{{lastupdated_at}} by {{lastupdated_by}}

Proposal of Git workflow for developers

As developer, you have two options for sharing your code with the project: either you use the central GammaLib repository https://cta-git.irap.omp.eu/gammalib or you use a fork on Github (https://github.com/gammalib/gammalib). In the following, the workflow for both options is explained.

Using the central repository

Using GitHub

Creating a fork

A first step you need to create a fork of gammalib on GitHub. You need to do this only once. The instructions here are very similar to the instructions at http://help.github.com/fork-a-repo/ - please see that page for more details. We're repeating some of it here just to give the specifics for the GammaLib project, and to suggest some default names.

Set up and configure a GitHub account

If you don't have a GitHub account, go to the GitHub page, and make one.

You then need to configure your account to allow write access - see the Generating SSH keys help on GitHub Help.

Create your own fork of a repository

The following example shows how to fork the GammaLib repository:

1. Log into your GitHub account.
2. Goto to https://github.com/gammalib/gammalib
3. Click on the fork button
4. Select your username (in this example jknodlseder)

After a short pause, you should find yourself at the home page for your own forked copy of GammaLib (in this example https://github.com/jknodlseder/gammalib).

Setting up the fork to work on

As next step, you have to setup the fork on your local computer. You also need to do this only once.

Clone your fork to the local computer

Use the command

$ git clone git@github.com:<user>/gammalib.git

```
remote: Counting objects: 22147, done.
remote: Compressing objects: 100% (4911/4911), done.
remote: Total 22147 (delta 17346), reused 21980 (delta 17179)
Receiving objects: 100% (22147/22147), 80.25 MiB | 42 KiB/s, done.
Resolving deltas: 100% (17346/17346), done.
```

to clone the GammaLib repository from GitHub. Here, <user> is your GitHub user name (jknodlseder in the example above).

Connect to the GammaLib repository

Now connect to the GitHub GammaLib repository using

```
$ cd gammalib
$ git remote add upstream git://github.com/gammalib/gammalib.git
```

upstream here is just the arbitrary name we're using to refer to the main GammaLib repository.

Note that we've used git:// for the URL rather than git@. The git:// URL is read only. This means we that we can't accidentally (or deliberately) write to the upstream repo, and we are only going to use it to merge into our own code.

You may verify that the connection has been established with

```
$ git remote -v
origin    git@github.com:jknodlseder/gammalib.git (fetch)
origin    git@github.com:jknodlseder/gammalib.git (push)
upstream    git://github.com/gammalib/gammalib.git (fetch)
upstream    git://github.com/gammalib/gammalib.git (push)
```

Your fork is now set up correctly, and you are ready to hack away.

Deleting your master branch

It may sound strange, but deleting your own master branch can help reduce confusion about which branch you are on. See deleting master on github for details.

To delete the master branch, type

```
$ git checkout devel
Already on 'devel'

$ git branch -D master
error: branch 'master' not found.

$ git push origin :master
To git@github.com:jknodlseder/gammalib.git
 - [deleted]         master
```

Don't worry if you get the message error: branch 'master' not found., this just signals that you never checked out the master branch.

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.

- Don't use your master branch for anything. Consider deleting it.
- 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
```

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

$ 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:

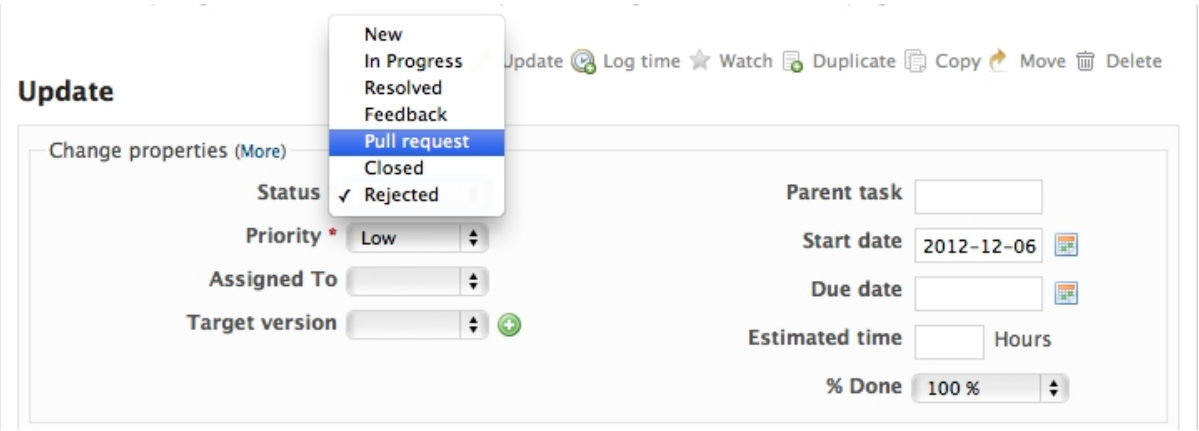
```
$ git push -f origin 007-my-new-feature
Counting objects: 8, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (5/5), done.
Writing objects: 100% (6/6), 611 bytes, done.
Total 6 (delta 3), reused 0 (delta 0)
To git@github.com:jknodlseder/gammalib.git
```

```
+ 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*:



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/gammalib.git
 - [deleted]         007-my-new-feature
```

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

Proposal of Git workflow for maintainers

Using the central repository

Using GitHub

Setting up the repo to work on

Clone central GammaLib repository to the local computer

```
$ git clone https://<manager>@cta-git.irap.omp.eu/gammalib
Cloning into 'gammalib'...
Password:
remote: Counting objects: 22150, done.
remote: Compressing objects: 100% (7596/7596), done.
```

```
remote: Total 22150 (delta 17330), reused 18491 (delta 14497)
Receiving objects: 100% (22150/22150), 80.12 MiB | 192 KiB/s, done.
Resolving deltas: 100% (17330/17330), done.
```

where <manager> is the user name of the integration manager (e.g. jknodlseder).

Connect to the developer's GitHub repository

Now connect to the GitHub repository of the developer using

```
$ cd gammalib

$ git remote add developer git://github.com/developer/gammalib.git

$ git remote -v
developer    git://github.com/developer/gammalib.git (fetch)
developer    git://github.com/developer/gammalib.git (push)
origin    https://jknodlseder@cta-git.irap.omp.eu/gammalib (fetch)
origin    https://jknodlseder@cta-git.irap.omp.eu/gammalib (push)
```

developer here is the GitHub user name of the developer from which we want to integrate changes.

Integrate the feature branch

Fetch the developer's feature branch

```
$ git fetch developer
remote: Counting objects: 4, done.
remote: Compressing objects: 100% (1/1), done.
remote: Total 3 (delta 1), reused 3 (delta 1)
Unpacking objects: 100% (3/3), done.
From git://github.com/developer/gammalib
 * [new branch]      007-my-new-feature -> developer/007-my-new-feature
 * [new branch]      INCLUDES_to_AM_CPPFLAGS -> developer/INCLUDES_to_AM_CPPFLAGS
 * [new branch]      devel      -> developer/devel
 * [new branch]      pep8       -> developer/pep8
 * [new branch]      release    -> developer/release
 * [new branch]      integration -> developer/integration

$ git branch 007-my-new-feature --track developer/007-my-new-feature
Branch 007-my-new-feature set up to track remote branch 007-my-new-feature from developer.

$ git checkout 007-my-new-feature
Switched to branch '007-my-new-feature'
```

Rebase

If there are only a few commits, consider rebasing to upstream:

```
$ git fetch origin
$ git rebase origin/devel
```

If there are a longer series of related commits, consider a merge instead:

```
$ git merge --no-ff origin/devel
```

Note the --no-ff above. This forces git to make a merge commit, rather than doing a fast-forward, so that these set of commits branch

off devel then rejoin the main history with a merge, rather than appearing to have been made directly on top of devel.

Check the history

Now, in either case, you should check that the history is sensible and you have the right commits:

```
$ git log --oneline --graph
$ git log -p origin/devel..
```

The first line above just shows the history in a compact way, with a text representation of the history graph. The second line shows the log of commits excluding those that can be reached from devel (origin/devel), and including those that can be reached from current HEAD (implied with the .. at the end). So, it shows the commits unique to this branch compared to devel. The -p option shows the diff for these commits in patch form.

Merge into integration branch

```
$ git checkout integration

$ git merge 007-my-new-feature
Updating ccba491..562f236
Fast-forward
 my_new_file |    1 +
 1 files changed, 1 insertions(+), 0 deletions(-)
 create mode 100644 my_new_file

$ git commit -am 'Merged 007-my-new-feature.'

$ git push origin
Password:
Counting objects: 4, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 306 bytes, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: To https://github.com/gammalib/gammalib.git
remote:    ccba491..562f236  integration -> integration
remote:  * [new branch]      github/integration -> github/integration
To https://jknodlseder@cta-git.irap.omp.eu/gammalib
   ccba491..562f236  integration -> integration
```

Verify the integration

The push will automatically launch the integration pipeline on Jenkins.

You should verify the all checks are passed with success.

Merge into devel

```
$ git checkout devel
Switched to branch 'devel'

$ git merge integration
Updating ccba491..562f236
Fast-forward
 my_new_file |    1 +
 1 files changed, 1 insertions(+), 0 deletions(-)
 create mode 100644 my_new_file

$ git push origin
```

Files

fork-gammalib.jpg	138 KB	12/08/2012	Knödlseder Jürgen
pull-request.jpg	67.9 KB	12/09/2012	Knödlseder Jürgen