Contributing Code

Fork vs. patch

The favored way of contributing code to darktable is through github forks. We will continue to deal with patches in our bug tracker or on the mailing list, but github fork are much easier for us to deal with. They allow per-line comment, they are easy to merge, we can more easily look at progress.

And more importantly, when you fork on github and have a public branch, we see it immediately through the github repo network (see https://github.com/darktable-org/darktable/network). This means that we could comment on a feature before you invest too much time in something we don't like. But please keep in mind: it's always better to discuss your projects on the mailing list or IRC beforehand...

So yes, fork a lot, it's a great thing, even for trivial patches.

Naming branches

If you work on a large feature (any feature that you expect to generate multiple commits) please give it a proper name related to that feature (i.e avoid naming the branch after your own name). It will make things easier once you are done and start working on something else, and it's easier for us when browsing through branches to figure out what you are working on. If you are working on something related to an issue in our tracker, using a feature request or bug issue number as branch name is a good idea, too. The other way round: it might be worth to add a ticket to the tracker and set the status to "In progress" so people can easily see you are currently working on this.

If you just want to do a quick fix it's ok to work on (your fork's version of) master. If you have multiple trivial patches, stack them on master, that's fine. Just make sure that you do a pull request often and re-merge the main master regularly. You can also create a branch named after a bug of course, but for trivial fixes, working on master is less of a problem for us.

Last, if you have finished working on a feature, have create a pull request and start working on something else, please recreate a new branch based on the main master branch, and do not commit new things on top of your first feature branch (that is: assuming both features are independant of course). This will allow us to merge your latest feature independently if there is a problem with the first one that has to be fixed before merging back. It will make things easier for you if we are not happy with your feature and you have to add commits after the pull requests, and make things easier overall.

Creating a Pull Request

Don't create a pull request until you are ready to merge your work. A pull request is a call for us to go through your work and then merge it back in the main darktable repository. So you should only create your pull request once you think your feature is ready.

This also means that you can post a "request for testing/review" on the mailing list for any work in progress, you don't need a pull request for that...

We might need some time to deal with your pull requests so don't be suprised if there is no news for a week or two... We do our best but we are all volunteers, too. On a related note, the bigger the change, the more time it will need for review. So don't expect new modules to be handled as fast as one-line changes, especially if the changes contain new image operators changing our processing pipeline.

Pull requests are attached to branches, not commits. This means that if you push some commits on a branch that has a pull request pending, the pull request will move with the branch head. This allows you to continue to work on a branch to fix whatever issue we have with it, but it also means that once you've done a pull request you should not add unrelated work on top of it. Use a new branch for that.

Since we need time to review a pull request, please try to keep it up to date by regularly merging with our master branch (a good rule of thumb is to re-merge master at least whenever auto-merge won't work, which can be seen at the bottom of the pull request). This will save us some time and remind us that you are waiting on us and that we should do something. Having your contribution based on the latest master branch prevents us from having to solve merge conflicts once your pull request is reviewed.

We will usually not pull unless there is a pull request. I.e. don't assume your single change will be noticed. It usually will be but unless you do a PR we will assume you have a good reason for not wanting it merged.

We might pull if we think your idea is very important and we are ready to work on it to complete it (or you are unresponsive to comment and we are ready to take matter in our own hand) More commonly we might pull any quick fixes, especially if we are about

to release, but these are special cases. Our rule of thumb is that your branch is yours and we will not pull until we are invited to.

Assigning and Merging Pull Requests

(this paragraph is more for devs than for external contributors)

Pull requests can be assigned to a dev. Assigning it means that the PR is not for grab anymore, it is an "exclusive lock" on that request. In other word, the assigned dev is the one responsible for merging and other devs shouldn't merge without asking him first. As a consequence

- Don't assign PR to other devs unless you've asked first
- Devs should feel completely free to de-assign themselves if they don't want/don't have the time to deal with a PR.
- Unassigned pull requests are up for grab. Any dev can merge them.

This does not mean that other devs can't comment on an assigned request. This is more of an "I am working on this area I want to look into it" kind of lock. Any dev should feel free to comment on any request or ask for comments from any other dev (by including the name in the comment, there is a special wiki markup in comments for that).

Commit messages

When committing think of clear commit messages that describe your changes for that specific commit. This will make it easier for others to follow up with your work and find problems if they occur. The commit messages are pulled with their commits to our main repository once your pull request is merged. Since our Redmine has access to this it processes the commit messages. So you are able to link to issues within Redmine from your commit message - or even close a bug without visiting the bug tracker. Just type "this fixes #1234" to close or just "#1234" just to mention the issue somewhere in your commit message. Have a look at the Bug_Workflow, section "Redmine git integration" for further information.

Please make sure that your commit messages follow the format that is generally considered nice:
A short summary in the first line, no more than 50 characters long. When you can't put everything in that one line, then add one empty line and then a more verbose description with lines no longer than 78 characters. The reason for this is that commit messages formated like this can nicely be sent as emails and read in the terminal.

TODO

- add a note about working on a clean copy to avoid pulling in other people's changes