{{lastupdated_at}} by {{lastupdated_by}}

Central repository workflow for developers

Workflow summary

This section gives a summary of the workflow. 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.

Cloning the code from the central repository

To clone the GammaLib source code, type

```
$ git clone https://user@cta-git.irap.omp.eu/gammalib
```

This will create a directory called ctools under the current working directory that will contain the ctools source code.

This will create a directory called gammalib under the current working directory that will contain the GammaLib source code. Here, user is your user name on the CTA collaborative platform at IRAP (the same you used to connect for reading this text). Executing this command will open a window that asks for a password (your password on the CTA collaborative platform at IRAP), and after confirming the password, your will get a clone of gammalib on your disk.

Note that the default branch from which you should start for software developments is the devel branch, and the git clone command normally automatically clone this branch from the central repository. Make sure that you will never use the master branch to start your software developments, as pushing to the master branch is not permitted. The master branch always contains the code of the last GammaLib release, and serves as seed for hotfixes. To know which branch you're on, type

```
$ git status
```

In case that you're not on devel, switch to the devel branch using

```
$ git checkout devel
```

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.

First make sure that you start from the devel branch by typing

```
$ git checkout devel
```

Then create a new feature branch using

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

This creates and switches automatically to the branch 007-my-new-feature.

Write code and commit your changes

Now you're ready to write some code and commit your changes. Suppose you edited the pyext/GFunction.i file. After editing, you can check the status by typing

```
$ git status
# On branch 007-my-new-feature
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#    modified:   pyext/GFunction.i
#
no changes added to commit (use "git add" and/or "git commit -a")
```

Git signals here that you have to add and commit your change.

You'll do this by typing

```
$ git add pyext/GFunction.i
$ git commit -am 'Remove GTools.hpp include.'
  [007-my-new-feature 065af59] Remove GTools.hpp include.
   1 files changed, 0 insertions(+), 1 deletions(-)
```

Continue with editing and commit all your changes.

Pushing your work into the IRAP git repository

To push your new feature branch into the IRAP git repository, type

```
$ git push origin 007-my-new-feature
  Password:
Counting objects: 7, done.
Delta compression using up to 48 threads.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 390 bytes, done.
Total 4 (delta 3), reused 0 (delta 0)
remote: To https://github.com/gammalib/gammalib.git
remote:    8139b97..07946a3  github/integration -> github/integration
remote:  * [new branch]      007-my-new-feature -> 007-my-new-feature
To https://jknodlseder@cta-git.irap.omp.eu/gammalib
 * [new branch]      007-my-new-feature -> 007-my-new-feature
```

This pushed your modifications into the repository. Note that this created a new branch in the repository.

Rebasing on devel (optional)

Eventually, the devel branch has advanced while you developed the new feature. In this case, you should rebase your code before asking for merging. If you're not sure what this means, better skip this step and leave it to the integration manager to work out how to deal with the diverged branches. If you feel however you want to help the manager with integration, rebase your branch using

```
$ git fetch origin
  Password:
$ git pull origin 007-my-new-feature
  Password:
  From https://cta-git.irap.omp.eu/gammalib
   * branch            007-my-new-feature -> FETCH_HEAD
  Already up-to-date.
$ git checkout 007-my-new-feature
  Already on '007-my-new-feature'
  Your branch and 'devel' have diverged,
  and have 1 and 1 different commit(s) each, respectively.
$ git branch tmp 007-my-new-feature
$ git rebase devel
  First, rewinding head to replay your work on top of it...
  Applying: Remove GTools.hpp include.
```

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

If your feature branch is already in the central repository 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 pull origin 007-my-new-feature
  Password:
  From https://cta-git.irap.omp.eu/gammalib
   * branch            007-my-new-feature -> FETCH_HEAD
  Merge made by the 'recursive' strategy.
$ git push -f origin 007-my-new-feature
  Password:
  Counting objects: 5, done.
  Delta compression using up to 4 threads.
  Compressing objects: 100% (3/3), done.
  Writing objects: 100% (3/3), 492 bytes, done.
  Total 3 (delta 2), reused 0 (delta 0)
  remote: To https://github.com/gammalib/gammalib.git
  remote:    1f00150..20cf8c5  007-my-new-feature -> 007-my-new-feature
  remote:    35d6408..1f00150  github/007-my-new-feature -> github/007-my-new-feature
  To https://jknodlseder@cta-git.irap.omp.eu/gammalib
     1f00150..20cf8c5  007-my-new-feature -> 007-my-new-feature
```

Note that this will overwrite the branch in the central repository, i.e. this is one of the few ways you can actually lose commits with git.

Note: I don't really understand why I need the git pull before the push, but without I get the following error message:

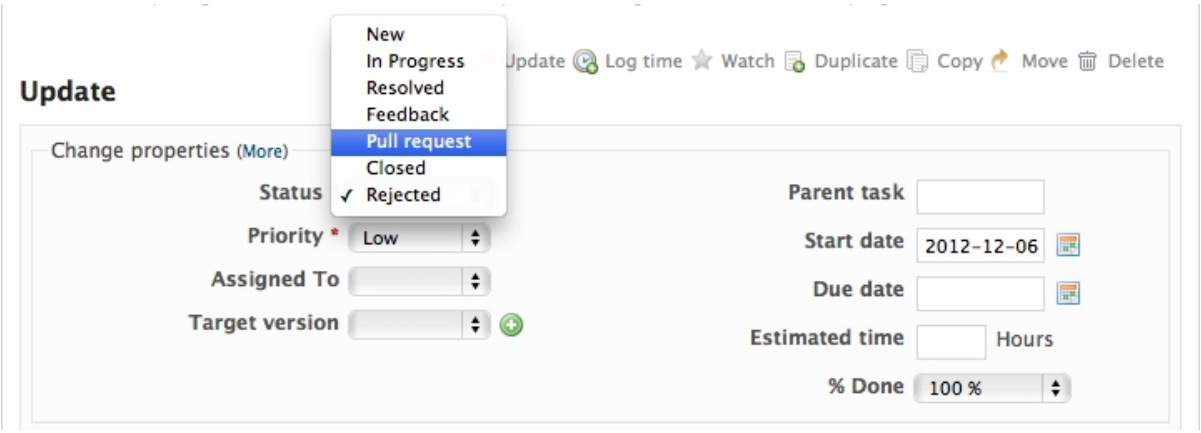
```
$ git push -f origin 007-my-new-feature
  Password:
  Counting objects: 11, done.
  Delta compression using up to 4 threads.
  Compressing objects: 100% (7/7), done.
  Writing objects: 100% (7/7), 1.44 KiB, done.
  Total 7 (delta 6), reused 0 (delta 0)
  remote: error: denying non-fast-forward refs/heads/007-my-new-feature (you should pull first)
```

When all looks good you can delete your backup branch using

```
$ git branch -D tmp
  Deleted branch tmp (was 1f00150).
```

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.

Files

pull-request.jpg	67.9 KB	12/09/2012	Knödlseder Jürgen