Git/GitDesktop/GitHub Process Flow for CM Operations

# Introduction

## Purpose

The purpose of this paper is to describe the common steps to be used when employing the git-based tools, including Git, GitHub, and GitDesktop, as the baseline set of Configuration Management tools to coordinate the CSSA CAP development activities.

## Scope

## Definitions

* Branch – the generic term for a named sequence of related commits within the Directed Acyclic Graph (DAG) that git uses as a basic organizing principle within its repository
* “master” – the standard pre-defined name for the mainline/primary/master branch within the git repository. Other branches are created and used relative to the master branch.
* MergeBranch – a git branch used for the purpose of supporting integration and evaluation of multiple candidate baseline updates prior to committing the changes to the ”master” branch
* DeveloperBranch – a git branch defined for an individual developer’s daily operations
* IntegrationBranch – a git branch used for integration of commits merged from multiple developers, with planned handoff to test for further integration test.
* ReleaseCandidateBranch – a git branch used for a relatively mature set of commits to be considered as a potential “release version”
* GitTools – within the context of this paper, this refers to the combination of Git, GitDesktop, and GitHub tools operating together.

## References

* [1] Git Pocket Guide, Richard E. Silverman, O’Reilly
* [2] Introducing GitHub: A Non-Technical Guide, Peter Bell & Brent Beer, O’Reilly
* [3] GitDesktop Operations: Terminology, Icons, and Operational use

## General Operations

An overview of the basic Git/GitDesktop/GitHub operations is summarized in reference [3].

For simplicity, a project could have all developers, testers, etc operating directly on the master branch, but that mode of operational mode has both pros and cons:

* It allows for more rapid sharing of code between developers;
* It allows for more sharing of code that may not be sufficiently mature
* It does not take full advantage of the GitTools.

A branch can be identified and named as an IntegrationBranch and handed off to a tester to perform some regression test on the content.

GitTools provide the basic CM capabilities, including:

* Create commits (a record of changes to CM controlled files)
* View the commit history
* Create branches
* Merging the commits from one branch into another branch
* Synchronize the contents of a local repository with the remote GitHub repository

## Developer Operations

Summary: Developers will define and use their own branch and publish the branch to the remote GitHub repo to make changes availble for peer review.

* GitDesktop will provide the common services to create named branches
* Developers should have their own branch defined for their daily operations (refer to [3]):
  + Create commits
  + Review local changes prior to commits
  + Review commit history
  + Manage separate local branchs as deemed necessary or useful
  + Commits can be merged from other branches that have been published to GitHub, including master and other developer’s branches having proposed changes
  + Commits in the developer’s branch can be syncrhonized to the remote GitHub repo for review by others before they are merged into an integration branch and handed off to integration test.
* Upon a developer’s determination that updates are ready for integration test, the developers’ repository, with the DeveloperBranch will be synchronized with the remote GitHub repository.
* Developers will coordinate with management and peers to announce significant updates published on developer branch for peer review and potential integration.

## Peer Review Support

Summary: Developers, testers, and management will work together to publish software updates on named branches for a quick (time-limited) review/comment cycle. Management must provide direction regarding how much time is allowed for review before proceeding with the default option: go on to the next step.

* Developers will publish their developer branch to the shared GitHub repository for review and comment as a part of peer review and management review.
* Comments from reviewers, peers and/or management as time permits, will feed into the decision to move the proposed updates into integration test and task the testers to spend time on regression test.
* Notes from developers and reviewers will be documented, in addition to the integrated documentation support provide as a feature of using Git, and feed into the regression test plan.

## Integrator Operational Support

*Summary: A software developer will be designated to serve in the integrator role to combine software updates from potentially multiple developers (in general). Depending on personnel availability, the integration may involve merging the single-developer’s results into an IntegrationBranch.*

* A developer, with coordination with management, can serve as integrator, merging commits from multiple sources to a named IntegrationBranch.
* IntegrationBranches can also be sync’ed to the remote GitHub repo for peer review support after various commits have been integrated together

## Integration Test Operations

*Summary: Test will execute integration test and change-specific software test in response to the creation of a new IntegrationBranch and direction from the management team to start an integration test cycle on the specified baseline.*

* Testers will access the content of named IntegrationTest branch for regression test when requested by developers and management.
* The IntegrationTest branch must be pushed to the remote GitHub repository to make it available to the tester.
* Test will identify two categories of tests required:
  + A common regression test suite designed to exercise the software capabilities to the extent agreed with management for the regression test;
  + A collection of specialized test cases, as necessary to provide coverage for modified areas of the code (if not already covered by the collection of regression tests)

## Formal Test Operations

*Summary: Inputs from the entire team will feed into the management decision to begin a formal test cycle with a specified set of formal test procedures.*

* After reaching a stable operating state, with the planned functionality, and required fixes demonstrated, the ReleaseCandidate branch will be submitted to formal test.
* The formal test suite will be defined to provide coverage of all implemented functionality.
* The formal test suite will be reviewed and agreed across the team including developers, testers, and management.