# Appendix SA-15.2: Project GitFlow work flow

The CivicActions Development Team uses a specific form of GitFlow as described in this document: <https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow>

## Security requirements for Project

Project has defined a set of security requirements to ensure the security of the code produced by the CivicActions developers. Most of these requirements are defined in the [CivicActions Security Policy](https://github.com/CivicActions/security-policy) but they are repeated here for clarity:

* Maintain current versions of software on all systems (sandbox, development, staging and production). This includes:
  + Drupal
  + Operating System (GNU/Linux or Mac OS X)
  + Editors, debuggers, and other tools
* Always apply relevant CVE patches within 30 days of their release, usually within two weeks.
* Limit live data on development machines, in particular:
  + no real PII data (other than test PII created by developers)
  + no copyrighted content
  + no access restricted files
* Custom code must conform to Drupal coding and security standards.
* Before being integrated into the development branch, all Project code updates must:
  + have an associated JIRA ticket
  + undergo static code analysis with PHP\_CodeSniffer (exceptions for CSS, features, perhaps others)
  + undergo peer review
  + pass automated testing

## Source code branch management

Three main and constant branches are in use:

1. **develop** - The working branch. New code which has passed automated testing and peer review is merged into the *develop* branch.
2. **release** - When code is in a stable state, passing all tests and meeting requirements set by the current sprint, the code in *develop* moves to *release*. More detail about this special branch in the next section.
3. **production** - Contains production ‘stable’ code.

Two other types of branches may be used: *feature* and *hotfix*. These branches are working branches, but are not constant.

## Basic Git workflow

Every step of the workflow creates an accountable audit trail of all changes:

* All configuration is captured using the Drupal Configuration Management so that it can be reviewed and tracked.
* Changes must pass an automated quality and security review as documented below (via PHP\_CodeSniffer enhanced with coder).
* Changes are committed to a working branch.
* The working branch must pass:
  + a set of more than 500 custom automated tests
  + peer code review
* The working branch is merged into the *develop* branch.
* Code undergoes peer review spot checks by the ISSO and CivicActions Director of Engineering.
* Code is deployed to developer sandboxes and a “devtest” instance.
* At the end of the sprint in which the changes are completed, the code is deployed to a staging site with a copy of the production database and undergoes further testing by the Product Owner.
* The Product Owner accepts the changes.
* A tag is “cut” that indicates a unique set of code and configuration.
* The new tag is merged into the production branch and is deployed automatically.

## Code development workflow

The ability to push code to the main branch is limited to just four persons. All other developers disable their ability to push code to the main branch.

Developers make all additions and modifications to code in their local copies of the repository. Developers create distinct temporary branches for their work.

Developer modifications to local copies of the source code undergo review and testing (discussed later) before being merged with any of three branches used for development.

Each of these branches exists for specific type of source code modifications. These branches are:

* Feature
* Release
* Hot Fix

These branches can be independently tested and evaluated before merging with main branch.

Developers register their local branches as distinct remote branches to keep their code changes separate from the main branches until their changes can be reviewed.

## System image updates

*Amazon Machine Images* (AMI) are scanned regularly for vulnerabilities and updates. Once they have been updated they are stored as “hardened” images to be used for deployments. Before being deployed they are scanned again and checked for updates. If there are updates available, the updates are applied before deployment.

## Source code review and merge management

The ability to merge modified code into production is limited to four individuals. All changes made by developers are reviewed by these individuals before the modified code is accepted.

### Pending merge requests

Developers create merge requests to integrate their code into the appropriate branch.

After the merge request is made, another developer peer reviews the code changes, accepts the changes, and approves the merge request.

### Review of pending merge requests

Code reviews must be performed by one of the four developers with permission to commit code. The person reviewing the code must be different than the code author.

During the code review, Git displays a line-by-line comparison of the original version and the proposed changes.

### Approved and record of approved merge requests

Approved merge requests are automatically regression tested. If the regression tests fails, the merge request is rolled back.