Workflows for Git

Curso de Ingeniería de Software
Primer Semestre 2022

UNIVERSIDAD DE COSTA RICA

SR-CIE
Carrera de
Informática Empresarial
Sedes Regionales

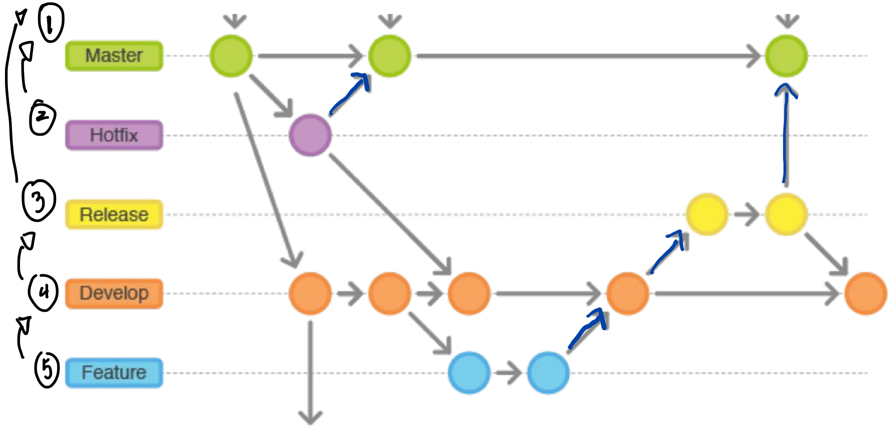
Contenidos

Git Flow

Git Flow

- **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:
  - **feature-*** — feature branches are used to develop new features for the upcoming releases. May branch off from **develop** and must merge into **develop**.
  - **hotfix-*** — hotfix branches are necessary to act immediately upon an undesired status of master. May branch off from master and must merge into **master** and **develop**.
  - **release-*** — release branches support preparation of a new production release. They allow many minor bug to be fixed and preparation of meta-data for a release. May branch off from develop and must merge into **master** and **develop**.

Git Flow



Git Flow

- **Advantages**
  - Ensures a clean state of branches at any given moment in the life cycle of project
  - The branches naming follows a systematic pattern making it easier to comprehend
  - It has extensions and support on most used git tools
  - It is ideal when there it needs to be multiple version in production
- **Disadvantages**
  - The Git history becomes unreadable
  - The master/develop split is considered redundant and makes the Continuous Delivery and the Continuos Integration harder
  - It isn’t recommended when it need to maintain single version in production

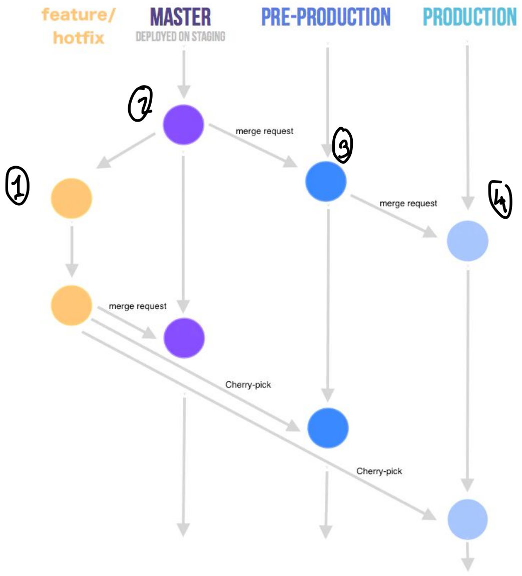
GitLab Flow

GitLab Flow

1. Use feature branches, no direct commits on **master**
2. Test all commits, not only ones on **master**
3. Run all the tests on all commits (if your tests run longer than 5 minutes have them run in parallel).
4. Perform code reviews before merges into **master**, not afterwards.
5. Deployments are automatic, based on branches or tags.
6. Tags are set by the user, not by CI.
7. Releases are based on tags. $\rightarrow$ V1.1 V.0.1
8. Pushed commits are never rebased.
9. Everyone starts from **master**, and targets **master**.
10. Fix bugs in **master** first and release branches second.
11. Commit messages reflect intent.

GitLab Flow

-


feature/ hotfix
MASTER
DEPLOYED ON STAGING
PRE-PRODUCTION
PRODUCTION
merge request
merge request
merge request
Cherry-pick
Cherry-pick
1
2
3
4

GitLab Flow

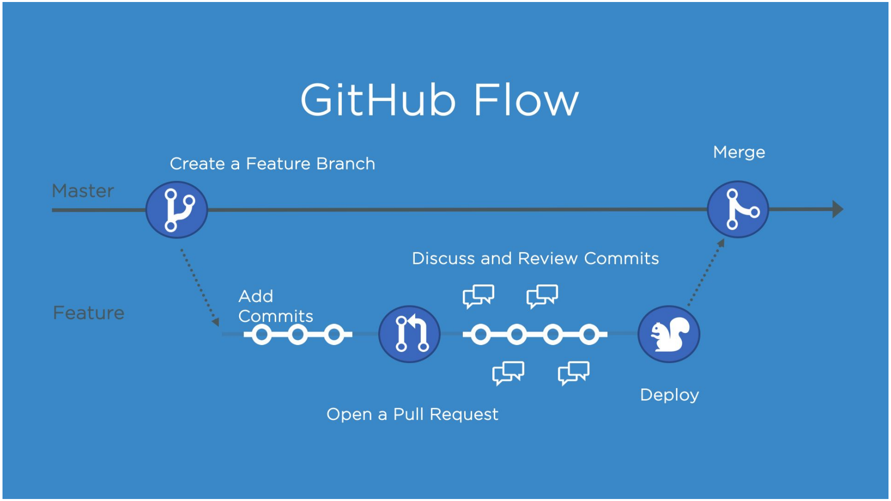
- **Advantages**
  - It defines how to make the Continuous Integration and Continuous Delivery
  - The git history will be cleaner, less messy and more readable (see why devs prefers squash and merge, instead of only merging, on this article)
  - It is ideal when it needs to single version in production
- **Disadvantages**
  - It is more complex that the GitHub Flow
  - It can become complex as Git Flow when it needs to maintain multiple version in production

GitHub Flow

GitHub Flow

1. Anything in the **master** branch is deployable
2. To work on something new, create a branch off from **master** and given a descriptively name(ie: new-oauth2-scopes)
3. Commit to that branch locally and regularly push your work to the same named branch on the server
4. When you need feedback or help, or you think the branch is ready for merging, open a pull request
5. After someone else has reviewed and signed off on the feature, you can merge it into master
6. Once it is merged and pushed to **master**, you can and should deploy immediately

GitHub Flow


GitHub Flow
Create a Feature Branch
Merge
Master
Discuss and Review Commits
Add Commits
Feature
Deploy
Open a Pull Request

GitHub Flow

- **Advantages**
  - it is friendly for the Continuous Delivery and Continuous Integration
  - A simpler alternative to Git Flow
  - It is ideal when it needs to maintain 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

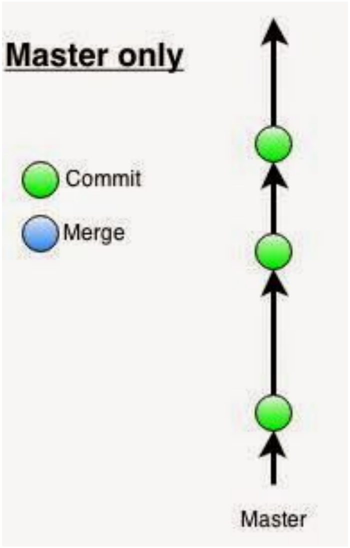
Master-only Flow

Master-only Flow

1. Each feature or hot fix is worked on in dev environment that is similar to production, that allows business owner direct access for testing and approval. Changes are committed locally.

2. Once approved by the business owner, commit and push changes to master on origin, and then deploy to production immediately.

Master-only Flow

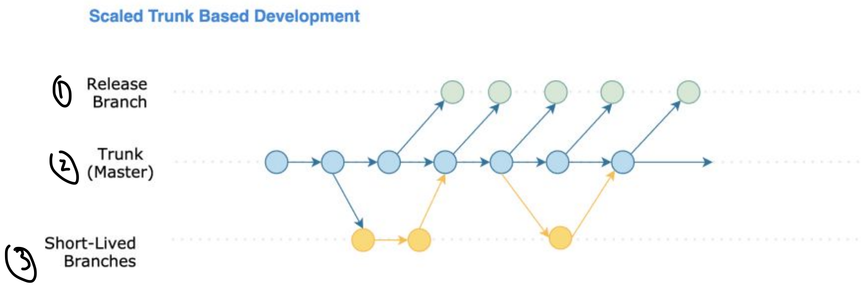


Trunk-based development

Trunk-based development

1. No existen branches de larga duración. No se le da mantenimiento a ningún branch.
2. Se debe hacer commit al menos una vez al día (esto no significa que vamos integrar cualquier código solo por hacer commit, el siguiente punto lo explica mejor). Esto lo que busca es eliminar la distancia entre los desarrolladores cuando se empieza a codear cosas nuevas.
3. Todo lo que se le haga commit es código funcional, esto implica que se cumpla la definición de hecho (definition of done), se hayan creado las pruebas necesarias y todo lo que sea requerido para asegurarse que el código no está introduciendo un bug. Esto significa que el equipo debe de tener un grado de madurez y responsabilidad alto al momento de entregar el código.
4. El trunk siempre debe encontrarse en un estado verde y óptimo con esto quiero decir listo para hacerle release.

Trunk-based development



Referencias

Porto Patrick. 4 branching workflows for Git. Descargado de https://medium.com/@patrickporto/4-branching-workflows-for-git-30d0aaee7bf

Alvarez, Mariano. ¿Por qué Trunk-Based Development? https://dev.to/malvarezcr/por-que-trunk-based-development-i5n