

{{lastupdated_at}} by {{lastupdated_by}}

GitHub repository workflow for developers

Workflow summary

This section gives a summary of the workflow once you have successfully forked the repository, and details are given for each of these steps in the following sections.

- The devel branch is the trunk.
- When you are starting a new set of changes, fetch any changes from the devel branch, and start a new feature branch from that.
- Make a new branch for each separable set of changes - "one task, one branch".
- Name your branch for the purpose of the changes, starting with the issue number, followed by the purpose - e.g. 536-refactor-database-code.
- If you can possibly avoid it, avoid merging devel or any other branches into your feature branch while you are working.
- If you do find yourself merging from devel, consider Rebasing on devel.
- Ask on the [Developer forum](#) if you get stuck.
- Ask for code review!

This way of working helps to keep work well organized, with readable history.

Updating the mirror of devel

From time to time you should fetch the latest changes from the devel branch from GitHub.

```
$ git fetch upstream
From git://github.com/gammalib/gammalib
* [new branch]   devel    -> upstream/devel
* [new branch]   master    -> upstream/master
* [new branch]   release   -> upstream/release
* [new branch]   integration -> upstream/integration
```

This will pull down any commits you don't have, and set the remote branches to point to the right commit.

Making a new feature branch

When you are ready to make some changes to the code, you should start a new branch. We call this new branch a *feature branch*.

The name of a feature branch should always start with the [Redmine issue](#) number, followed by a short informative name that reminds yourself and the rest of us what the changes in the branch are for. For example 735-add-ability-to-fly, or 123-bugfix. If you don't find a [Redmine issue](#) for your feature, [create one](#).

Create the feature branch using

```
$ git fetch upstream
From git://github.com/gammalib/gammalib
* [new branch]   devel    -> upstream/devel
* [new branch]   master    -> upstream/master
* [new branch]   release   -> upstream/release
* [new branch]   integration -> upstream/integration
$ git branch 007-my-new-feature upstream/devel
Branch 007-my-new-feature set up to track remote branch devel from upstream.
$ git checkout 007-my-new-feature
Switched to branch '007-my-new-feature'
```

The first command makes sure that the latests commits are fetched from the GitHub repository, the second command creates the

feature branch, and the last command switches to the feature branch.

Generally, you will want to keep your feature branches on your public GitHub fork. To do this, you git push this new branch up to your GitHub repo:

```
$ git push origin 007-my-new-feature
Total 0 (delta 0), reused 0 (delta 0)
To git@github.com:jknodlseder/gammalib.git
* [new branch]    007-my-new-feature -> 007-my-new-feature
```

From now on git will know that 007-my-new-feature is related to the 007-my-new-feature branch in the GitHub repo.

The editing workflow

Overview

Here a typical command sequence, where a file is added, the change is committed, and the commit is pushed to the repository.

```
$ git add my_new_file
$ git commit -am 'NF - some message'
[007-my-new-feature 403d7d2] NF - some message
1 files changed, 1 insertions(+), 0 deletions(-)
create mode 100644 my_new_file
$ git push origin
Counting objects: 4, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 303 bytes, done.
Total 3 (delta 1), reused 0 (delta 0)
To git@github.com:jknodlseder/gammalib.git
04bab2c..403d7d2 007-my-new-feature -> 007-my-new-feature
```

Rebasing on devel

Eventually, the devel branch has advanced while you developed the new feature. In this case, you should rebase your code before asking for merging. Rebasing is done using:

```
$ git fetch upstream
remote: Counting objects: 5, done.
remote: Compressing objects: 100% (1/1), done.
remote: Total 3 (delta 2), reused 3 (delta 2)
Unpacking objects: 100% (3/3), done.
From git://github.com/gammalib/gammalib
04bab2c..ccba491  devel    -> upstream/devel
$ git checkout 007-my-new-feature
Already on '007-my-new-feature'
Your branch and 'upstream/devel' have diverged,
and have 1 and 1 different commit(s) each, respectively.
$ git branch tmp 007-my-new-feature
$ git rebase upstream/devel
First, rewinding head to replay your work on top of it...
Applying: NF - some message
```

Here we created a backup of 007-my-new-feature into tmp for safety.

When all looks good you can delete your backup branch using

```
$ git branch -D tmp
```

Deleted branch tmp (was 403d7d2).

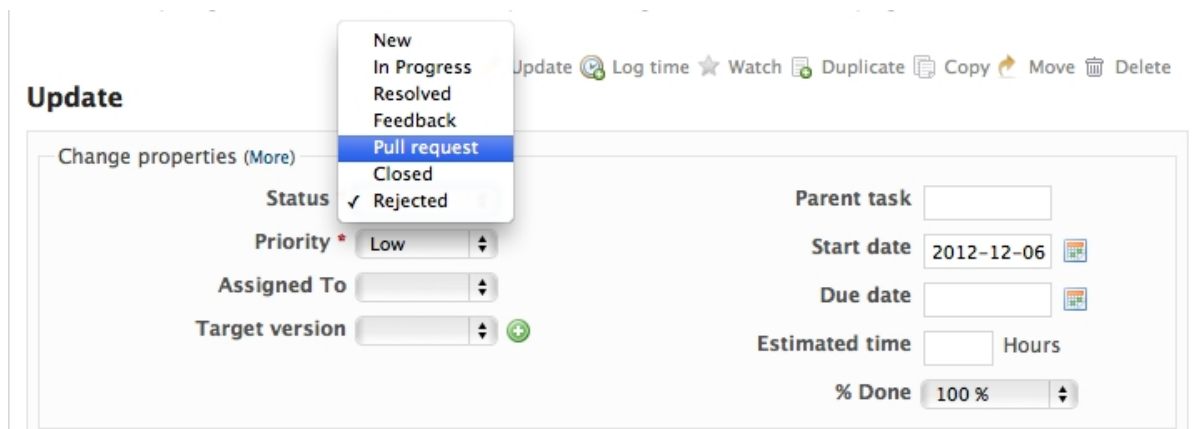
If your feature branch is already on GitHub and you rebase, you will have to force push the branch; a normal push would give an error. Use this command to force-push:

```
$ git push -f origin 007-my-new-feature
Counting objects: 8, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (5/5), done.
Writing objects: 100% (6/6), 611 bytes, done.
Total 6 (delta 3), reused 0 (delta 0)
To git@github.com:jknodlseder/gammlib.git
+ 403d7d2...816ac2c 007-my-new-feature -> 007-my-new-feature (forced update)
```

Note that this will overwrite the branch on GitHub, i.e. this is one of the few ways you can actually lose commits with git. Also note that it is never allowed to force push to the main GammaLib repo (typically called upstream), because this would re-write commit history and thus cause problems for all others.

Asking for your changes to be reviewed or merged

When you are ready to ask for someone to review your code and consider a merge, change the status of the issue you're working on to *Pull Request*:

A screenshot of a web-based task management interface. At the top, there's a row of icons: 'Update', 'Log time', 'Watch', 'Duplicate', 'Copy', 'Move', and 'Delete'. Below this is a 'Update' section with a 'Change properties (More)' link. A dropdown menu is open, showing options: 'New', 'In Progress', 'Resolved', 'Feedback', 'Pull request' (highlighted in blue), 'Closed', and 'Rejected'. Below the dropdown, there are several form fields: 'Status' (with a checkmark and 'Rejected'), 'Priority' (set to 'Low'), 'Assigned To' (empty), 'Target version' (empty with a green plus icon), 'Parent task' (empty), 'Start date' (set to '2012-12-06'), 'Due date' (empty), 'Estimated time' (empty with 'Hours' label), and '% Done' (set to '100 %').

In the notes field, describe the set of changes, and put some explanation of what you've done. Say if there is anything you'd like particular attention for - like a complicated change or some code you are not happy with.

If you don't think your request is ready to be merged, just say so in your pull request message. This is still a good way of getting some preliminary code review.

Some other things you might want to do

Delete a branch on GitHub

```
$ git checkout devel
Switched to branch 'devel'
$ git branch -D 007-my-new-feature
Deleted branch 007-my-new-feature (was 816ac2c).
$ git push origin :007-my-new-feature
To git@github.com:jknodlseder/gammlib.git
- [deleted]      007-my-new-feature
```

Note the colon : before 007-my-new-feature. See also: <http://github.com/guides/remove-a-remote-branch>

Files

