Branch Strategies

What? Why?

How do we use SCM to effectively manage deployments?

2 Approaches

Branches VS NO Branches

With branches

- isolation
- code review
- decomposition

Challenges:

- hard to maintain
- tend to be messy
- slow down development
- delayed feedback and integration

No branches *

- (*) you still have main branch
- fast development (direct commits)
- fast feedback and integration
- easy deployment strategy
- simplicity

Challenges:

- easy to break things
- more responsibility (it’s pros too!)
- needs rigorous automatic testing

What do we choose?

- be “safe” and slow
- be “broken” and fast

Branch strategies pitfalls

Long living branches

They are:

* hard to review (no context as time goes by)
* tend to have a lot of changes (a dozens of files)
* inconsistent with a main branch
* not integrated (sandboxes and laptops) - it works for me!

Branches and environments

- Branches are NOT environments
- Branches *could* be used for dedicated environments deployments
- dev => dev, test => test, prod => prod - just makes things messy
- excessive branches , KIS

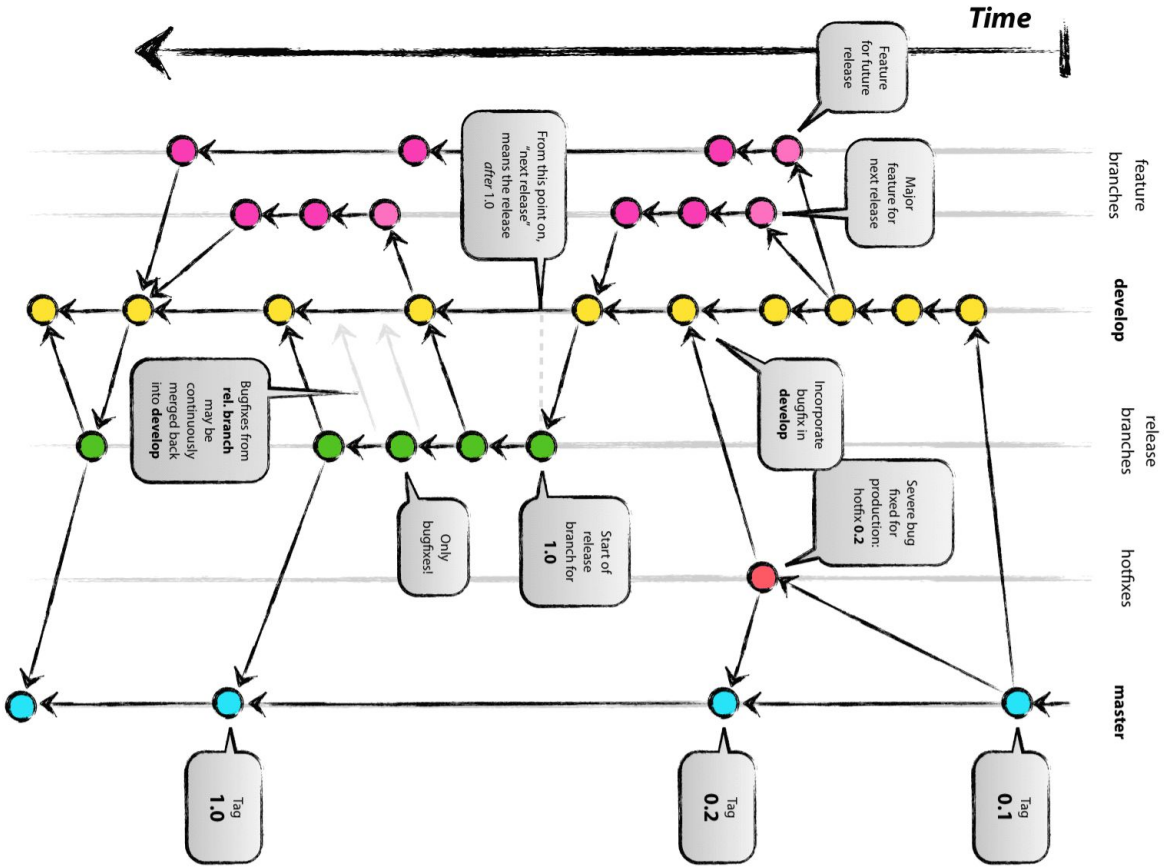
Main branch is broken

- I’ll fix it later
- We don’t have a reference build and app
- We not sure about application state
- Tests fails - let’s skip them

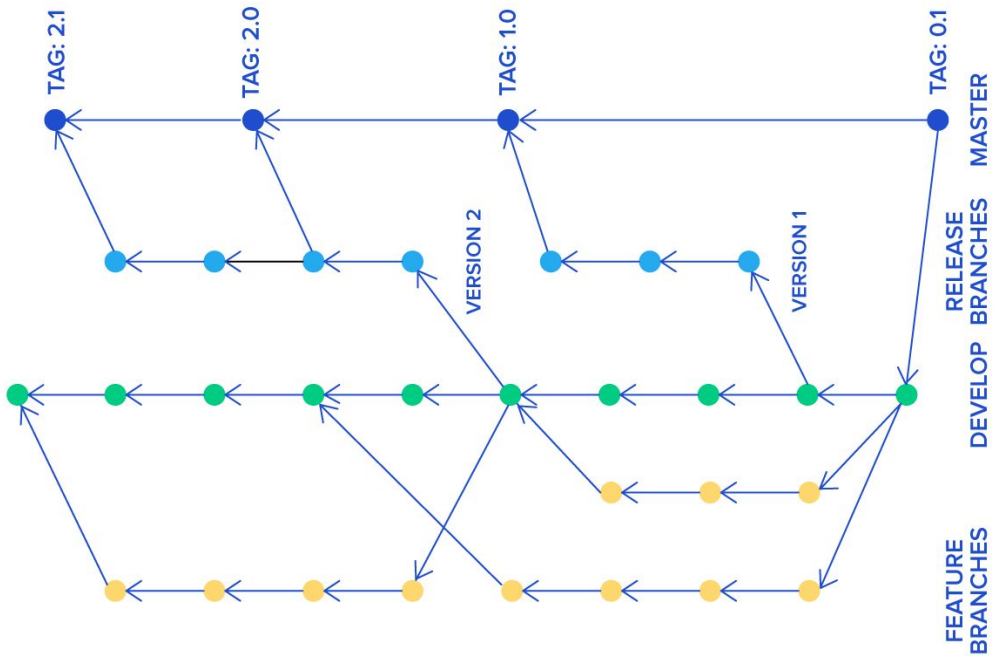
GitFlow Strategy

- By Vincent Driessen, 2010
- Dedicated branches for dedicated workflows (feature branches, development, release, hotfixes)
- “With branches” strategy
- Works well if we have big enough team to split by different flows - development, release maintenances, and bug fixing
- Has a lot of (long living) branches and a lot of merges - require a lot of discipline and dedicated person to look after
- Relatively slow

GitFlow


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

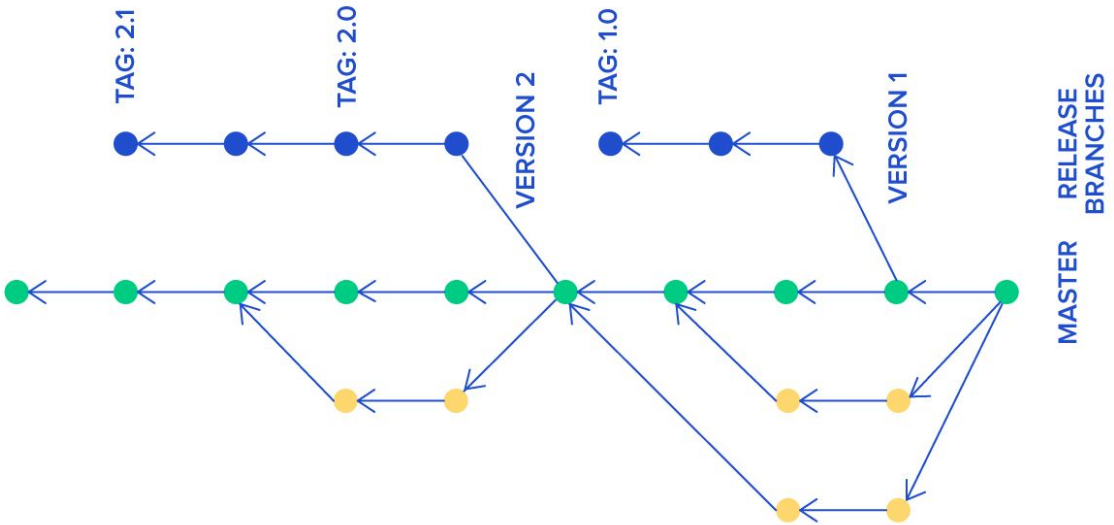
GitFlowish Models



Trunk Based Development

- one main and stable branch
- no other long living branches - no merge hell
- direct commits to main or frequent PR from short living branches
- easy to break - easy to fix - responsibility - roll back system - a lot of tests
- GitHubFlow development - similar
- release branches - optional or release always from trunk
- code reviews as whole source code or code review in PR or pair programming
- reproduce bugs on main and cherry pick commits (back to release branch)
- strict code style, mature team

Trunk Based Development Model



Mitigating TDD challenges

- fast delivery
- branches by abstraction VS branches
- everything in a main branch!

Rollbacks

- Available in both GF and TBD using tags artifacts.
- Hase nothing to with branches
- Branches to go back to a certain point of source code

Q/A