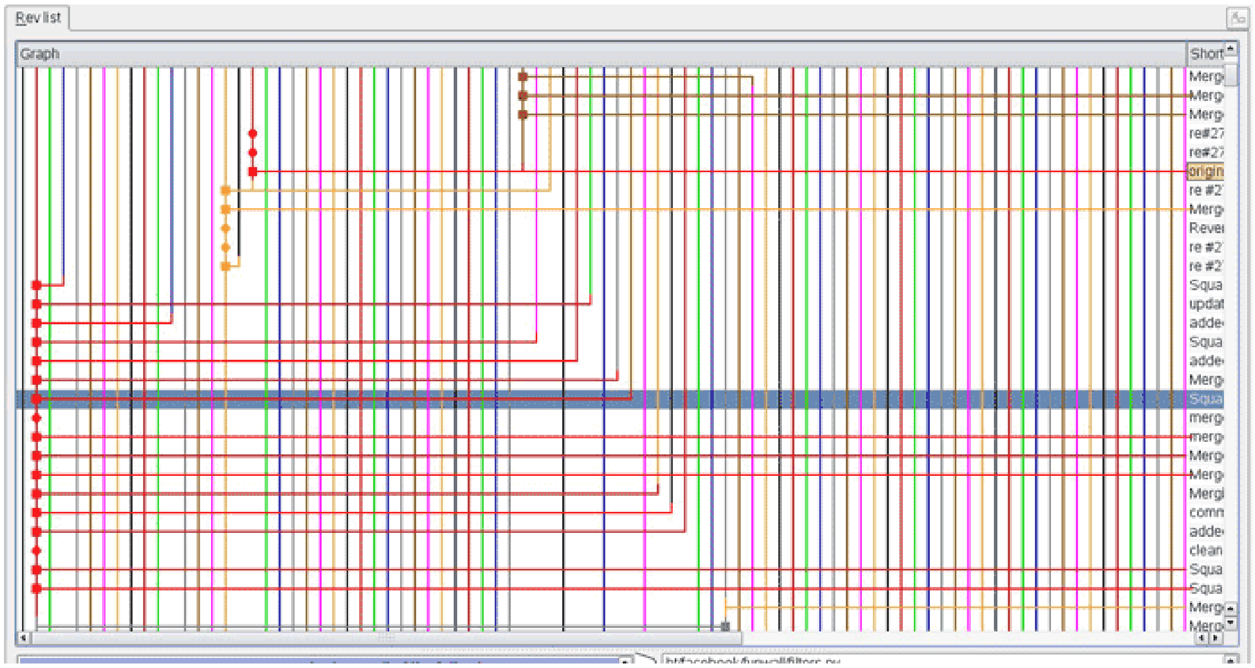
[AT Ralph] Git Practice 2

Target: Practice "A Successful Git Branching Model"

For More: Practice Github flow

Git Branch Management Overview:

Are your branches like this?

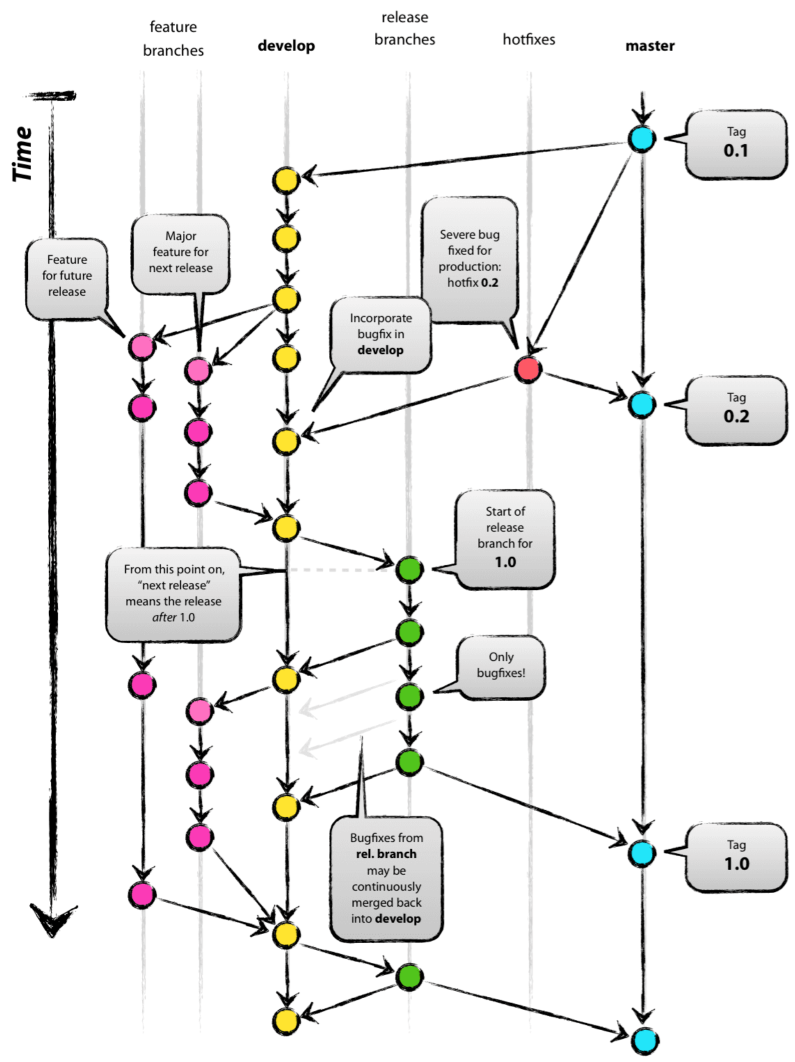


We need approach to manage our git branch, what's the best way?

Basic theory:

A successful git branching model - Vincent Driessen

http://nvie.com/posts/a-successful-git-branching-model/

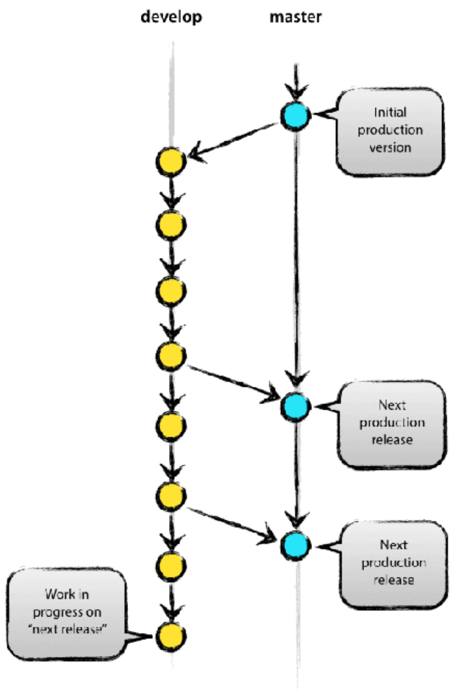

feature branches
develop
release branches
hotfixes
master
Time
Tag 0.1
Feature for future release
Major feature for next release
Severe bug fixed for production: hotfix 0.2
Incorporate bugfix in develop
Tag 0.2
Start of release branch for 1.0
From this point on, "next release" means the release after 1.0
Only bugfixes!
Bugfixes from rel. branch may be continuously merged back into develop
Tag 1.0

Three Git Branching Approach:

1) Git Flow

Long time branch: master, develop

Short time branch: feature, hotfix,release


develop
master
Initial production version
Next production release
Next production release
Work in progress on "next release"

The advantage is clear and easy control, disadvantage is too complex, need to maintenance two long time branches. Most of tools will assume master as default branch, but gitflow approach is doing developing in develop branch. Need to switch branches very often, sometime annoying.

The bigger problem is, this flow is based on "version release", the target is have new version in some period. However, many website project such as blog is "continue deliver", if any code change, deploy once. In this case, the master and develop branch has no bigger difference, no need to maintain two long term branches.

Practice 1:

Practice with the git flow regarding develop/staging/production/post release

[redacted]

The command you may need:

git add .

git commit -m "hello"

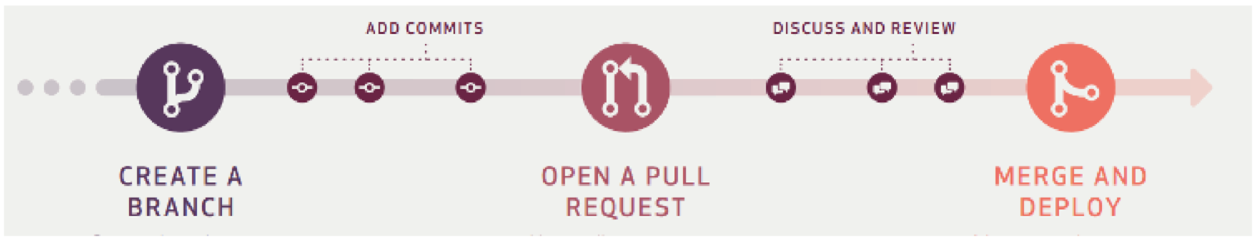
git branch

git checkout

git push origin master:master

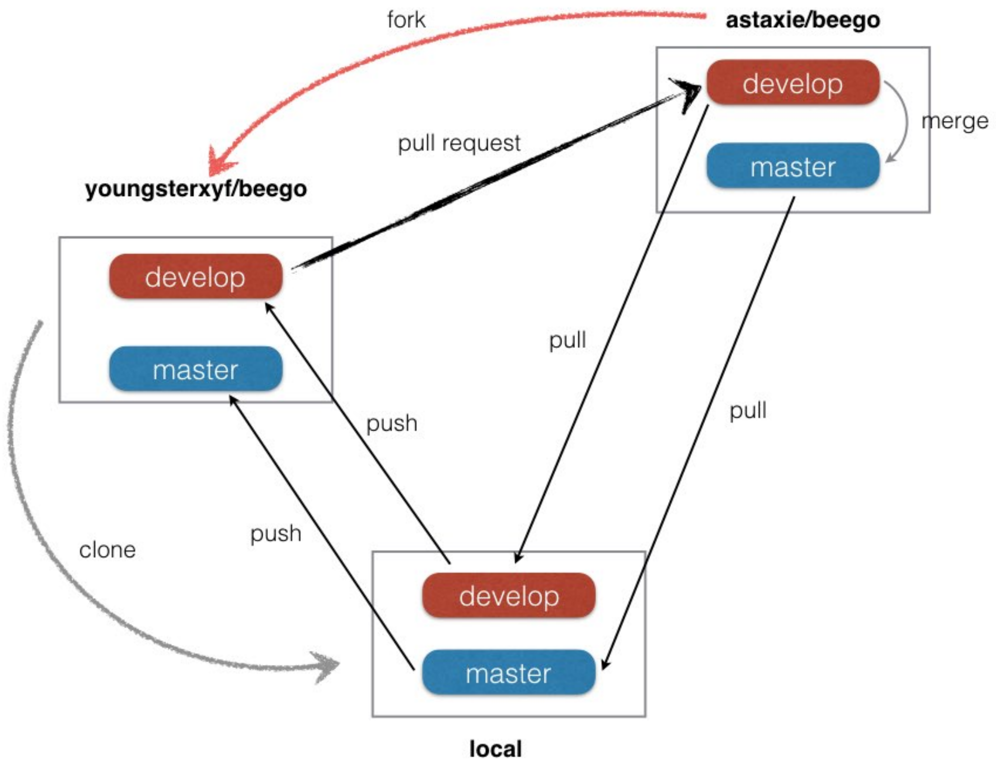
2) Github Flow

Best for continue deliver project



GitHub Flow is a lightweight, branch-based workflow that supports teams and projects where deployments are made regularly. This guide explains how and why GitHub Flow works.

https://guides.github.com/introduction/flow/



Only need one long time branch: master

1. according to requirement, pull new branch from master. Not need feature or path branch
2. when complete develop on new branch, or when need to discuss with the owner, create a "pull request"
3. pull request is a notice, let others know you, it is also a kind of communicate mechanism, let everyone check and discuss with your code, during the discuss, you can still commit code
4. if you pull request accepted, merged into master, deploy, you forked old branch should be removed

Practice 2:

Practice the Github flow:

Fork the project from https://github.com/Equilar-Insight/Automation_Practice master

Add folder with your name in your local, enter the code you have done in last several sessions.

Then Create "New pull request" to Equilar-Insight Automation_Practice project.

3) Gitlab Flow

Recommand by gitlab.com, Tim implemnt gitlab to equilar too. [redacted]

For example: google chromnium project

Google Chromnium Project

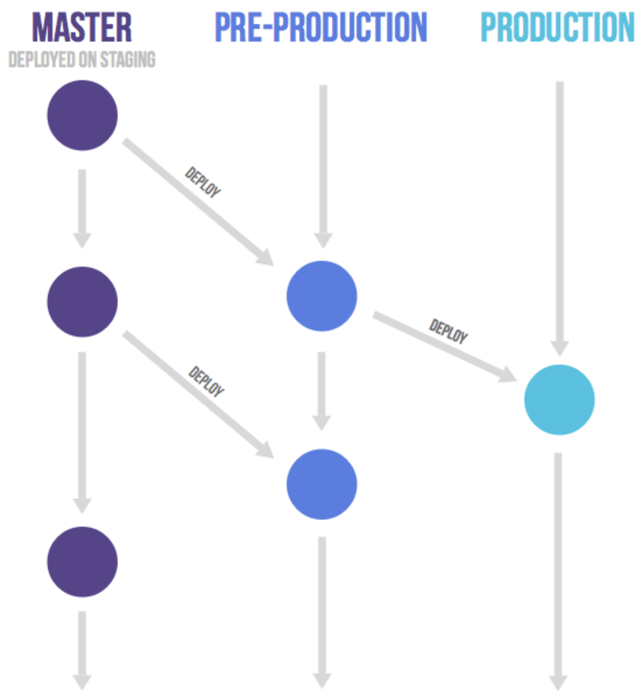
Goals

Our goal with the "Upstream First" policy is to eliminate kernel version fragmentation.

We know that hardware partners support other devices as well as Chromium OS-based devices. So rather than having partners target multiple kernels for multiple devices, we'd like to encourage them to target one kernel: the upstream kernel.

By having everyone target a single kernel, we hope to avoid duplication of effort and get everyone (our hardware partners, their partners, and the upstream community) working together on improving a single set of patches. By working together, we can build a better kernel and avoid later having to rework changes that are incompatible with the upstream kernel.

https://www.chromium.org/chromium-os/chromiumos-design-docs/upstream-first



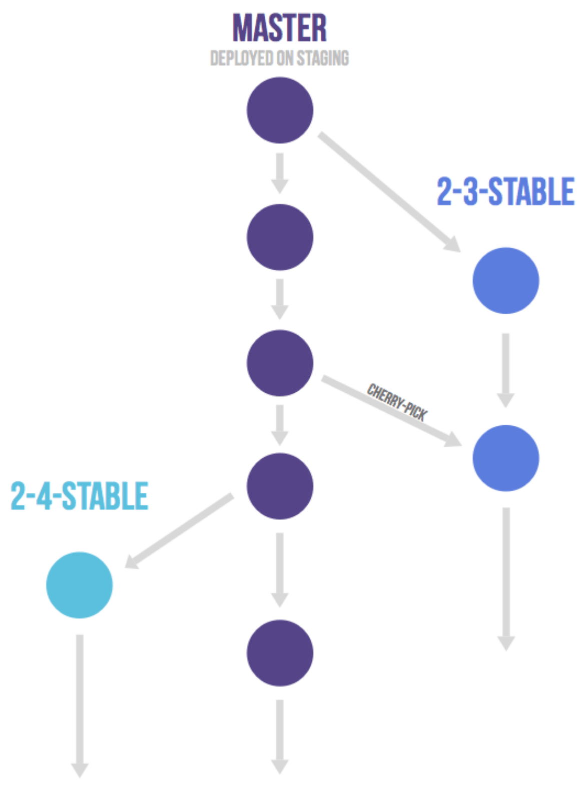
For continues deliver project, suggest to create different environment branches. For example "Develop Environment" is master, "Staging Environment" is pre-production, "Production Environment" is production.

Develop branch is upstream of pre-production, pre-production is upstream of production. The code change, must be from upstream to downstream.

For example, if production has bug, now need to create a new function branch, and merge into master, if no problem, then do cherry-pick to pre-production, only if no problem for all steps, can move to production.

Only if emergency situation, can skip upstream, directly edit downstream.

Release:



When release, suggest for every stable version, create a new branch from master, for example: 2-3-stable branch and 2-4-stable branch.

Only when doing bug fix, allow to merge to those branch and update the small version number.

4) Other flow

Such as gg, if you interested, https://github.com/Fmajor/gg

No limited.

Supplements:

- How to use a scalable Git branching model called git-flow (by Build a Module)
- A short introduction to git-flow (by Mark Derricutt)
- On the path with git-flow (by Dave Bock)
- (github flow) Fork A Repo - User Documentation
- (github flow) Using pull requests
- (github flow) Creating a pull request
- Git Pro