Git: Branching Strategies

|  |  |
| --- | --- |
| Author: | Ian Addison |
| Document Name: | Git: Branching Strategies |
| Document Location: | \\lukpc210\Projects\Product Projects\Operations\Vitic 2017.1\Project Documents\POC\sprint\_anchorman\Git Branching Strategies.docx |
|  |
| Date Created: | 31/01/2017 |
| Date Last Modified: | 31/01/2017 |
| Status: | Draft |
| Confidentiality: | <Select/enter confidentiality status> |
| Circulation: | <Detail circulation list> |
| Document Version: | 0.1 |

Contents

[1. Overview 2](#_Toc473625332)

[2. Feature Branching 3](#_Toc473625333)

[3. Feature Toggles 4](#_Toc473625334)

[4. Worked Example 5](#_Toc473625335)

# Overview

This document gives an overview of two common branching strategies (feature branching and feature toggles) that can be used with Git. It covers the strategies, advantages and disadvantages, and gives a worked example.

# Feature Branching

Using this approach there are two main branches:

* **master**: This branch contains production releases. The source code at head is always in a production ready state

**develop**: The main branch for work in progress on the next release. When the project is ready for release, code on the develop branch is merged into master and tagged with release number

In addition to these two branches, there are several supporting branches:

* **feature**: A feature branch is created (from the develop branch) for each feature being developed. The branch exists for the time the feature is in development. A single feature may consist of several user stories. All development work across all of these user stories would be done on the feature branch and the branch then merged to develop once the entire feature is complete
* **release**: Used to support new production releases. A release branch will be created from the develop branch when a release is scheduled. The purpose of this branch is to allow for any last minute changes (e.g. minor bug fixes, metadata for release etc). Once these changes have been made, the release branch is merged into master and the develop branches. By using release branches, it means the develop branch is freed up so features for the next big release can be developed in parallel to the current release
* **hotfix**: A hotfix branch is created for any critical bugs found in production that require resolution immediately (i.e. can’t wait until next release). A hotfix branch is created from master and bug fixing is performed on the branch. Once the bug is fixed, it is merged into the develop and master branches

Potential problems with this solution are:

* Merge conflicts are more likely to occur due to long running feature branches. These merge conflicts can be time consuming to resolve
* Although each branch is under continuous integration, because of the long running nature of branches there is less integration across branches. This means when a feature branch is merged into develop there is a higher chance that it breaks another feature being developed in parallel to the feature being merged

A branching strategy that defines a solution to this problem is discussed below.

# Feature Toggles

The aim of this strategy is to alleviate merge problems that may be found with long running branches. The idea is that each feature has a Boolean flag which through configuration will turn the feature on or off.

As with the feature branching strategy, branches are used for new functionality but with the following main difference from feature branching:

One feature may have many branches: In the feature branching strategy, an entire feature (e.g. search for a substance) would be developed on a single branch. In the feature toggle strategy, separate branches would be created for each user story of a feature and as soon as each user story was complete the branch would be merged to master.

This means that branches are typically short lived and as a result merge conflicts are less likely to occur. However it also means that code is being merged to master before a feature is completed. This is where “feature toggles” come in, so that the feature can be turned off through configuration until it is completed and ready to go live.

When a release is to be performed, the configuration is updated so that the flags for any features to be included in the release are turned on. The code on the master branch is then tagged in preparation for the release. An advantage of this approach is that features can be turned on temporarily in order to obtain user feedback on an upcoming feature.

In this strategy, hotfixes are performed using semantic versioning.

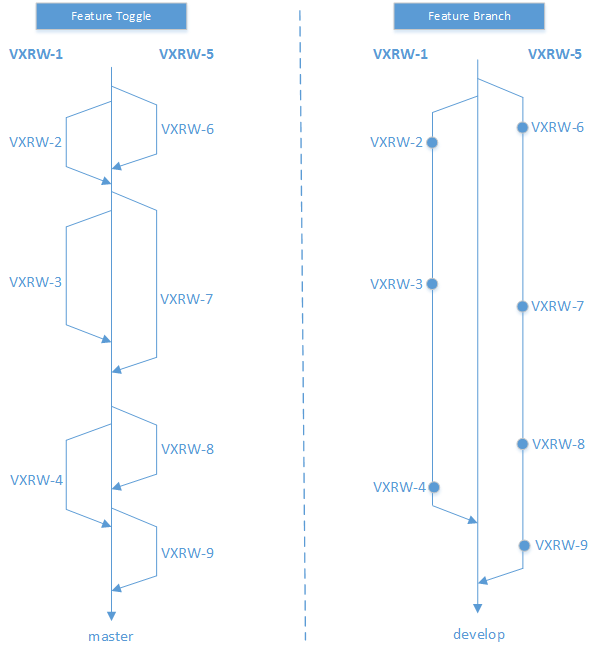
Although this solves the problem of long running branches and merging, this strategy introduces its own problems:

* Configuration and logic is required in order to support feature toggling
* Code which has not yet been fully tested and QA’d could make it into production. Code around new features that are turned off will have been tested with respect to existing functionality, but not functionality of the new feature itself
* Once a feature is completed and part of the product, the configuration for that feature needs to be removed from the code base

# Worked Example

This example assumes that 2 features are required. The “VXRW-N” refers to the ticket numbers:

* VXRW-1: Search for structure. This is made up of the following user stories:
  + VXRW-2: Search UI (frontend)
  + VXRW-3: Search controller (backend)
  + VXRW-4: DAL (backend)
* VXRW-5: Import CSV file. This is made up of the following user stories:
  + VXRW-6: Import UI (frontend)
  + VXRW-7: Import controller (backend)
  + VXRW-8: Import validator (backend)
  + VXRW-9: DAL (backend)

The diagram below shows the differences (in terms of branches created) between the two strategies.

Regardless of which strategy is used, at the point where the branch is merged to master a pull request (a request by a developer to merge their code changes into master) is raised. The changes in the pull request are then peer reviewed by at least one other developer with the outcome being one of:

* Approve: The changes are approved and the branch can be merged into master
* Add comments: Comments may be added by the reviewer which the original developer must responds to (or change their code accordingly)