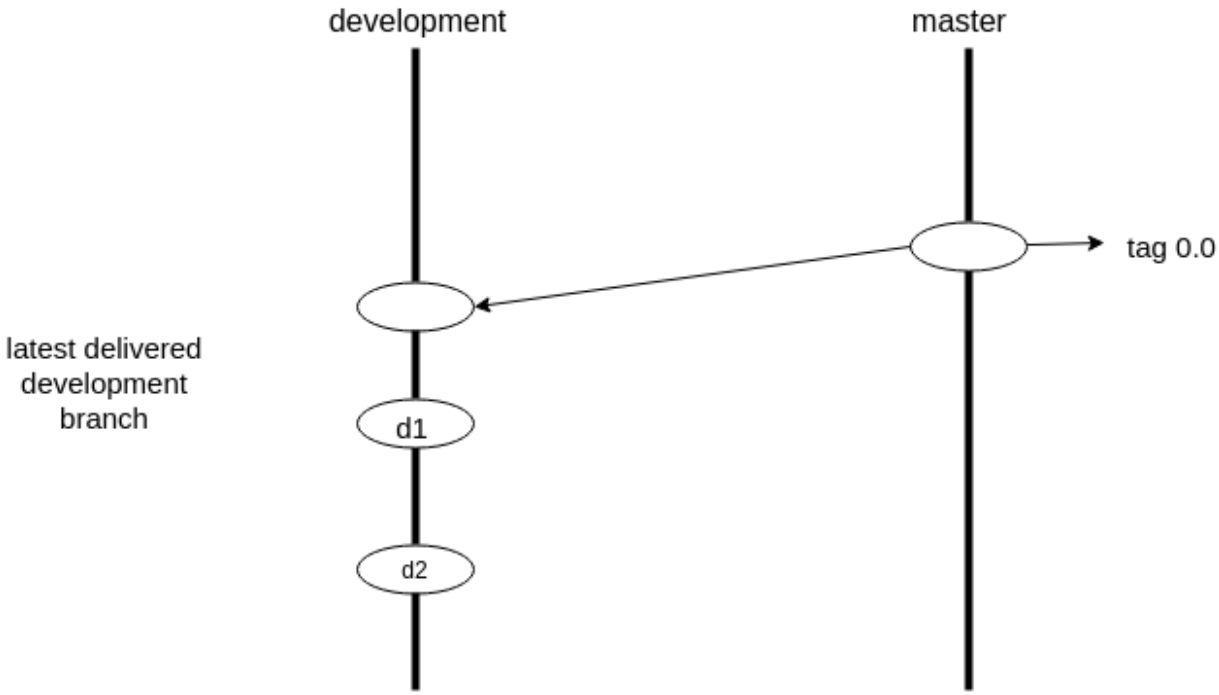
- Development

Master Branch: Main branch where the source code of HEAD always reflects a production ready state.



Development Branch: Main branch where the source code of HEAD always reflects a state with the latest delivered development

changes for the next release.



Supporting branches:

- Feature Branches
- Release Branches
- Hotfixes Branches

Here name implies itself a lot, these branches support us in perspective of continuous code collaboration, clean deployment helps us in fixing production bug, pre releases ready branch and many more.

- Tracking of features
- prepare for production releases
- fixing live production Issues.

Let's spend some time on each branch and be ready to perform some basic git operations, Each of the given branches are very particular to a specific scenario and Be careful while doing the git operations like which branches, may be their originating branch and which branches must be their merge targets.

Feature branches: Use whenever you are adding new features

Branch should be off from:
    Development

Must merge back into:
    Development
Branch naming convention: **feature**

Git operations on Feature Branch

```
$ git checkout -b featureName development

$ git checkout develop

$ git merge --no-ff myfeature

$ git branch -d myfeature

$ git push origin develop
```

Note: Always use --no-ff while merging.

Release Branches: Code is making ready for new releases for production.It allows for small bug fixes and preparing meta-data for a release (version number, build dates, etc.). By doing all of this work on a release branch, the develop branch is cleared to receive features for the next big release.

Branch should be off from:
    Development
Must merge back into:
Development and **Master**
Branch naming convention: release-*

GIT Operations on **Release Branches**

```
$ git checkout -b release-1.2 development
$ git commit -am " version number to 1.2"
```

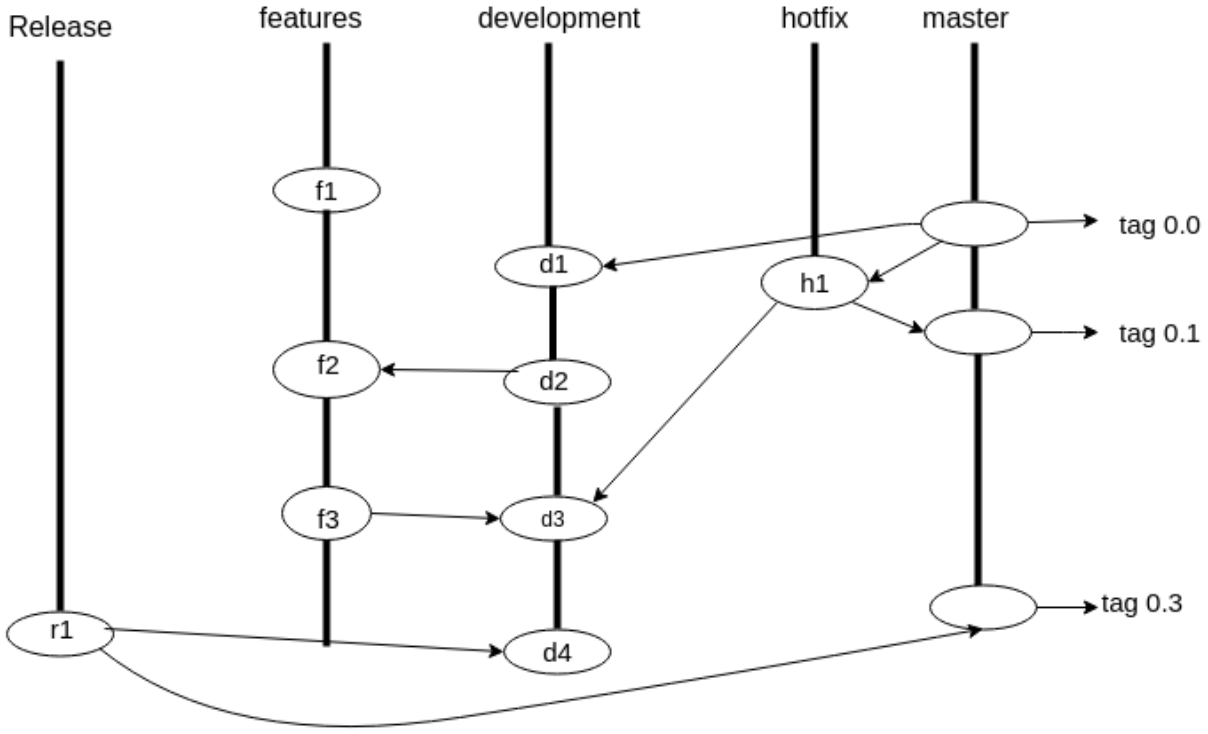
Merging release branch to **master**

```
$ git checkout master
$ git merge --no-ff release-1.2
$ git tag -a 1.2
```

Hotfix branches : Handles production issues

```
Branch should be off from:
master
Must merge back into:
develop and master
Branch naming convention:
hotfix-*
```

Combined git strategies along with all the branches



Jenkins Jobs: (Let take a look at our existing Sciex Project)

Commit Job:

SciEx has four projects (as of today) in git (iOS, Android, web, sciex-core).Each of these shall have a commit job. This job shall run, on every commit. This job will do compile (for non-nodeJS) and then run the unit tests. Technically this job should pass all the time (If it's failing, then dev has committed the changes without unit testing or compiling).

Nightly Job:

For each of the components in the sciex, we shall have nightly job. This job shall run every night for each of the components. Each of the components shall have the nightly job. The job would do the compile (for non-nodeJS), run the unit tests and creates a package.

Integration Job:

The integration job would a downstream job for the 'nightly' job. This job shall run when all the nightly jobs are completed and their respective packages

are ready. This job shall build a single package out of the different component and deployable in the dev environment. Once the package is deployed in the 'dev' environment, this job shall run the set of automated tests against the build.These tests shall be E2E (End-to- End) and if they pass, this build is eligible for the QA to take the build.

QA Job:

This job is manual job; this would deploy the successfully built and tested package from the dev environment to qa environment. The QA would then run the automated / manual tests on this build. The testing team to file bugs for this build. They make take new build available from the dev after successful 'integration' job.

Staging job:

This is a manual job. This shall deploy the QA certified build from the qa environment and then team would do testing both manual and automated. This build shall be eligible for the release for the production.