CommitPolicies

This document is intended to capture some of the policies we would like all committers to follow when it comes to checking in changes into our GIT repository.

Commit Log Format

Each commit to the repository must have a non-empty log message. The message format should start with a one-line short summary, with a capitalized first letter of the first word. The short summary need not be a complete sentence with proper punctuation and grammar since it is best to keep it to 50 characters or less.

After the short summary line, you may continue the message with one or more paragraphs separated by an empty line, each line about 72 characters or less. The empty line after the short summary is necessary if you want to add more text since source control tools like git may send the log message as an email and use the first line up to an empty line as the email subject.

When submitting a commit with significant changes authored by another contributor, be sure to attribute such changes to the author in your commit comment.

Example Commit Log Message

```
Short, one line summary (50 chars or less), not a paragraph

More detailed explanatory text as necessary, patch attributions, and
what platforms the commit has been tested on. Wrap it to about 72
characters or so. In some contexts, the first line is treated as the
subject of an email and the rest of the text as the body. The blank
line separating the summary from the body is critical (unless you omit
the body entirely); tools like rebase can get confused if you run the
two together.

This is an example of a second paragraph. The 'Author:' line is required
if the commit represents code submitted by somebody who does not have
commit permissions. The person doing the commit is implicitly the
reviewer of the submitted code. The email address is optional but it
is nice to have so the original author can be uniquely identified. It
is OK for the committer to modify or cleanup the code and if done so,
should be noted in the commit log.
```

The master branch

Changes to the master branch are governed by a Review-Then-Commit policy that has been modified in the following way: Instead of a Consensus Approval, commits require at least one +1 Vote from another committer.

The master branch is expected to be stable, and changes should be tested and reviewed.

Release Branches

1. We only schedule one main release / year. Each main release branch is 2 years LTS (Long-Term Support).

2. A release branch is cut off master once a year. For example, 7.x.

3. The community can agree to make a minor release, e.g. v7.2.0 from a main release branch at any given time during the 7.x 2-year life cycle. Such a release is still branched off the corresponding main release branch. (Minor releases should not be branched from master.) Any such minor release is also LTS, that is, its support term is the same as that of the main release branch upon which it is based.

4. The Release Manager and community should agree on which commits from master should go into any minor release. This would be for example a new feature, or a new plugin etc. Critical crashes / security fixes is dealt with as normal with a patch release.

5. Critical fixes (e.g. security fixes) on an LTS branch must be applied to every LTS minor release on that branch. As an example, this could mean making a v7.1.1 release as well as a v7.2.1 release.