ActiveNet GIT Workflow

The workflow for ActiveNet is similar to [GIT Workflow](http://nvie.com/posts/a-successful-git-branching-model/) but with a little change.

## Main Branches

The main repository will always hold two branches:

* master
* develop

master

The main branch should be considered origin/develop and will be the main branch where the source code of HEAD always reflects a state with the latest delivered development. As a developer, you will be branching and merging from develop.

Branch origin/master will be used to translate commits from SVN to GIT. This branch is maintained by the tool Subgit, no direct branching and merging are allowed.

Changes from SVN

Merge from master

master

develop

## Supporting Branches

Supporting branches are used to aid parallel development between team members, ease tracking of features, and to assist in quickly fixing live production problems. Unlike the main branches, these branches always have a limited lifetime, since they will be removed eventually.

The different types of branches we may use are:

* Feature branches
* Bug branches
* Release branches
* Hotfix branches

### Feature Branches

Feature branches are used when developing a new feature or enhancement. No matter when the feature branch will be finished, it will always be merged back into the develop branch.

* Must branch from: develop
* Must merge back into: develop
* Branch naming convention: feature-<JIRA number>

Using merge request to merge the branch back to develop

Feature-branch1

Develop

Create feature branch when starting feature

Feature-branch2

### Bug Branches

Bug branches differ from feature branches only semantically. Bug branches will be created when there is a bug found. Bug branches are used to explicitly track the difference between bug development and feature development.

* Must branch from: develop or release-\*
* Must merge back into: develop and ongoing release branches
* Branch naming convention: bug-<JIRA number>

Create bug branch when fixing bugs

Merge bug branch back to develop and releases

Develop

Bug-branch1

Release-branch1

### Release Branches

Release branches support preparation of a new production release. They allow for last minute dotting of i’s and crossing t’s. Furthermore, they allow for minor bug fixes and preparing meta-data for a release (version number, build dates, etc.). By doing all of this work on a release branch, the develop branch is cleared to receive features for the next big release.

* Must branch from: develop
* Must merge back into: develop
* Branch naming convention: release-<release number>

Develop

Release-branch1

Merge the release branch back to develop after releasing to product; keep the branch for future hotfixes.

Create release branch when all the features planned in this release are ready

### Hotfix Branches

Hotfix branches are very much like release branches in that they are also meant to prepare for a new production release, albeit unplanned. They arise from the necessity to act immediately upon an undesired state of a live production version. When a critical bug in a production version must be resolved immediately, a hotfix branch may be branched off from the release branch that marks the production version. The essence is that work of team members (on the develop branch) can continue, while another person is preparing a quick production fix.

* Must branch from: release-\*
* Must merge back into: develop and ongoing release branches
* Branch naming convention: hotfix-<hotfix number>

develop

Create hotfix branch when fixing bugs after releasing to product

Merge the hotfix branch back to develop and release branch after release to product

hotfix-branch1

release-branch1

## Merging

### Merge Requests

Merge requests are created in a GIT management application and ask an assigned person to merge two branches. The merge requests serve as a code review tool and no separate tools such as Crucible should be needed. If the review reveals shortcomings anyone can commit and push a fix. Commonly the person to do this is the creator of the merge request.

When you feel comfortable with it to be merged you assign it to the person that knows most about the codebase you are changing and mention any other people you would like feedback from. There is room for more feedback and after the assigned person feels comfortable with the result the branch is merged. If the assigned person does not feel comfortable they can close the merge request without merging.

The assigned person for a merge request must be sure testing code is included as well as the production code.

## JIRA Integration

### Referencing JIRA Issues

If you reference a JIRA issue, e.g., ANE-53701, in a commit comment, a link which points back to JIRA is created.

The same works for comments in merge requests as well.

### Code Review Workflow

The current code review workflow in JIRA will be changed a little bit. As we are using merge requests instead of Crucible, please set the Crucible URL in JIRA ticket as the link to the merge request; in the same time please set the code reviewer to the one who is assigned to take care of your merge request.