Pull Request Tips

When making pull requests, there are a number of things you may forget to do and hence lead into problems when trying to merge, and others. Therefore, ensure that you do the following:

- **Got ticket?** Ensure the pull request you are about to create, is for an existing JIRA ticket. If the ticket does not exist, create it. Since we use JIRA tickets to audit and inform release notes, we shouldn't be handling pull requests in GitHub without any documentation in JIRA (except for very trivial changes like fixing typographical errors). When you create a ticket, it starts in a "Needs Assessment", or "In Backlog/Ready for Analysis" state. Do not work on it until when some one assesses it and eventually makes it "Ready for Work" or "Waiting for Dev". This will help you avoid wasting time working on what is not necessary. Also the ticket assessor may even give you some pointers regarding how to get started with this particular ticket, which files to change, and more. We call this ticket curation. When ready to start working on the ticket, click the "Claim Issue" or "Claim for Development" button to assign it to yourself, and hence prevent others from duplicating your efforts by working on the same ticket.

- **Branch name == ticket name.** In most cases, when working on a bug fix or new feature, you should create a new branch using the name of the ticket. For example, if you are working on ticket FOO-123, then start by creating a branch off of master named FOO-123 using a command like this from an updated master branch: `git checkout -b FOO-123`. Every time you finish a branch and want to create another, you need to first switch to the master branch, and then create the new branch from there. If you do not do so, your pull request for the new branch will get mixed up with commits from the other branch.

- **Reference the ticket.** The summary/title of the pull request should contain the ticket's JIRA id e.g TRUNK:4196: Fix some issue and you should include the link to the ticket in the description field. If you are working on a sub task of another parent ticket or story, the ticket id to use in the summary is that of the sub task and not that of the parent. The pull request title should also be a good summary of the work done. If you are not sure of what to put here, just use the ticket title.

- **Pull against master.** Always create pull requests on OpenMRS's master branch, unless the ticket description or comments specifically say otherwise. The reviewer of the pull request will always take care of back porting your changes to other branches, whenever necessary. In theory below the title of your pull request in github, there should be a line saying something like 'XYZ wants to merge 1 commit into openmrs:master from XYZ:master'

- **Update first.** Pull the latest changes from upstream and resolve conflicts if any. Use this command: `git pull --rebase upstream master`

- **Tests passing?** Run "mvn clean package" before committing. Running this maven command not only ensures that all tests are passing, but also formats all your changes according to our conventions.

- **Follow our coding conventions.** Make sure you have read and followed our Coding Conventions. Please read them carefully and check the diff of your commit. Make sure your IDE is properly configured as explained. This helps reduce the time we need to review your code.

- **Avoid a noisy pull request.** When committing, you may at times see files to be committed, which you did not intentionally change. Some could simply have formatting changes as a result of running mvn clean install. Please ensure that you do not include any such files in your commit.

- **Squash!** Best practice is to commit at the level of an idea (feature/fix). While you may have made 10+ commits while you worked, before you submit a pull request, make yourself look even smarter as if you knew how to do it all at once by "squashing" your changes into one that addresses the ticket. For example `git rebase --onto HEAD~3`, where the "3" in HEAD~3 is refers to the number of prior commits you want to squash and then `git commit --rebase Large features may be done across a handful of commits, but most bug fixes and small features can be squashed into a single commit, making the commit history easier to follow. Since GitHub added a feature to squash on merge, squashing is less critical. In some ways, continuing to squash commits after a pull request and its review has been started can end up making it harder to follow the code review comments. So, consider squashing when creating a pull request, don't worry about squashing subsequent commits performed in response to the reviewer(s) comments. Ultimately, the person merging the pull request can perform a "squash on merge" via GitHub to make the final result appear a single, clean commit.

- **Help us help you.** Use a meaningful message for the pull request. This message should tell us what the pull request is all about, and include the ticket number. If you are not sure of what this message should be, at least use the ticket summary, as it appears in JIRA.

- **JIRA workflow.** Click the "Request Code Review" or "Ready for Testing" button in JIRA for the ticket to notify us that it is ready for review. Remember include the pull request url on the ticket, as a comment. Please do not put the ticket state in "CODE REVIEW (POST-COMMIT). This will be done after merging your changes by whoever will have reviewed them. If it is a reference application project, you will instead click the "Ready for Testing" button.

- **Use JIRA.** If your changes have anything to do with the user interface, do not attach the screenshot on the pull request. Instead attach it on the JIRA ticket. When adding the attachment, include a comment to state the name of the attached file, just in case the ticket gets more than one. If these are changes for an Open Web App, remember to attach the compiled app, on the ticket.

- **Avoid multiple pull requests for the same ticket.** In most cases you don't need to create more than one pull request for the same ticket unless there is something wrong you did while creating it that you can't change. But if you are just making code changes to the original pull request you submitted, all you have to do is commit to the same branch linked to the pull request, once you push the code to your remote branch, github will automatically pick up the new changes and update the pull request. The only time you don't have to continue with the same pull request is when it has been merged or closed.
• **Replacing a pull request.** If you decided to create a new pull request for the same ticket, please close the old pull request and add a comment for why you are closing it and point to the new pull request in your comment.

• **Abandoning a ticket.** If you decide not to do any more work on a ticket, either because of lack of time, or any other reason, remember to unassign yourself from it such that it becomes available to someone else to take it over from where you left. If during the course of working on it, you discovered something that may be useful to whoever takes it up, share that as comments on the ticket. If you had attempted to open any pull request, remember to also close it with a comment stating why.

• **Travis CI build failures.** After creating a pull request, check to ensure that Travis CI builds successfully. If you end up with something like this: "continuous-integration/travis-ci/pr — The Travis CI build failed" then click the "Details" link next to it for details regarding the failure. This mostly happens if your forgot to run "mvn clean install" to ensure that all tests pass before committing.

Lastly, please make sure that you have the Using Git page, especially the "Submit the code" section.

### Resources

• **Anatomy of a perfect pull request**