Branching & Merging Strategies using GITHUB

KEEP CHALLENGING™

Run Better.
Run Different.

Cognizant

Agenda

- Source Code Management
- How to implement Source Code Management?
- Source Code Management Tools
- Why GIT?

Source Code Management

A source control management system (SCM) is software that provides coordination and services between members of a software development team. At the most basic level, it provides file management and version control so that team members don't write over each other's changes, and only the newest versions of files are identified for use in the workspace.

Why is Source Code Management important?

The main benefits claimed for Source code management are:

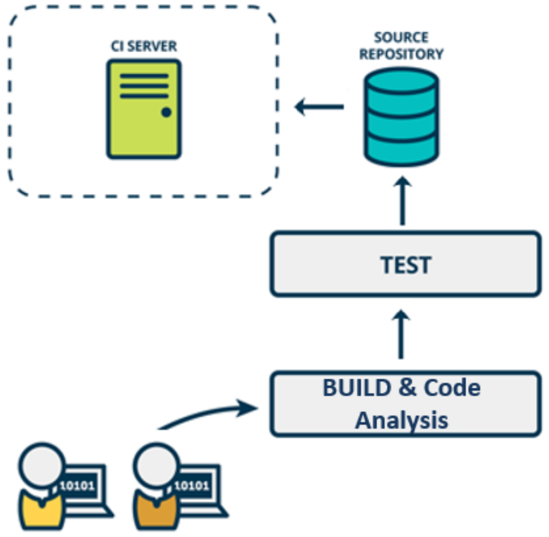
- **Code Modification History -** Version control provides a complete history of every commit made by every developer
- **Concurrent File Editing -** When working in a team there are times when multiple developers need to edit code in the same file. At minimum basic file locking is required to prevent developers from overwriting each other's modifications
- **Tagging -** One of the features of version control tools is the ability to create a snapshot of the codebase at any given moment. Such a snapshot is called a revision tag
- **Branching -** Branches allow you to maintain a separate copy of your project that is not simply frozen in time like a tag is. Branches can be updated individually without affecting other branches or the trunk
- **Merging -** Merging is the act of combining changes from divergent codebases. It typically occurs between two branches, or between a branch and the trunk

Problem statement

- Someone pushes code with errors and the Code breaks.
- Need to find who broke it?
- Everyone’s workflow halts and the search begins on everyone side to identify the issue.
- Intermittent commit has unpredictable result with others code.

How to implement Source Code Management?

- Developer develops the code
- Build locally and do any code analysis
- Runs all unit and functional tests
- Commits the code to SCM repository



Source Code Management Tools

There are a variety of free and open source control management tools that are configurable based on a project's needs.

- **SVN : Subversion** is the most popular open source Configuration Management tool used for version controlling. Being open source, it is freely available over the net. It provides many quality control and cost effective

- **GIT** is a distributed revision control system with an emphasis on speed, data integrity, and support for distributed, non-linear workflows

- **Rational Clearcase** is an enterprise-grade configuration management system that provides highly secure version control with work and build management support

- **Team Foundation Server** (commonly abbreviated to TFS) is a Microsoft product which provides source code management (either via Team Foundation Version Control

- **Mercurial** is a cross-platform, distributed revision control tool for software developers. It is mainly implemented using the Python programming language

Why GIT?

Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.
Git is easy to learn and has a tiny footprint with lightning fast performance. It outclasses SCM tools like Subversion, CVS, Perforce, and ClearCase with features like cheap local branching, convenient staging areas, and multiple workflows.

Features:

- **Frictionless Context Switching**. Create a branch to try out an idea, commit a few times, switch back to where you branched from, apply a patch, switch back to where you are experimenting, and merge it in.
- **Role-Based Codelines**. Have a branch that always contains only what goes to production, another that you merge work into for testing, and several smaller ones for day to day work.
- **Feature Based Workflow**. Create new branches for each new feature you're working on so you can seamlessly switch back and forth between them, then delete each branch when that feature gets merged into your main line.
- **Disposable Experimentation**. Create a branch to experiment in, realize it's not going to work, and just delete it - abandoning the work—with nobody else ever seeing it (even if you've pushed other branches in the meantime).

branching

- New commits are only added to the current branch
- Can implement experimental code without affecting others
- When the life of the branch is over, Merge it into the primary codebase, optionally the branch can be deleted.
- Branching is crucial on large teams, as it helps to work independently

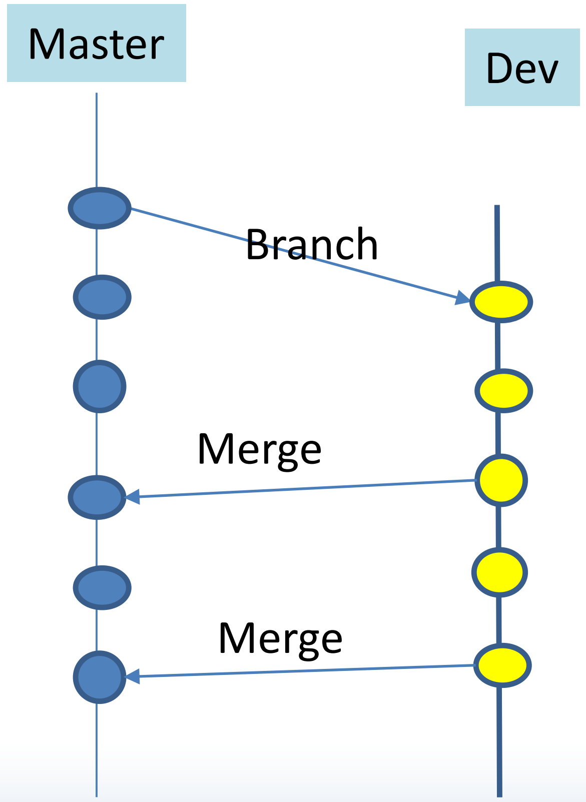
merging

- Once the changes are complete and stable in the current development branch
    - Merge the branch back to the stable sharing code branch.
- Typically it will be done as below:
    - Initiate a pull request
    - If your team performs code review, It will likely happen here
    - Choose how to merge the code
- Conflict resolution can be time consuming if to many changes on the same file.
    - Need to coordinate with your team and other team working on the file and also limit editing the same files

Branching strategies – Master and Development Branch

- The two primary branches – Master and Dev (Development)
- Master is the default branch in git used for the stable code base for releases
- Create Dev branch for development
  - Use for untested code
  - Primary working branch for the team
  - Contains the latest features
- When code validated and good for a release
  - Test Dev branch for stability
  - Merge commits in Dev into Master
  - Test and release Master
  - Continue to code on Dev and then promote to the release using the Master branch

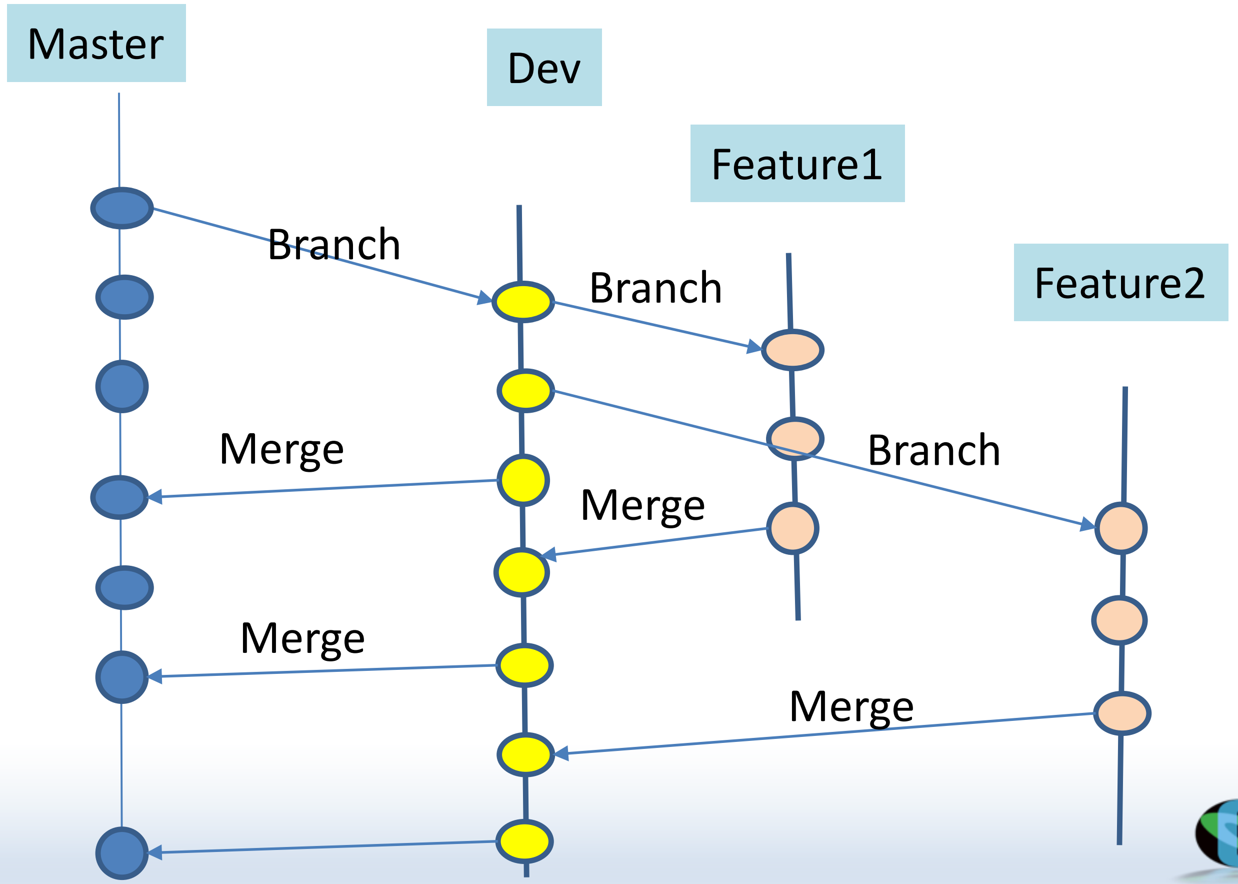
Branching strategies – Master and Development Branch



Branching Strategies – Feature Branch

- Branch off of Dev Branch
- Primary working branches for an individual(s) for a specific feature
- When new feature is finished in the feature branch
  - Merge into Dev branch
  - Code reviews (If applicable)
  - Hope it doesn’t break develop (it might)- we should to Continuous Integration or Nightly builds to find out
- Temporary branches can be removed if its successful or failure, in case of failure, Delete the branch without merging into develop
- Can be many feature branches being developed in parallel

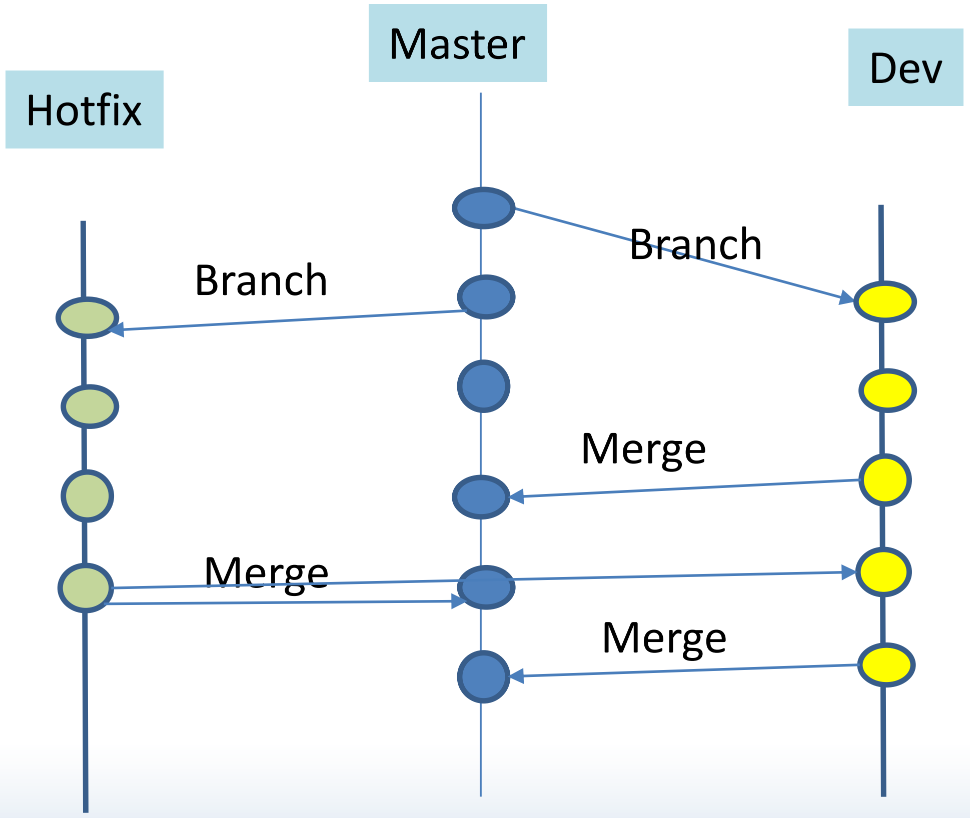
Branching Strategies – Feature Branch



Branching Strategies – Hotfix Branch

- Production release contains a bug, that needs urgent fix.
  - Create a hotfix branch from master
  - Fix the bug
  - merge Fix changes into:
    - production release branch Master
    - Dev Branch for a

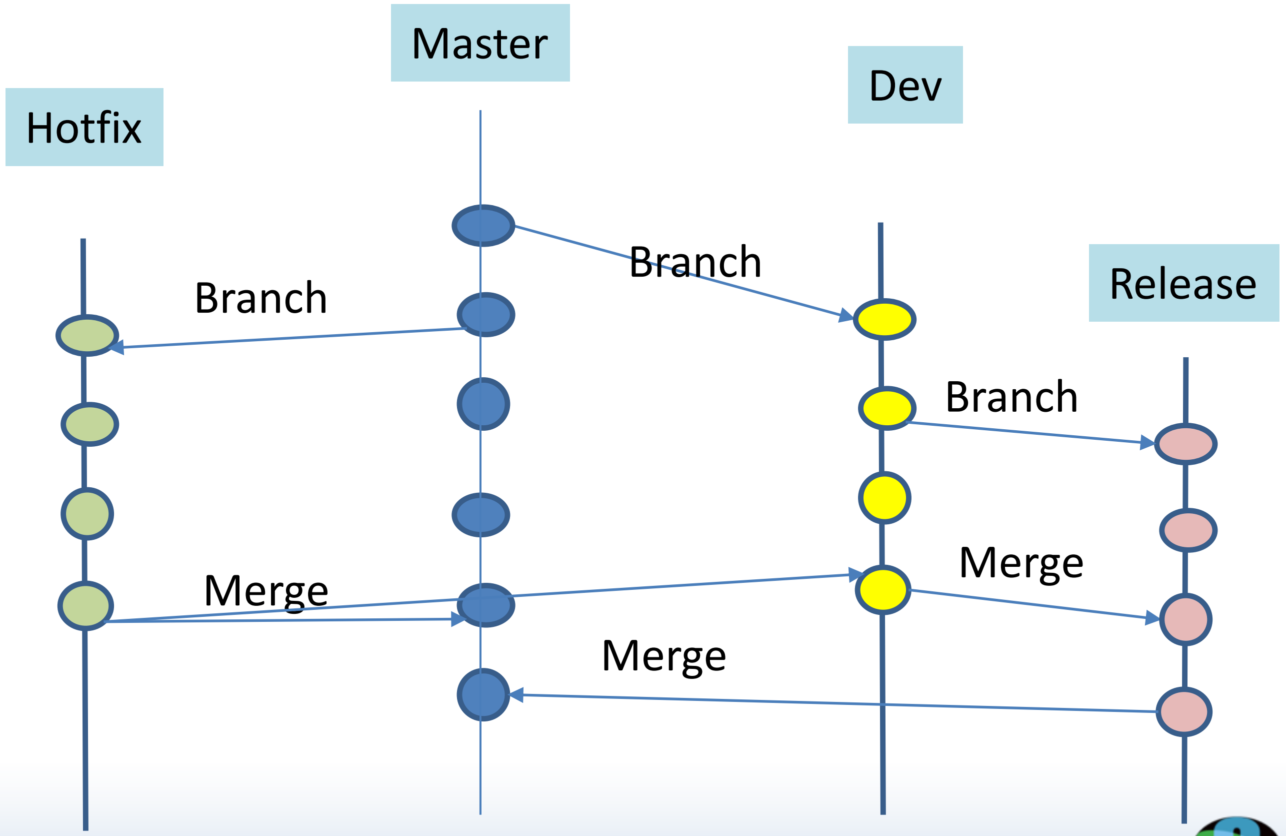
Branching Strategies – Hotfix Branch



Branching Strategies – Release Branch

- Branch off develop when approaching a release
- No features added
- Extensive testing and bug fixes
    - Merge all changes back to develop
- When confidently stable
    - Merge into master as a release

Branching Strategies – Release Branch



Demo

- GIT Installation Demo
- GIT Branching and Merging Demo

References

http://www.cio.com/article/2438652/developer/source-code-management-systems--trends--analysis-and-best-features.html

https://xebialabs.com/the-ultimate-devops-tool-chest/source-control-management/

https://git-scm.com/

Cognizant