DevelopmentMode

The Different Development Modes

Commit-Then-Review

C-T-R mode (CTR) is short for "Commit-then-Review". This is the standard development mode that we operate in. It means that committers may commit patches to the development tree without the code/rules/etc being reviewed by other committers first. Typically SVN trunk is in C-T-R mode until a new release is approaching. We then either switch trunk to R-T-C mode or use a branch that is in R-T-C mode for the final development up to release, to stabilise the codebase.

When using a branch for development (e.g., when trunk was being used for 4.0, while development for 3.4.x was done in a 3.4 branch) any fix or change for an issue that applies to both trunk and branch should be committed first to trunk, then to the branch, to ensure that the change is not inadvertently left out of trunk. When trunk is in C-T-R mode and the branch is in R-T-C mode this would require a vote after the commit is applied to trunk but before the commit is applied to the branch. Note that this workflow is likely to change if the project moves from using svn to git, which more commonly makes use of feature branches for testing followed by a pull release to apply to a master (trunk) branch rather than committing to master first.

Review-Then-Commit

R-T-C mode (RTC) is short for "Review-then-Commit".

Non-trivial patches (see below for a definition of what's considered a 'trivial' change) must be reviewed by committers, and need consensus approval before being committed into the development tree. This is done by opening a Bugzilla ticket, setting the Target Milestone to the correct release version of the tree, attaching the suggested patch to the ticket via the web interface, and putting the ticket in "review" status (indicated by adding `[review]` as a prefix to the ticket summary).

The patch is then voted upon, and if gets a consensus approval and is not vetoed, can be applied to the tree. Votes should generally be permitted to run for at least 24 hours to provide an opportunity for all concerned persons to participate regardless of their geographic locations. "Consensus approval" refers to a vote which has completed with at least three binding +1 votes and no -1 vetos.

The author of a patch is allowed to vote as long as they're a committer. Typically, if a committer uploaded the patch, it's assumed they're implicitly voting +1 on their own patch, unless otherwise specified.

Trivial Patches: When R-T-C is Optional

Trivial patches include:

- documentation
- finishing off pre-existing T_ tests
- changes to rules in the "rules" or "rulesrc" trees
- non-controversial non-semantic style changes (fixing indentation, adding comments, but not actual code)
- very simple, non-controversial, and absolutely safe bug fixes (i.e.: removing repetitive my() enclosing sections)

These can be applied without a vote.

Time Delays for Code Modifications

Votes should generally be permitted to run for at least 72 hours to provide an opportunity for all concerned persons to participate regardless of their geographic locations.

(Since it does say "generally", it seems reasonable exceptions to the 72 hour rule are allowed if you specify such in your vote, but let's always allow at least 24 hours or at least 48 hours if the weekend is involved. Remember, though, that if someone later vetos with a technical explanation, then the code gets pulled.)

How To Review A Patch

Please don't vote +1 unless you actually did *something* to check the patch. That means some form of testing or code review. You do not necessarily need to apply the patch to your local copy of SA, but do take a look at it before voting +1.

It's not your responsibility as the reviewer to run 'make test'. It is assumed that a review patch already passes this, so don't worry about it.

If you don't understand the workings of the code you're voting on, don't worry about it too much. Do your best, and if all else fails, vote on the structural aspects of the patch, and its code quality. It's not expected that every committer knows how *all* of SpamAssassin works – but it is important that R-T-C has enough people voting!

Running 'make test'

'make test' should be run before committing anything, unless the change doesn't modify the code in any way (such as a documentation change). If you check in something that breaks 'make test', you have Done A Bad Thing.

If you send out a patch for C-T-R, and your patch manages to break 'make test' on its own (ie. not through interaction with other current review patches), that is Also Bad. However, it's not the fault of the reviewers – it's yours 😉

Reverting Code, and Vetoing

To veto code, you must issue an explicit -1 veto in a bug, or in a reply to the check-in on the dev mailing list. If the veto is for a security-related fix, you may veto on a private forum. In addition, the veto must be accompanied with a technical justification.

Vetos should be avoided for purely procedural reasons. If you are vetoing code, it is considered polite to allow the author an opportunity to respond or revert the code themselves, but it is not quite as imperative to wait if the change is very broken and fixing it would require significantly more effort than reverting it.