Branch Management

The COmanage Project has adopted a modified version of git-flow as its branch management strategy. In short, the differences are

1. One hotfix branch exists per minor release. That is, `hotfix-1.0.x` is used for (eg) `1.0.1`*and*`1.0.2`. While these will typically branch from and merge into `main`, they may also branch from any release tag, and may not merge into `main` if `main` has already moved on to a new minor (or major) release. Both `main` and `hotfix-*` should be deployable at all times.
2. Release branches are not currently used. All features for release merge into `develop`, which eventually merges into `main` to create a release tag.
3. Feature branches (topic branches) are optional and should be named for the corresponding JIRA issue (eg: `feature-co500`). In general, feature branches are used when merging directly into `develop` is undesirable, perhaps because the feature is experimental. Features may also be used when a priority enhancement is made for a specific deployment, and the enhancement is required before the next scheduled minor release.
    a. Feature branches may also be used to target develop for a future (ie: not the next) feature release. In this case, the feature branch is named for the target release (eg: `feature-3.1`) rather than a JIRA issue.

Do *not* commit the same change to multiple branches. Pick the "earliest" relevant branch and commit there. For example, if you commit to `hotfix-3.0.x`, do not also commit to `develop`. Your commit will flow to develop at the next merge. This makes it easier to track where a change came from.

Under limited circumstances, it may be necessary to cherry pick a commit or otherwise violate this rule. Please discuss before doing so.

In general, do not merge `hotfix-*`, `develop`, or `main`. That will happen at release time. You can merge a temporary `feature-*` branch into `hotfix-*` or `develop`.

Summary

Branch	Description	Branches From	Merges To
main	Current release or release candidate	-	-
develop	New features scheduled for next minor release	-	`main`, `hotfix-*` (if appropriate)
hotfix-*	Bug fixes and minor changes scheduled for next bugfix release	`main`	`main`, `develop`
feature-*	Experimental or prioritized features	`develop` or (rarely) `hotfix-*`	`develop`, `hotfix-*` (if appropriate)
container-maintenance	Maintenance updates for container images, e.g. updated PHP version	main	main, develop, hotfix-*

Managing develop and Pull Requests

1. Pull Requests should be assigned a Reviewer, contextually determined. Absent any other decision, the Reviewer is the Component Lead.
2. The Reviewer should perform "appropriate" testing, such as pulling the commit to a local branch for testing, or applying a patch locally. The Reviewer may hand off for additional testing as needed.
3. The Reviewer can perform the merge, or hand the PR to the Component Lead to merge.
4. While develop is not guaranteed to be stable, it also shouldn't be left broken for extended periods of time. (Of course, in most cases "broken" is subjective, since only a small class of bugs will prevent the application from running at all.) Whoever discovers an issue owns it until handed off to someone else (eg: to fix, or to test the fix, or to merge the fix).
5. In general, these guidelines are intentionally somewhat vague to allow for professional discretion.

References

- http://nvie.com/posts/a-successful-git-branching-model
- http://scottchacon.com/2011/08/31/github-flow.html