1. **What will be the git branching strategy, elaborate why & why not with pros and cons?**

**Why do we need a Branching strategy? What is it?**

[Git Flow](https://nvie.com/posts/a-successful-git-branching-model/) describes multiple branches for development, releases and the orchestration between them. There are even [scripts and extensions](https://github.com/nvie/gitflow) provided to help use/maintain Git Flow.

* Need a set of rules/strategy to commit, merge and promote changes to a repository. This defines a git collaboration workflow.
* Let’s you define a structured delivery of work, increasing efficiency & reducing error
* Introduce opportunity for code review and protected branch.
* Exploring strategies to reduce merge conflicts
* Clean your working directory before doing a merge
* If you have work in progress, either commit it to a temporary branch or stash it
* Commit often, and merge often
* Create more of small topic branches rather than long running branches
* Push/pull often, publish your work often
* Rebase your work than merging only on your local repository and local branch.
* Separate different types of branches like: master, hotfixes, release, develop, feature & topic branches.
* Keep changes physically small. Small diffs, small amount of lines changed (easier to debug)

**Pros ?**

* 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.
* Works well with in Continuous Integration or Continuous Delivery scenarios.
* Can release more often, including multiple times per day.
* Much tighter feedback loop for developers, business, and end-users.
* Run automated tests on all branches.
* Deploy depending on the exact strategy — it may be from the master branch, from the production branch, or from multiple branches.
* Ensures a clean state of branches at any given moment in the life cycle of project

**Cons ?**

* Git Flow can slow things down when having to look at large pull requests if you are trying to iterate quickly.
* It isn’t recommended when it needs to maintain single version in production.
* feature conflicts arises on environment branches which already has other in development features.

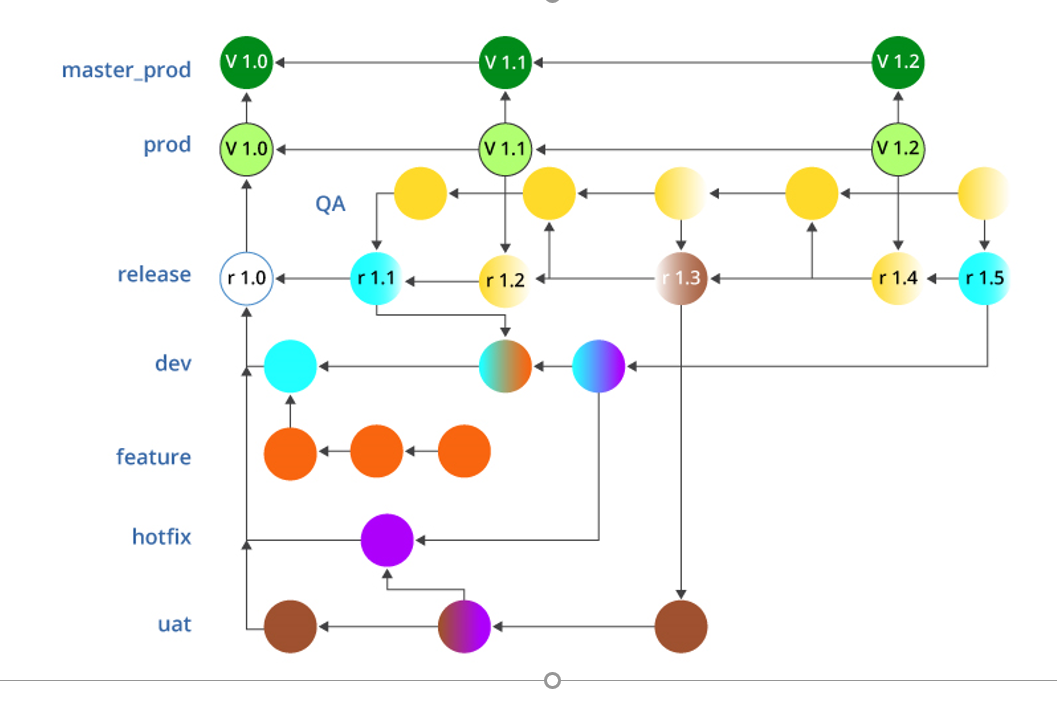
**git branching strategy**

A good branching strategy is the one that adapts to your project and business needs. Every organization has a set of its own defined SDLC processes.

**First we need to Develop and decide a Branching Model considering below 6 points**

* Decide on the VCS and the type of model architecture
* Select the git workflow model
* Select a branching strategy
* Create branches and define guidelines
* Define member roles and permissions
* Protect branches on Jenkins/gitlab and maintain branching guidelines

**An example branching structural strategy that I have used in my one of project:**



* **Diagram:** Branching strategy
* Clone the project available at github:
* git clone http://github.com/suman/XXXX.git

**Branches Brief explanation:**

* “master-prod”: Accepts merges/code/commits only from the “prod” branch
* “prod”: Perform only a merge --squash from “release” branch.
* Merge only when approved by “QA”
* Tag every merge in the format: v1.0, v1.1 … v1.\*
* “release”: merge from the branches “dev”, “uat”, “QA”.
* Every release commit/project code version has to be approved by “QA”.
* Tag every merge in the format: r1.0, r1.1 … r1.\*
* “dev” and “uat” never merge with each other.
* “hotfix” branch commits are shared among any feature branches such as “dev” and “uat”
* “feature” branch is private to “dev” alone and is dropped after merging.
* CI/CD DevOps tools can be used to automate the above development and deployment to master\_prod.
* Every project release: r1.0 .. r1.x on the ‘release’ branch can be tracked by Jenkins CI tool and will trigger a build, on a successful build continuous testing suite
* cases will be triggered on the code. If the test passes the release will be delivered to ‘prod’ branch.
* Every source code delivered to ‘prod’ branch will be automatically deployed to ‘master\_prod’ branch.

All the steps will be mentioned in a Jenkins file on a branch ‘name’ condition.

1. **Which tool you will consider for CI/CD**

Considering DevOps to be an ideology towards achieving a quality product, every organization has its own guidelines and approach towards it.

Some of the popular tools I have used are:

GIT, SVN, Perforce - Version control tool Docker - Containerization tool

Github and BitBucket - Version Control hub Kubernetes(K8’s) - Orchestration tool

Jenkins - CI/CD tool AWS, GCP - Cloud based service

Maven - Build tool Linux RHEL6/RHEL7

SonarQube - Code Analysis tool Shell scripting, Groovy scripting, YAML

Sonatype Nexus - Artifact repository Virtualization – VMware, Hyper-V

Apache Tomcat, Nginx - Application server Nagios, EFK, Prometheus – Monitoring Tool

Apache HTTPD – Web server C/C++, JAVA - Programming Language

JIRA – Defect Tracking Tool K8s Package Manager - Helm Charts

Ansible, PUPPET - Configuration management tool Docker Registry – Image Repository

1. **What will be your build promotion plans (Dev - QA - Prod)**

**Build**:  This job includes the configuration for the building project, job triggers, scm location, jdk version to use, maven goals, artifact upload to repo like Nexus, Artifactory, and  email notification.

**Test**: This job can call test suites and decide to call a downstream job or not.

**Dev Deploy**: Simple job with a trigger to the Promotion Job if the deployment was successful.

This job can call the script to perform a deployment or use tools like Jenkins, GitLab, Bamboo, teamcity or IBMUrbancode.

**QA Promotion**: This job includes a send email notification to the person/group responsible for approval. The email contains a link for promotion and an optional comment for approval notes.

The Promotion Email link can look like this:  [~~http://localhost:8080/job/SampleAppQApromotion/1/promotion/~~](http://localhost:8080/job/SampleAppQApromotion/1/promotion/)

Once approved, the deploy job will run and If you are a Jenkins Admin or have global privileges, you will see Force Execution Option.

After Re-execute Promotion Jenkins Admin will see the Force Promotion option and In Pipeline, the star icon will show that a particular build is promoted and by which user.

**QA Deploy:**  QA environment deployment job. This job can call a script to perform a deployment or use tools like Bamboo or Urbancode.

We can chain multiple Promotion Jobs and Deploy jobs to accomplish the need for another environment, for e.g**. Dev >> QA >> Prod**

1. **Provide CI/CD implementation plans with stages**
2. **How will you manage module version dependencies**