Branching Strategies & Trunk Based Development

codurance

Branching Strategy

How teams work together in the context of a version control system

Defines how to use branches to achieve concurrent development

Enhance collaboration, efficiency and accuracy in the software delivery process

Use cases that should be supported

Typical day-to-day workflow: standard development and delivery

Emergency hotfixes: changes to resolve an urgent problem ASAP

Small vs big changes: even if the standard promotes small batches, sometimes you have to deal with big changes

Experiments vs standard code: when you need to share code but don’t want it into production

Merge conflicts

Merge conflicts can happen when we trying to integrate changes from several branches into one (i.e. main), usually in order to release.

Those changes are contradictory in some way or another and cannot be merged to be deployed. Or if deployed can cause difficult to track bugs.

Typical reasons:

- Changes affecting to the same files in different branches
- Removing or modifying code in a branch that it’s used in another one

Some conflicts can be resolved automatically by Git, but most of the time it will need our help

Branching Flows

GitFlow

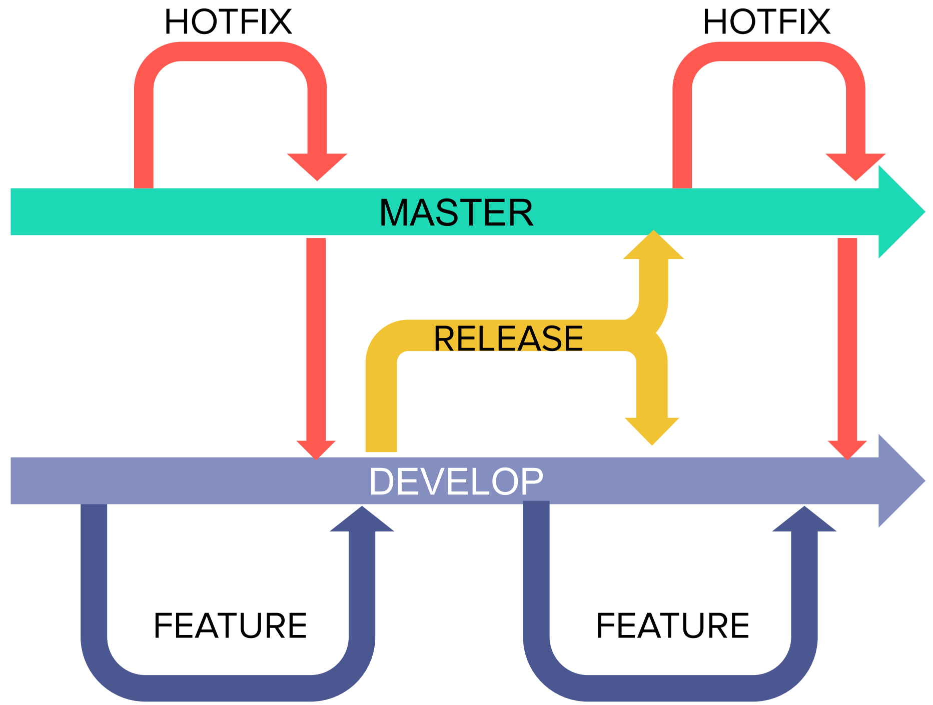
Two branches with infinite lifetime:

- **Main/master**: HEAD reflects production-ready state
- **Develop**: HEAD reflects changes for next release

When stable, changes in Develop should be merged into Main

Parallel development with supporting branches:

- **Feature**: develop features
- **Release**: preparation for production release (release-*)
- **Hotfix**: act immediately (hotfix-*)



codurance

Git Flow Branches

Trunk (main or master): default first branch. Is the one that receives all the changes that will be released to production. Infinite life.

Development (develop): holds all the changes, but they doesn’t go directly to prod. Integrates changes with trunk. Infinite life.

Feature branch: usually contains the changes that conform a feature (can be short or long lived)

Release branch (release-*): contains a set of changes that are intended to go to production from develop.

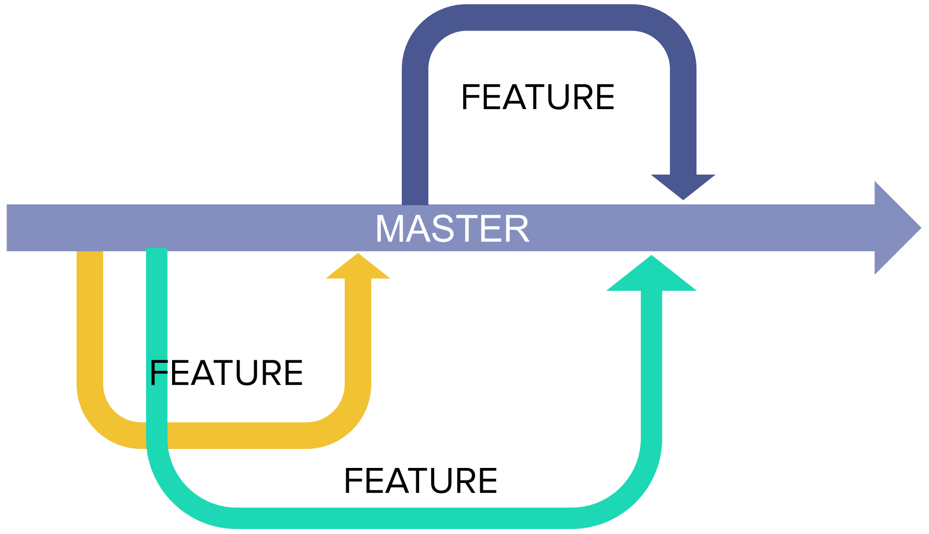
Hot fix (hotfix-*): short lived branch with the changes to fix an urgent problem. Extracted from trunk.

GitHub Flow

Trunk: always releasable (direct release)

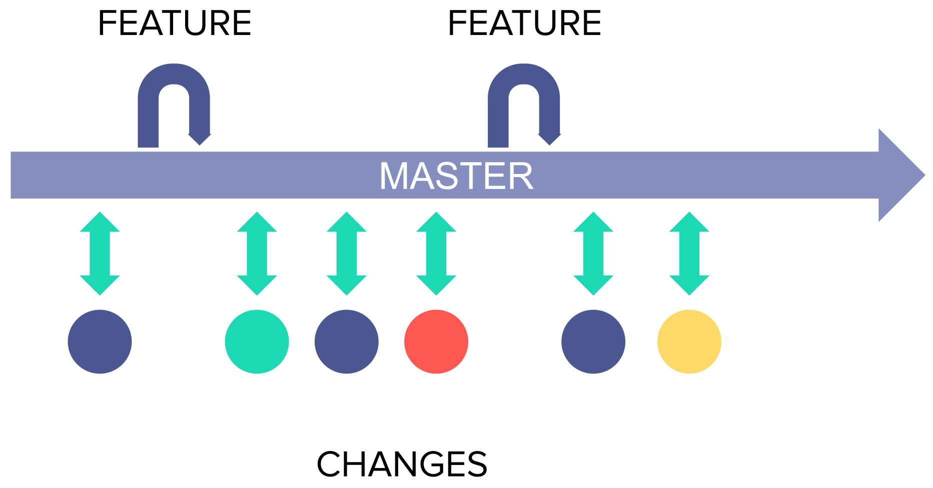
Feature branches: developers work. Conflicts can arise if branches live for too long

Release branch can be used, but it's not mandatory



Only Trunk Branch: always releasable (direct release), all changes go to trunk and are integrated at least once everyday

Short lived (hours) feature branches or local branches are allowed for convenience



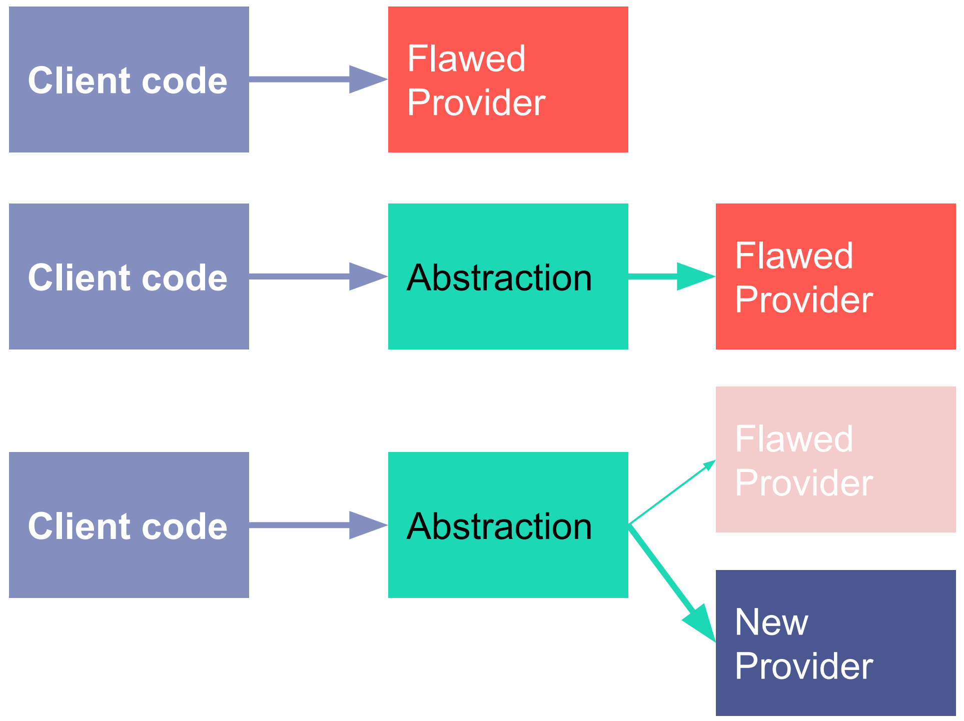
Branch by abstraction

Introduce an abstraction to isolate code that has to change, so the rest of the system is unaware of the details.

New code is not initially wired. You progressively move functionality from the old flawed code to the new.

You can use Feature Flags to control what parts of the code are used in production.

When all functionality is moved, remove the old provider.



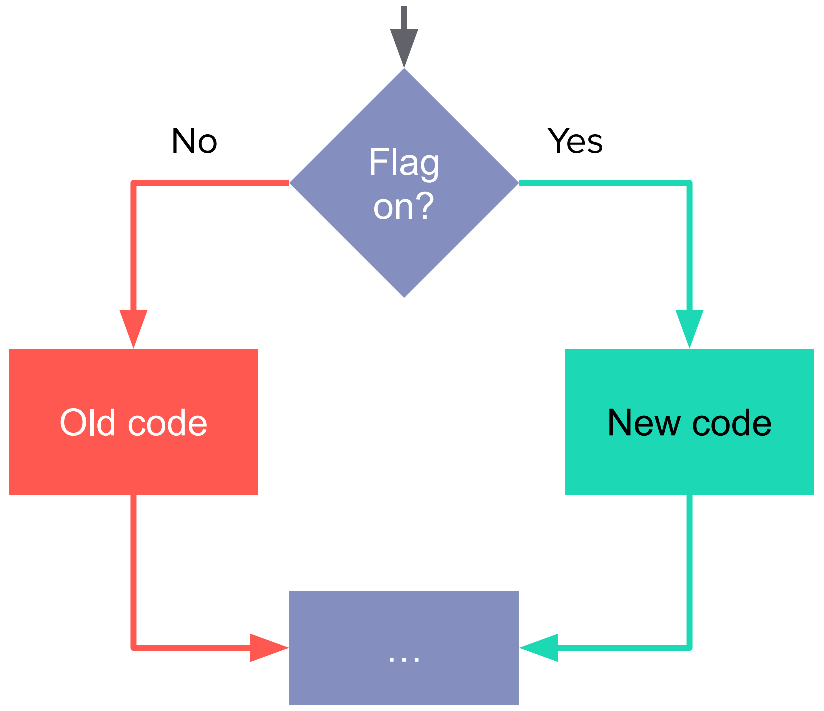
Feature flag

Flag to allow or disallow visibility of deployed changes that can be externally configured.

You put calls to new and old code in the branches of a conditional structure that checks a certain flag.

This way you don't need to re-deploy to activate or deactivate features or code.

You can try some new code and revert immediately if something goes wrong.



Exercise on branching

You have a code base using a library that you need to replace

You have calls to that library from 2 or 3 places

Branch by abstraction:

Your work is to replace that library with a new one of your own without breaking the build (all tests must pass)

Feature flag:

You need to provide two alternative implementations that can be switched using a feature flag