Contribution Workflow

Overview

In comparison to the normal 'ticket' flow contributors need to be able to walk their own *new feature* tickets through the process as well as contributed bug fixes. It may look bad to start, but once you get a basic understanding of Git, Github, and Lighthouse it is super easy.

Note:

This document is not *a 'how-to' for setting your work environment. There are numerous step-by-step guides available for that already.*

Ticket state flow:

new → **open** → **needs_ack** → **acked** → **proposed** → **candidate** → **resolved**

The first two don't have as much meaning for new-feature tickets as for issues.

- [*new*] – This state is useful if you want some input on your idea before you even start coding.
- [*open*] – If you are working on your feature but don't have a full solution to share this is a good state to use.
- [*needs_ack*] – You have some code ready to be looked at by others and it is posted publicly.
- [*acked*] – Someone has looked at your idea and code and acknowledges it should be included in the proposed updates branch.
- [*proposed*] – The update can be replaced by a better update, altered, backed out, etc… It is not expected to be stable at this point, but is provided so it can be tested and enhanced by the community.
- [*candidate*] – The update seems like a good solution, is stable, and no-one seems to have reasonable objections to including it in the project.
- [*resolved*] – The update is now included in the master branch and will be a part of the next tagged release.
- [*hold*] & [*invalid*] – have their general meanings and will be commented when the state is changed.

Branch flow and meaning

topic → pu → next → master

- [*topic*] – A branch off of the master branch that has your new-feature on it that you want reviewed.

Comment [JL1]: I would not mention that it "looks bad". IMO, that put the reader in the wrong state of mind. Also, "super easy" sounds high school-ish. I'd go for something more formal in the lines of: "Don't be daunted by the process. Once you get a basic understanding [...] the workflow will make sense".

Formatted: Font: Not Bold

Formatted: Font: Italic

Comment [JL2]: Links?

Formatted: Font: Italic

Formatted: Font: Italic

Comment [JL3]: You shift from addressing the reader ("You are working [...]") to a neutral stating of facts ("The update can be replaced [...]"). I'd stick to one of the two. My personal preference goes to the neutral form.
This is true of the document as a whole, btw.

Comment [JL4]: I'd name them explicitly, and put this whole sentence at the end of the section, after [*hold*] & [*invalid*]

Comment [JL5]: I'd just say "Use this state to let others know that you are currently working on this ticket"

Comment [JL6]: Looking at this workflow again and again, I think "acked" and proposed should be one and the same thing.

Comment [JL7]: Hold and invalid are not thoroughly explained in this document. One thing I'd like mentioned is that it should be mandatory to explain why a ticket gets in this state, maybe with a substate (will not fixed, worked as designed, not a bug, etc...) an at least a few lines of info. Don keeps closing tickets on SF without any explanation, and this leads to frustration from the users.

Comment [JL8]: Would here be the place to give the format of such a topic branch? i.e. jl/LH-22/wiggle_stuff_around ?

- [*pu*] – The proposed update has been acknowledged as reasonable and needs testing and community feedback.
- [*next*] – The update has tested OK and should be included in an upcoming release, but still needs more community feedback and usage.
- [*master*] – The update will be a part of the next release and the issue is considered resolved.

Note:

Sometimes these steps can go very quickly and portions of the flow will be skipped because the update is obvious and trivial. This flow is not written in stone and is not meant to be followed pedantically. Rather it serves as an assurance that we truly are thinking about quality.

Comment [JL9]: YAY! :-D

Flow Details.

The 'idea' stage.

Related ticket states: [*new*] [*open*]

If you want to help but need some ideas visit:

The IdeaTorrent.
Our Freenode #Notepad++ channel.
The plugin forums.
The open forums.

Comment [JL10]: The format of this section does not match the one from subsequent ones. It lacks an italicized subtitle (e.g. *Getting Acknowledgment* below) and a short description of what is expected of the user at this point.
Also, while the ticket creation is detailed, the ticket opening isn't.

Comment [JL11]: Shouldn't LH serves as an IdeaTorrent for N++CR?

Here are some things to remember when creating your ticket:

- You will need to be signed in to your Lighthouse account to create a ticket.
- Please use an informative subject.
  - It should start with '[New Feature] featurename: description' or '[Enhancement] enhanced area – descritpion' or some such prefix.
  - Modified proposals then use '[New Feature v2] …..' v3 and so on. This makes it very simple for everyone ~~to~~ interested to track what is going on.
- Assign the ticket to yourself.
- Don't bother assigning the *Milestone* it will be figured out later.
- Add some descriptive tags.

Comment [JL12]: IMO, a link to "how to create a LH account" would be nice here.

Comment [JL13]: I'd appreciate if we'd have the actual list of keywords that should be used here, including for bugs.

Comment [JL14]: A simple "how to modify a proposal" paragraph would be nice.

Comment [JL15]: I don't think that's always true.
1- For bugs, they should just be assigned to npp-community. No?
2-If we're to use LH as an IdeaTorrent replacement, then a feature can be proposed but not own by the submitter.

Comment [JL16]: I'd like to have clear set of tags for bugs, crashes, features, etc, so we can create ticket bins to easily track those.

All new features and enhancements need a ticket.

The 'acknowledgment' stage.

Releated ticket states: [*needs_ack*] [*acked*]

Getting Acknowledgment

Git, Github and Lighthouse work together, so it is recommended to use them. How? After an initial setup you puseh a local branch to your Github forked account, the commit message is sent to Lighthouse. Lighthouse then looks at the commit and uses a small block of embedded text to populate a post to the ticket referenced. Your full commit message becomes your ticket post and the state of your ticket is updated. The patch is posted to the project timeline, an update is emailed to people watching the ticket, and also included in the ticket is a link to your commit on Github where comments can be made and your code reviewed line by line.

Giving Acknowledgment

When giving an acknowledgment an update is checked to ensure that it:

- doesn't break any existing tests.
- is well written.
- is limited to the issue being fixed.
- actually fixes the issue.

Acknowledgment of someone's code is just a way of saying "It looks good to me, and sounds reasonable", it is not meant to be an intensive scrutiny.

The 'proposal' stage

Related states: [*proposed*]

Proposed Updates

Ticket with a state of 'acked' are merged into the main repositories *pu* branch and graduate to `proposed` where they get scrutinized in detail. Other developers in the community will make suggestions or offer their own changes and updates.

While at this stage the update is checked to ensure that it:

- includes tests to ensure the fix doesn't break again in the future.
- builds on each target platform in each of the configurations.
- has any needed documentation.

This step can go round and round sometimes. While getting an update into the project is important it is also important that it be maintainable in the future. Sometimes the person introduced a poor solution and does not follow through but the idea is a good one and it may be awhile before someone picks it up again.

Comment [JL17]: You cover the "how to" with Github (which strangely does not seem mandatory, btw), but not the manual way (that would be needed if one does not use Github). Also, the condition specifying when to switch to "needs_ack" are not covered (as they are *for Giving Acknowledgement,* below)

Comment [JL18]: Too informal IMO. I'd use "Here's how: " instead.

Comment [JL19]: Link? Do we have an "internal" document for this?

Comment [JL20]: Repetition. "to Lighthouse, which then looks [...]" or "to lighthouse. It then looks [...]"

Comment [JL21]: An update? This is not clear to me. Shouldn't it be "the code in the branch" or something more precise?

Comment [JL22]: While I totally agree, this is still very vague as we don't have any style guideline. Maybe we should come up with certain rules in the future to help the reviewer (and the reviewed) come to consensus more rapidly.

Comment [JL23]: This does not apply solely to bug fixes (as the sentence seems to imply), but to all submitted code. It should be clearer on that point.

Comment [JL24]: "Required" instead of needed?

Comment [JL25]: "Sometimes, a coder can introduce a poor solution". "*The*" person isn't introduced before in the section.

The ‘integration’ stage.

Related states: [*candidate*] [*resolved*]

Candidacy.

The tickets with a state of `proposed` graduate to `candidate` when the community agrees it is ready.

When the proposed update looks to be ready, a milestone will be set for inclusion into a release and it will be merged into the *next* branch of our repository.

The main purpose of this stage is to get wider community coverage of the proposed update before exposing all users to it. Possibly revealing hidden problems, or getting ideas from someone else on how to make it even better before releasing it. `Candidate` tickets will appear crossed off in the LighthouseApp ticket monitor but not counted as resolved since it can still be ‘undone’ if a problem is found.

When the proposed update reaches this stage you can be pretty assured it will work and not cause problems. Unless someone finds something wrong there is nothing left for you to do.

Comment [JL26]: Again here, the “you”is brought out of nowhere.

Resolution.

The tickets with a state of `candidate` stay on the *next* branch until it is time to merge with the *master* branch to meet their milestone. They are then `resolved`.

By the time an update becomes a candidate the issues with it have been found and resolved, so there isn’t really much left to between that point and when it becomes resolved except to wait for the milestone and hope other developers and users are using it. Developers are encouraged to use the next branch for daily use to help make sure candidates are worthwhile.

When the milestone is reached the update will be merged into the *master* branch and the ticket will be marked as resolved. The only thing left for anyone to do is use it and be happy.