Table of Contents

git instructions for the RICH mirror alignment

Because we are no longer using SVN.

Table of Contents

Nota Bene

NEVER edit the code directly at the pit (e.g. in the AlignmentRelease area). When using multiple projects with git, things can get extremely confused and it will be very difficult to find and merge any changes into to code!

Please make your own branch of any code you change. Then simply `lb-checkout` the code from your branch into the AlignmentRelease area when you want to install it.

Read the rest of this TWiki first, but more information about the code and how it exists at the pit can be found here: LHCbRichMirrorAlignCodeOnline

git

It is time to git! Please read everything and **please follow the workflow Anatoly describes**. If you have any questions **email the alignment-development mailing list**. If we cannot answer your question, email Jordi, and if all else fails then Marco Clemencic / lhcb-core-soft@cernNOSPAMPLEASE.ch

The basic idea behind git is that "branches" are used to develop new features of the code, typically each branch forks from the main **master** branch (the "default" branch) of a git repository, and each new branch is isolated from the other branches, until one chooses to **merge** them. Typically we use these other branches for development, and then merge them back to the master branch when the new feature is tested and ready. At LHCb we have to ask for a **merge request** in CERN's GitLab to merge one of our branches into (typically) the master branch of a Project. Then the Project manager can decide whether to allow the merge to occur.

NOTA BENE: **However**, Panoptes is currently using **two** main branches. One is called **2018-patches** and applies only to 2018 operations, the other is called **master** and will be used in the upgrade. This significantly changes our workflow:

- ALWAYS lb-checkout out code from **2018-patches** and commit any changes to a new branch
- As usual, set up a merge request when ready to merge the new branch and make CERTAIN that you are merging into the **2018-patches** branch.
- Within the discussion section of the merge request, kindly ask Jordi to rebase these commits into the **master** branch once the merge request is approved.
    - Jordi will happily do this so long as there are no conflicts. If there is a conflict then **you** will have to resolve it because "the author knows best".
        - To solve it, you may have to cherry-pick commits *from* **2018-patches** into a *second* new branch (see below), and then set up a merge request when ready to merge the *second* new branch into the **master** branch.

This seems complicated but it's the only way we can ensure that our changes go into both branches, as the master branch will soon have much code stripped away and changed in favor of the upgrade era code.

Standard documentation and other git stuff

- The main resource
  - https://twiki.cern.ch/twiki/bin/view/LHCb/Git4LHCb
- Other References (don't know how useful these are):
  - https://git-scm.com/doc
  - http://rogerdudler.github.io/git-guide/
  - http://gitref.org
- Places where people who actually know how to use git hang out at:
  - https://github.com
  - https://about.gitlab.com

Transition of Panoptes from svn to git.

The Panoptes project has been moved to gitlab. https://gitlab.cern.ch/lhcb/Panoptes
The lhcb-rich-software egroup is linked to the gitlab group such that all members have "developer" access.

There is a major change in the workflow, since we cannot use tagged packages in git anymore.

The main consequences are the following:

- CmakeLists and cmt/requirements **still** need to have a package version number, these should be updated when necessary.
- **However**, packages don't have version tags anymore. From now on, a Panoptes project tag affects all its packages. What used to be "version v2r26 of Rich/RichOnlineMonitors" is now the "Rich/RichOnlineMonitors that comes with Panoptes v5r6."
- The tag collector becomes obsolete from the point of view of Panoptes.
- Since we still need to release Panoptes versions, **all packages should at least compile when they are added to the 2018-patches / master branch of the git repository**. For this reason, it is important to have meaningful development branches for stuff that is not yet stable, which will be merged once the new features are fully implemented and tested.

The recommended workflow to add features to packages is the following:

- Implement the features in a branch.
- Test them.
- Push the branch to an upstream repository (e.g. a fork of the project in gitlab).
- Submit a merge request, so the new features are included in the 2018-patches / master branch.

Please get used to the git workflow ASAP.

The JIRA rich project.

Jordi has created a JIRA (project and) component group for (the RICH group and) the mirror alignment, which you can find here. Jordi is not an expert in JIRA, so Jordi cannot guarantee that the configuration that he has done is ideal. Any suggestions are welcome.

The JIRA group does NOT impose any change in the workflow. We could keep managing issues by email, or external service, or keep track of your tasks by tweeting them... However, people who have used JIRA have a good experience in their opinion, so **we have decided to use it**.

In JIRA there are these things called "*components*", which correspond to different unrelated topics (or mostly unrelated) within a project. Jordi has created four software components: monitoring (for Panoptes issues), **alignment (for mirror alignment)**, piquet (for piquet software tools) and reconstruction (for reconstruction software). Please make sure any new JIRA issues for the alignment are created with component **alignment**.

(In principle, JIRA can be useful for hardware projects as well, so feel free to create as many components as you want for hardware and for other software topics.)

Rich JIRA issues have references like RICH-XXXX, where XXXX is a number. **If you mention a JIRA issue in a git commit, JIRA will automatically create a comment in its corresponding issue page**, so please do this (example below).

Members of lhcb-rich-software have "developers" access to the rich JIRA project. (Let Jordi know if you prefer that Jordi changes it to lhcb-rich.)

Anatoly's example git workflow for Panoptes [or Rec]

Anatoly was kind enough to provide us with his git workflow so far (and thanks to Jordi for help). He is still learning as are we, but this will certainly help you get started.

Paras is keeping this updated as new information arrives.

NOTE that this is an example, your use case may be different!

To build

Change usernames and email addresses below as appropriate

[Change projects throughout below, if instead of Panoptes you are working with e.g. Rec]

Create a new shell

- `ssh -Y asolomin@lxplusNOSPAMPLEASE.cern.ch` (where asolomin = your username)

Know your git version

- `git version` (there may be version-dependent commands below)

Only once and forever do these three things:

- Follow Prerequisites
- `git config --global user.name "Anatoly Solomin"` (where Anatoly Solomin = your name)
- `git config --global user.email anatoly.solomin@cern.ch` (where anatoly.solomin@cernNOSPAMPLEASE.ch is your email address)

These should be done every time you want to start coding (**NOTE** the choice of User_release_area is completely up to you, actually, but should be in **your** AFS area):

- `mkdir -p /afs/cern.ch/user/a/asolomin/public/rich_align` (where asolomin = your username)
- `setenv User_release_area /afs/cern.ch/user/a/asolomin/public/rich_align` (where asolomin = your username)
- `LbLogin` (for some reason you have to run this after setting the user release area)
- `cd $User_release_area`

Then this you do only **once**, when you start from scratch (**though perform it separately when starting from each new Panoptes release**, here we assume we are working with 2018-patches):

- `lb-dev --nightly-cvmfs --nightly lhcb-2018-patches Panoptes/2018-patches`
    - This creates a local git repository in `PanoptesDev_2018-patches`. This is called a "satellite project".
    - (**NOTE** we do need to use the nightlies, unless we know that other people are working on code that may affect us... simply use `lb-dev Panoptes/2018-patches` instead if this is not the immediate case)

These you do as long as you are working with the project and version in the previous line (**NOTE** the "Dev"):

- `cd $User_release_area/PanoptesDev_2018-patches`
- `git lb-use Panoptes`
    - This adds the gitlab remote to the local repository of the satellite project, and **fetches** that remote (makes a local copy of the remote repository, although you still don't see it in your working tree).

The output of this last command will give you information like this:

```
 * [new branch]      master     -> Panoptes/master <br>
 * [new branch]      2018-patches     -> Panoptes/2018-patches <br>
 * [new tag]         v1r0       -> Panoptes/v1r0 <br>
[...] <br>
 * [new tag]         v8r1       -> Panoptes/v8r1
```

You will see that there are several references that you can use. Some of them are branches (in this example, 2018-patches and master) and some of them are tags. You will refer to them as Panoptes/[reference], as in the last column of the previous output.

Let's say you want to develop on top of the Rich/RichMirrorAlignmentOnline package that you can find on the 2018-patches branch. To checkout a package (*e.g* before you start work on a new package for the first time):

- `git fetch --all` (**ALWAYS do this before any lb-checkout, it updates the local information about branches with the current information from the git repository.**)
- `git lb-checkout Panoptes/2018-patches Rich/RichMirrorAlignmentOnline` (**or perhaps a different package that you might be working on (e.g. Rich/RichMirrAlign)**)
    - This is the git equivalent of checking out the package from the SVN head.

Now you may edit the code and add new features. You would obviously not necessarily edit the same files as shown in this example, but whatever pieces of code you actually wanted to edit.

Of course you should use your editor of choice (*e.g.* `emacs`, `vi`, etc...) or `nano` as in this example:

- `nano Rich/RichMirrorAlignmentOnline/CMakeLists.txt`
- `nano Rich/RichMirrorAlignmentOnline/cmt/requirements`
- `nano Rich/RichMirrorAlignmentOnline/doc/release.notes`
    - **If you make any code change, always update the release notes! (only change the other two files if the package "version" needs to be updated)**

Check that your code compiles, particularly if you made a change to C++ code:

- `make -j 8` (use 8 cores)

Get a list of files that changed:

- `git status`
    - lets you know which files have changed since your last `git lb-push ...`

Or better yet, check all of your changes:

- `git diff`
    - shows all changes between what you checked out and your last commit.

For **everything** that changed **and** any new files use `git add` (*e.g.* for the previous example):

- `git add Rich/RichMirrorAlignmentOnline/doc/release.notes` (**Please always** update the release notes!!!)
- `git add Rich/RichMirrorAlignmentOnline/CMakeLists.txt`
- `git add Rich/RichMirrorAlignmentOnline/cmt/requirements`

Then make sure to update the commit message (for example):

- `git commit -m 'RICH-9999 Updated Release Notes.'`

RICH-9999 is the associated JIRA task (if one exists for the task you are working on), which automatically links the commit (or later, merge request) to the JIRA task

Note, there is a shortcut for the above two sets of commands:

- `git commit -a -m 'RICH-9999 Updated Release Notes.'`
    - is a combination of `git add` for all files that changed (**but only those that already existed, new files you must add manually**) and `git commit`

If after your commit, you edit a file, say `Rich/RichMirrorAlignmentOnline/doc/release.notes` you **must** `git add` it again:

- `git add Rich/RichMirrorAlignmentOnline/doc/release.notes`

If after your commit, whether you changed any files or not, you wish to edit or expand your commit description before a push use:

- `git commit --amend`

As new commits are only stored in your local repository, there's no cost to committing often.
You should try to make a new commit every time you've made modifications that can be considered a single unit of changes.

You will now give your branch a name. You could choose anything (but pick something unique!). However let's try to stick to the following prescription: username-title-YYYYMMDD (*e.g.* asolomin-MirrAlign-20180703).
So for example, to push your changes to a new branch called asolomin-MirrAlign-20180703:

- `git lb-push Panoptes asolomin-MirrAlign-20180703`

Here, your equivalent of `username-title-YYYYMMDD` is the name of your development branch, and you will be able to see it in gitlab.

If you then would like the commits in `username-title-YYYYMMDD` to be merged into `2018-patches`, you must perform a Gitlab merge request. You may submit a merge request for asolomin-MirrAlign-20180703 from the Panoptes gitlab page You can also use the hyperlink provided when you commit, though if it's a sensitive change you may want to look at your commit in gitlab first. NOTE: Your pushed branch (that contains your commit) **MUST COMPILE** locally to get accepted (though it may not compile against the nightlies, consult Chris and Jordi if you find this problem).

Important: if you noticed a mistake, fix it, and immediately want to commit a corrected version to a new branch, you should

- `git lb-push Panoptes asolomin-MirrAlign-20180703-2` (i.e. any **different** name)

Then, submit a merge request for asolomin-MirrAlign-20180703-2, instead of asolomin-MirrAlign-20180703 Go ahead and delete the branch.

Now say you already made a merge request, then you could commit a fix and push it to the same branch for which you made the merge request. Before you start work on the fix however, make sure to change the first word of the title of your merge request to "WIP: " (work-in-progress). This instructs the Project manager to wait until you remove "WIP: " to merge the branch.

If you no longer want the merge request to be accepted (e.g. it was done by mistake) then close the merge request. If the associated branch is no longer in use you may delete it from the Panoptes gitlab page.

To continue working on the same branch after a git lb-push you don't need to do anything, just change code, commit, and push, and repeat the cycle until ready for a merge request or until your branch is merged. If you already opened a merge request for a branch, you can still keep pushing to that same branch and it will become part of the existing merge request as well.

This shouldn't happen in our workflow, but just so you know about it... If changes were made to your branch [*e.g.* in GitLab] **by** someone else, and then you want to keep working on the branch:

- `git fetch --all` # (you **must** do this in order to get access to remote changes in Panoptes/2018-patches or any other branch)
- `git lb-checkout Panoptes/asolomin-MirrAlign-20180703 Rich/RichMirrorAlignmentOnline`

After the merge requests are applied and only after, when you are ready to start work again you will want to update your local packages from the 2018-patches, in case there were any changes in the 2018-patches in the meantime:

- `cd $User_release_area/PanoptesDev/2018-patches`
- `git fetch --all` # (**Remember**, you **must** do this in order to get access to remote changes in Panoptes/2018-patches or any other branch)
- `git lb-checkout Panoptes/2018-patches Rich/RichMirrorAlignmentOnline` (and the same for ALL other packages you checked out from 2018-patches, unless you are still working on a branch of a different package)

*If someone else has lb-pushed a new feature branch called `that-branch` to Panoptes, you can bring that branch instead to your local copy with

- `git fetch --all`
- `git lb-checkout Panoptes/that-branch Rich/RichMirrorAlignmentOnline`

In this manner you can change your local copy of the repository for a package very simply, from that which exists in one branch to another. However you should be typically starting fresh from the `Panoptes/2018-patches` branch before adding new features.

If you want to see what's been happening with respect to the local code interacting with the git repository, at any stage, just go here:

`git log`

- tracks everything you have been doing

If you want it with diffs:

`git log -U3` (3 can be changed to any number)

- git log with diffs from 3 (or another number) lines before to 3 (or another number) lines after any changes

Please also read the further notes on the workflow.

To run

Say you wanted to run a test of your code. Right now we can only test Rich/RichMirrorAlignmentOnline on the farm. But if you edited Rich/RichMirrAlign you could test it in the following manner:

new shell (e.g. `ssh -Y asolomin@lxplusNOSPAMPLEASE.cern.ch`)

- `ssh lbgw -Y`
- `cd /group/online/AligWork/MirrorAlignments`
- `scp -pr Rich1/20180628_172734 asolomin@lxplusNOSPAMPLEASE.cern.ch:/afs/cern.ch/user/a/asolomin/public/rich_align_test/Ri`
- `exit`
- `cd /afs/cern.ch/user/a/asolomin/public/rich_align_test/Rich1/`
- `cp -pr 20180628_172734 20180628_172734_test`
- `setenv User_release_area /afs/cern.ch/user/a/asolomin/public/rich_align`
- `LbLogin`
- `cd $User_release_area/PanoptesDev/2018-patches`
- `make install`
- `cd /afs/cern.ch/user/a/asolomin/public/rich_align_test/Rich1/20180628_172734_test` (or wherever your test is)
- `lb-run PanoptesDev/2018-patches RichMirrAlign.exe Rich1MirrAlign_i0.conf`

Slight differences to the above for the Hlt project

Substitute the following above, where appropriate

- `lb-dev Moore/v25r3` (or latest version, check in gitlab)
- `cd $User_release_area/MooreDev_v25r3`
- `git lb-use Hlt`
- `git fetch --all`
- `git lb-checkout Hlt/2016-patches Hlt/HltSettings` (perhaps you will be asked by HLT Operations to use a different branch, or need to edit a different package)
- `nano Hlt/HltSettings/python/HltSettings/Physics_pp_Draft2016.py` (example)
- `nano Hlt/HltSettings/doc/release.notes`
- `make -j 8` (check that your code compiles!)
- `git add Hlt/HltSettings/python/HltSettings/Physics_pp_Draft2016.py`
- `git add Hlt/HltSettings/doc/release.notes`
- `git commit -m 'RICH-9999 Rich Mirror Line prescales reduced by a factor of 3'`
- `git commit --amend`

You will now give your branch a name. At this stage you may choose anything (but pick something like {your name}-{something describing your change} that helps you keep track of things). Say you pick `pnaik-20160630-RICHMirrorAdjust`.
This will push your changes to the `pnaik-20160630-RICHMirrorAdjust` branch:

- `git lb-push Hlt pnaik-20160630-RICHMirrorAdjust`

Then, submit a merge request for `pnaik-20160630-RICHMirrorAdjust` from the Hlt gitlab page. Create a merge request with the `2016-patches` branch. NOTE: Your pushed branch (that contains your commit) **MUST COMPILE** to get accepted.

Important: if you noticed a mistake and immediately want to commit a corrected version, you should

- `git lb-push Hlt pnaik-20160630-RICHMirrorAdjust2` (i.e. any **different** name)

Then, submit a merge request for pnaik-20160630-RICHMirrorAdjust2, instead of pnaik-20160630-RICHMirrorAdjust

If you no longer want the merge request to be accepted (e.g. it was done by mistake) then close the merge request. If the associated branch is no longer in use you may delete it from the Hlt gitlab page.

Cherry-picking commits from 2018-patches into master

In the new workflow, we always work with 2018-patches as our starting branch, and we will use 2018-patches as the default branch at the pit as well. Thus make **sure** that your merge requests are to merge your branch into **2018-patches**. However, since for the time being we also want the upgrade code to have all of the changes we made, we are required to either:

1. Rebase the commits that are in your merge request into **master**

or

2. Manually cherry-pick commits we make to 2018-patches to a new branch, and then merge that new branch into **master**.

The former can be done by Jordi as long as there are no merge conflicts (see above on how to request that Jordi do this for you). There are several ways to do the latter, which will not be discussed here. However may be something that you have to do, so if you can't figure it out please contact all the mirror alignment experts for advice.

Notes from Paras about our global workflow

Typically before we submit a merge request, we want to "svn update" or whatever the equivalent of getting changes that others made in the repository over to our branch, and then "rebased". However, as of June 2017 I am still not exactly sure how to do it correctly.

Fortunately with GitLab, we can leave (simple) rebasing to the Project managers. Just prepare your change then commit. You can start your merge request in gitlab right away, but if you don't want it to be actually merged please put "WIP: " in front of the title of the merge request. Then you can commit more to that branch if you want. But **as soon as it is ready** please remove "WIP: " and ask Jordi or Chris to merge the branch. Then after the branch is merged update locally to what is in 2018-patches:

- `git fetch --all` # (you **must** do this in order to get access to remote changes in Panoptes/2018-patches or any other branch)
- `git lb-checkout Panoptes/2018-patches Rich/RichMirrAlign` (or instead of `Rich/RichMirrAlign`, whichever package you are going to work on)

Further details on git commands (thanks to Jordi)

1. You should not associate the 2018-patches / master branch with the SVN trunk. In SVN, tagged packages were expected to work, and committers could add unstable stuff to the trunk. With the new workflow that we believe is reasonable for Panoptes, the 2018-patches / master branch is expected to always be stable, as opposed to the SVN trunk. Unstable contributions to git should take place in branches. So, to summarize, in svn, new contributions go from an unstable trunk to stable branches and from there to release. In git, new contributions go from unstable branches to stable 2018-patches / master and from there to release.

2. Basically the only place a package v-tag should show up is within the CMakeLists and cmt/requirements of each package. Otherwise we do not tag packages officially (they can be tagged unofficially in commits and in the release notes). However, the Panoptes project should be the only thing that is officially v-tagged.

3. The lb-use and lb-checkout commands also solve the case where you want to use packages as plugins from other projects. For example, you can very consistently do:

```
git lb-use Rec
git fetch --all
git lb-checkout Rec/v20r1 Rich/RichAlignment
```

So this is how you would use the Phys/Rec package in Panoptes.

4. New Panoptes releases are created when there's interest in them. It may be for several reasons: to have new features released, to use a new release of Online, to compile with a new CMTCONFIG,... Jordi never opposes creating a new release if it is requested.

5. Currently Jordi Garra Tico and Chris Jones are the only people that can approve merge requests.

Frequently used commands

If there are any we forgot above, we should place them here.

Troubleshooting

Q: I pushed a commit to a branch for which I already have a merge request. However my new commits don't show up in that merge request!

A: Push another commit

- https://cern.service-now.com/service-portal/view-outage.do?n=OTG0036561

Legacy repository information

In the cloned gitlab repository, https://gitlab.cern.ch/LHCb-SVN-mirrors/Panoptes, all contributions to svn up to release v5r5 have been getpacked and committed, but no other history is kept about them. After release v5r5, all the history has been kept.

For reference, Jordi has kept a complete copy of the SVN repository at https://gitlab.cern.ch/lhcb-rich/panoptes In this longer version, all the svn history is kept and, additionally, all tagged releases have been getpacked and attached at their corresponding point in history. This is not intended to be used in practice, just for reference. Jordi will update this repository just once in a while but not very frequently.

Jordi has created the gitlab group lhcb-rich, intended for rich related repositories.
https://gitlab.cern.ch/groups/lhcb-rich

This topic: LHCb > LHCbRichMirrorAlignGitInfo
Topic revision: r35 - 2018-07-03 - AnatolySolomin