GIT for Hardware Designers

Jeffery Scott, Sanjeev Singh

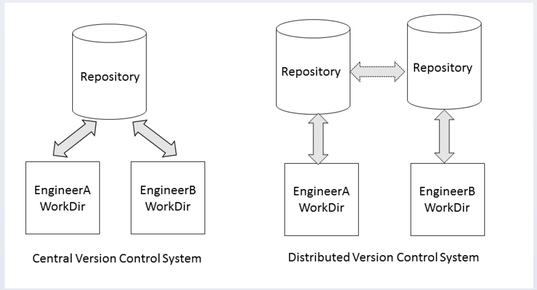
Juniper Networks

Modern Version Control System

A modern version control system is needed to support the dynamic workflow of AGILE hardware design.

Conventional central version control systems (CVCS) like “Subversion” use a single repository for the project. However they don’t scale well to this dynamic flow.

We propose using “Git” a distributed version control system (DVCS) to solve these issues.



Advantages

Git uses branches as the canonical solution to solve all the problems related to sharing and supporting multiple versions of the design.

Major features supported are

1. Lightweight Branching

Git branches are much more lightweight than other version control system. Hence designers are encouraged to create branches for each independent line of development and commit often to share code.

2. Better Merging.

Git can detect multiple merge scenarios and supports multiple strategies to deal with them. The various strategies supported are

- Fast Forward
  - If merging back into the original branch which has had no new commits, Git simply “fast forwards” the branch pointer.
- 3-Way Merge
  - Used when both the branches have commits that are not in the other branch. Git creates a merge commit using the 2 latest commit and their common ancestor.
- Rebase
  - Git allows the user to take a series of commits on a branch and replay them on top of another commit.

2. Powerful Commands For Release Workflows.

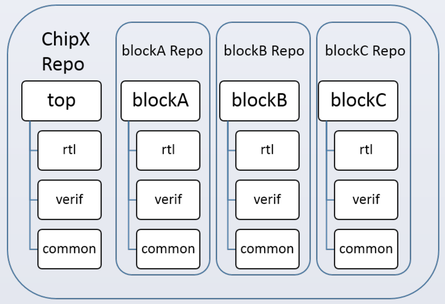
- Cherry Pick Commits
  - Pick Specific Commits From Other Branches then commit them to the local branch.
- Assigning Blame
  - Find out when the problem line was introduced in the repository.
- Find Bad Commits Using Bisect
  - Repeatedly divide the commit history between the “good” commit and the “bad” commit to find the commit that introduced the bug.

Recommended Setup

1. Block Centric Repositories.

Each block should be a self contained repository with minimal dependencies to other blocks. The chip repository is viewed as a composite repository of block level repositories.

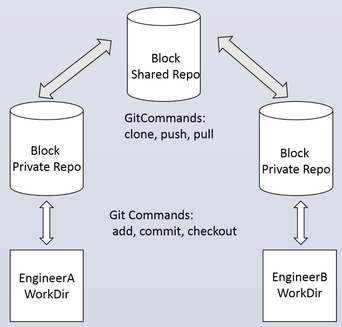
This is important since Git fundamentally looks at the whole repository and carries the entire information around. So Git will scale very badly if there is one whole repository for the chip.



2. ***Shared Central Repository***

Technically, Git does not attach any semantic value to any repository. However it is useful to create a shared repository in a central area that is the “blessed” repository.

Designers can continue developing on their local repositories but treat this central repository (commonly referred to as “origin”) as the repository for sharing code with other members of the team and for maintaining the production ready version of the code.



3. ***Naming Convention For Branches***

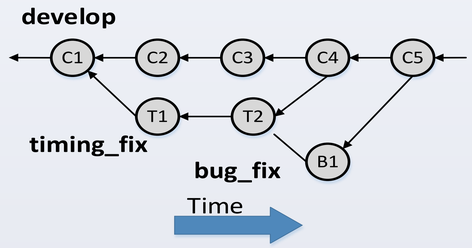
We recommend creating the following infinitely living branches on the central server.

- “origin/develop” that is the state of top of tree. All regressions are run on the tip of this branch and once passing, will be a candidate for making a release.
- “origin/master” as the release branch. Commits on this branch reflect the releases done by the projects. These should be tagged as per the release conventions.

For sharing code we recommend creating “Topic” branches that are temporary and eventually merged back.

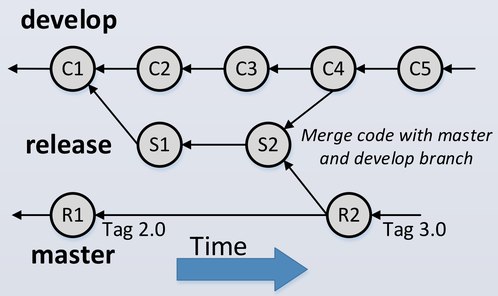
Typical Scenario

- Designer needs to support multiple versions of the design and work with various independent teams.
  - Provide a timing fix to the PD team.
  - Provide a bug fix to the DV team.
  - Continue working on new feature sets.



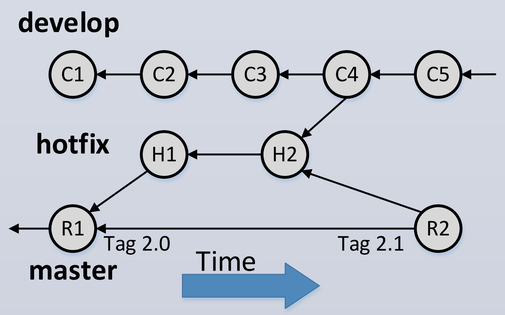
Designer needs to support the release process.

- Provide bug fixes to release process.
- Merge bug fixes to the next release.
- Continue working on new feature sets.



Designer needs to provide a hot fix to a release.

- Quick fix for released code.
- Merge fixes to the next release.
- Continue working on new feature sets.



Conclusions

Feature	Pros	Cons	Comment
Private Repositories	Provide isolated environments to experiment without cluttering up the central repository.	Require infrastructure support for backups and secure access to the central repository.	With cloud computing this is not an issue.
Robust Collaborative and Release flows	Branch with advanced merging and sharing features allow more flexible flows.	Require training and support for the new workflows.	Many commercial companies offering such training.
Block Centric Repositories	Promotes IP centric workflows.	Reorganizing a chip centric repository into multiple block level repositories is not a trivial task.	Help reuse in the long run.

References

[1] Git [Online]. Available: http://www.scm-git.com

[2] Getting Git Right: Available https://www.atlassian.com/git/

[3] Github: Available http://help.github.com

[4] A Successful Git branching model by Vincent Driessen . Available: http://nvie.com/posts/a-successful-git-branching-model/

[5] Git repository size. Available: http://stackoverflow.com/questions/984707/what-are-the-file-limits-in-git-number-and-size/984973#984973

Contact

Jeffery Scott (jscott@juniper.net)

Sanjeev Singh (sanjeevs@juniper.net)