

The LOFAR Subversion Repository

The LOFAR repository can be found at <https://svn.astron.nl/LOFAR>. You will need a valid username and password in order to gain access. Both read-only and full access can be provided. You can also [browse the repository](#). If you are a software developer and need access to the LOFAR repository, please contact [Marcel Loose](#).

You may wish to consult the online book [Version Control with Subversion](#) for more information about Subversion.



Before reading further, please read the page [Development Management Policy](#) and the document [Release Management Procedure](#).

Structure of the repository

The LOFAR software repository is set up like most other Subversion repositories.

```
branches/  
tags/  
trunk/
```

These names more or less speak for themselves. Main development is done on the trunk. The tags directory contains tagged release or development versions; the branches directory contains release or development branches. See section [Development Management Policy](#) on this page for details.

Accessing the repository

There are several ways to access the repository. You can [browse](#) the repository with your web browser, or use [ViewVC](#), which provides a very nice interface to the repository.

You can also access the repository with a Subversion client. Usually, you will use this method when you want to checkout code. The simplest client is a command line client, which is available on most modern Linux/Unix systems. Windows users may wish to use [TortoiseSVN](#).

Common actions

This section will provide a very brief overview of the most common actions that you will generally take.

Checking out code

The general syntax for the checkout command is

```
svn checkout URL... [PATH]
```

If PATH is omitted, the basename of the URL will be used as the destination.

To checkout a working copy of the complete trunk (i.e. main development line) of the LOFAR software tree, you should issue the following command:

```
svn co https://svn.astron.nl/LOFAR/trunk LOFAR
```



Do not forget to add /trunk to the URL. Otherwise you will checkout every tag and branch of the LOFAR repository as well, leaving you with hundreds of copies of the complete LOFAR software tree.

However, you may wish to checkout the root directory only, and let CMake do the rest. This can be done by adding the -N flag like:

```
svn co -N https://svn.astron.nl/LOFAR/trunk LOFAR
```



Subversion has a somewhat weird behaviour with respect to a non-recursive checkout (`svn co -N`): non-recursiveness is *sticky*. In fact, the whole concept of recursive and non-recursive checkout is broken in Subversion (see [Issue 695](#) in Subversion's issue tracker). This has some surprising side effects. Notably, if you first do a non-recursive checkout of a top-level directory (e.g. `svn co -N https://svn.astron.nl/LOFAR/trunk LOFAR`) and subsequently run an `svn update` (*without* the -N option) in your workspace, Subversion will tell you that your working copy is up-to-date. You'd think that subsequently issuing a `svn co` without the -N option would solve your problems, but it does not. Subversion will refuse to do this, because it considers a recursive checkout a different URL.

Note: When you want to do a build of one or more LOFAR packages, you can use CMake to do this for you. Please refer to the [CMake for LOFAR](#) for details.

Staying up-to-date

Since you're probably not the only one developing code for LOFAR, you will need to update your workspace from time to time. Do not wait too long, or you might have to go through the hassle of resolving conflicts.

The general syntax is

```
svn update [PATH...]
```

This will merge the changes made to the repository since your last checkout with your local changes. You may wish to update to a specific revision. In that case, you should add the -r [*revision*] arg to the `svn update` command.

For each updated file you will see what action was taken:

```
A  Added
D  Deleted
U  Updated
C  Conflict
G  Merged
```

If any conflicts arise, you will need to resolve them. See [Handling merge conflicts](#).

Adding files and directories

You can add files and directories with the `svn add` command. The general syntax is:

```
svn add PATH...
```



When you add a directory, Subversion recursively adds all the files within the directory and its subdirectories, unless you specify the `-N` [`--non-recursive`] option. This behaviour is different from CVS.

Copying and moving files and directories

Copying is *the* fundamental operation in Subversion upon which everything else is based. Successive revisions of a file are copies of a file with the content changed.

Unlike CVS, Subversion remembers history when copying files *and* directories.

The general syntax for the copy command is:

```
svn copy SRC DST
```

SRC and DST can each be either a working copy (WC) path or URL:

```
WC  -> WC:   copy and schedule for addition (with history)
WC  -> URL:  immediately commit a copy of WC to URL
URL -> WC:   check out URL into WC, schedule for addition
URL -> URL:  complete server-side copy;  used to branch & tag
```

Moving a file is equally simple:

```
svn move SRC DST
```

Note: this subcommand is equivalent to a 'copy' and 'delete'.

SRC and DST can both be working copy (WC) paths or URLs:

```
WC  -> WC:   move and schedule for addition (with history)
```

URL -> URL: complete server-side rename.

Committing changes

In order to commit a change into the LOFAR software repository, you need a valid Redmine task-id. The commit message **must** start with either Task #<id> or Fixes #<id>, followed by a *meaningful* log message. The latter form will automatically change the state of the task to FIXED.

For the regex savvy

The format of the commit message is verified using the following regex pattern

- `^(fixes|task)\s*#(\d+)/i`

Examples

Valid commit commands

```
svn commit -m "Task #1033: Fixed an integer overflow bug in  
add_and_subtract()"  
svn ci -m "Fixes #675 Final fix for the open socket linger period"
```

Invalid commit commands

```
svn commit -m "Task :1033: Fixed an integer overflow bug in  
add_and_subtract()"  
svn ci -m "Final fix for the open socket linger period"
```

Note: If you omit the -m options, Subversion will bring up an editor so you can type a longer message.



After a commit, *only* the files that were actually changed will have been updated. This means that part of your workspace will still be at the BASE revision, and part will be at the HEAD revision. You should do an `svn update` at the top-level directory to bring the unmodified files up-to-date as well.

Branching and Merging

The concepts of branching, tagging and merging may seem intimidating and confusing at first, but do not let you scare away. You should definitely read [Chapter 4. Branching and Merging](#) of the book [Version Control with Subversion](#)

Branch and tag names must adhere to conventions described on the page [Release Management Procedure](#).

Creating a task or release branch


This is a simple method to create a branch and/or tag. Note, that the branch is created *for the whole LOFAR tree*.

Suppose we want to create a new task branch, which is split off the trunk at svn version number 19123.

- First create a software Task in Redmine (under LOFAR → Tasks → your category) that can be attached to the creation of the release branch (Task 1493 in this example).
- Based on that number, the task will be named (for example) LOFAR-PyBDSM-Task1493. To create the branch directly in the repository give the command:

```
svn copy -r 19123 \  
https://svn.astron.nl/LOFAR/trunk \  
https://svn.astron.nl/LOFAR/branches/LOFAR-PyBDSM-Task1493 \  
-m "Task #1493: Creating task branch LOFAR-PyBDSM-Task1493"
```

Instead of using a particular trunk revision, you can create a task branch from the HEAD, which is the latest version of the trunk, by omitting the `-r <rev.nr.>` in the `svn copy` command.

 If your current working directory is a working copy, you can use the new caret (^) notation as a shorthand for the URL of the repository's root directory. The `svn copy` command then becomes:

```
svn copy -r 19123 \  
^/trunk \  
^/branches/LOFAR-PyBDSM-Task1493 \  
-m "Task #1493: Creating task branch LOFAR-PyBDSM-Task1493"
```

A release branch is created in the same way, but should be named like LOFAR-Release-<major>_<minor>, e.g., LOFAR-Release-1_8. It also requires a task to be created (in subproject LOFAR → Tasks → Rollout).

For the regex-savvy

The release branch name (i.e., the part following `/branches/`) should match with the following regex:

```
^\w+(-\w+)*-Release-\d+_\d+$
```

Creating a (task) tag

This is optional, but some find this useful as a fallback in case of problems. To create a tag LOFAR-Task1718-premerge in your task branch LOFAR-Task1718, use the command:

```
svn copy \  
https://svn.astron.nl/LOFAR/branches/LOFAR-Task1718 \  
https://svn.astron.nl/LOFAR/tags/LOFAR-Task1718-premerge \  
-m "Task #1718: Creating pre-merge tag in repository"
```

Optionally you can add the `-r <revision>` flag if you want the tag to be associated with an earlier status of your code in the task branch. Again, if your current working directory is a working copy, you can use the new caret (^) notation.

For the regex-savvy

The name of the tag should match with the following regex:

```
^(\w+-)+Task\d+-\w+
```



A pre-commit hook will prevent you from accidentally committing to a tag. The only thing you can do to a tag is delete it.

Preparing the reintegration of your TASK! branch with the parent branch

Eventually, you want to reintegrate your development efforts on the task branch with the parent branch from which the task branch has been created (usually the trunk). This is done in three steps:

- Commit all your edits to the task branch
- Merge the current state of the parent branch into your task branch, and commit all changes again.
- Reintegrate the task branch into the parent branch

Commit all edits

Make sure you have committed all changes in your workspace into the SVN repository

Merge the parent branch into the task branch

You have to make sure that any changes made on the parent branch since you created your task branch do not break your code. Therefore, merge the changes on the parent branch with your task branch. See the section [Keeping a Branch in Sync](#) in the book *Version Control with Subversion*.

Before you proceed



Make sure that you have no local (uncommitted) changes in the working copy of your task branch.



Make sure that your working copy contains a *complete* LOFAR tree, which is usually **not** the case when you've let CMake do the (partial) checkout as part of the build. If you don't have a complete tree, you'll be in for a lot of tree conflicts when doing the merge. Issue the following command to update your current working directory to a complete tree:

```
svn update --set-depth infinity
```

If you don't want to clobber your current working copy, you should do a separate checkout of your development branch.

Let's merge

So, let's merge the changes from the trunk (in this example!) into your task branch. Make sure you're at the top-level LOFAR directory in your checked out task branch before issuing the following command:

```
svn merge ^/trunk
```

You may wish to use the `--dry-run` option first to see the result of the merge. This will not actually change any files, but will tell you what you can expect and will detect conflicts.

If there are any merge conflicts, you need to resolve them. To do so, you will have to run the above merge-command first on your workspace of the task branch.

Then resolve all of the conflicts (see [Handling merge conflicts](#) for guidelines).

Finally, check-in all changes into your task branch, e.g.:

```
svn ci -m "Task #1234: Committing changes related to merging the trunk into my task branch"
```

Recompile and test your task branch again. When you're satisfied you are ready to merge the changes made to the task branch back into the trunk. Before doing so, you may wish to tag the task branch as done.

Reintegrate your changes with the parent branch

Before merging your changes back to the parent branch, make sure you've committed all your changes (see above)! To reintegrate with the parent branch, you need to have a local working copy of it. This can be done either by:

- Switching your working copy to the parent branch, e.g.:

```
svn switch ^/trunk
```



Make sure you have a *clean* working copy after the switch, which contains the complete code tree. Use the option `--set-depth infinity` if you had a sparse working copy before the switch.

- Checking out a working copy in a new directory

```
cd LOFAR_trunk
svn co https://svn.astron.nl/LOFAR/trunk LOFAR
```

Now you're ready to reintegrate your changes with the parent branch, e.g.:

```
svn merge --reintegrate ^/branches/LOFAR-Task1718-creation
```

Note the use of the `--reintegrate` option. This is *required* when merging back into the parent branch. Please refer to the section [Reintegrating a Branch](#) in the book *Version Control with Subversion* for more details.

If there are no conflicts, or all conflicts are resolved, do not forget to check in the results:

```
svn ci -m "Task #1234: Reintegrated task branch LOFAR-Task1718-creation
(short description of what you did) into the trunk"
```



Once a `--reintegrate` merge is done, the task branch is *no longer usable* for further work. If you want to continue to work on this branch, you should first delete it from the SVN repository (see below), and then recreate it again from the parent branch.

Now that you've reintegrated your task branch with the parent branch, it's time to delete the obsoleted task branch:

```
svn delete ^/branches/LOFAR-Task1718-creation -m "Task #1234: branch LOFAR-
Task1718-creation no longer needed"
```

Merging changes in a **RELEASE!** branch with the trunk

After a change has been made to a release branch, the change must also be ported to the trunk so the change is not lost when the next release branch is created. This must be done immediately after the change to the release branch has been committed, for the following reasons:

- You are still aware of the changes made so any conflicts can be solved more easily
- The chances that your changes are the only ones made to the release branch are much larger

The best thing to do is to merge the whole release branch into the trunk. To do so:

1. Get a clean working copy of the trunk.
2. Merge the release branch into the trunk (do *not* reintegrate!)
3. Resolve conflicts and run the (unit) tests
4. Commit your changes

Get a clean working copy of the trunk

```
svn co https://svn.astron.nl/LOFAR/trunk <local dir>
```

Or switch an existing working copy (e.g., of your task branch) to the trunk

```
svn switch ^/trunk
```

In this case, make sure your working copy is clean, i.e., `svn stat` should not show any modified or unversioned files.

Merge the release branch into the trunk

Go into `<local dir>` and execute:

```
svn merge ^/branches/<release branch name>
```

Do *not* reintegrate!

Resolve conflicts

Textual conflicts must be resolved. Tree conflicts are usually related to *mergeinfo*, which can be accepted as theirs fully. [See below](#) for more info on resolving merge conflicts.

If you note that there are more changes in the release branch than your own, please find out who committed those changes and resolve the conflicts together. If that is not possible, you can add a revision range to the `svn merge` command which contains only your changes (though this is the least preferred option as it forwards the problem to the person who makes the next change to the release branch and needs to merge these into the trunk..!).

Commit your changes

After all conflicts are resolved and if all (unit) tests pass, changes must be committed to the trunk:

```
svn ci -m "Task #xxxx: Merged changes in release branch <release branch name> to the trunk"
```

Handling merge conflicts

For guidelines on how to merge changes made in a branch to the trunk, see [the previous section](#).

A merge conflict can occur when two or more people make different changes to the same file. When Subversion encounters a merge conflict (e.g. during an update), it will place specific markers in the original source file.

- The lines between the markers: <<<<<< .mine and ===== contain our local changes;
- The lines between ===== and >>>>>> .r<nn> contain the changes in the repository since our last checkout or update.

Furthermore, Subversion will make copies of the unmergeable files; for example:

- MACScheduler.cc.merge-left.r16851 → Older revision
- MACScheduler.cc.merge-right.r17270 → Newer revision
- MACScheduler.cc.working → Copy of the version that was present in your directory

Resolving the conflict

To resolve this conflict you have three options: scrap your changes, keep your changes, or use parts of both

Scrap your changes

If you decided to scrap your changes and use the version in the repository, then all you have to do is revert your changes

```
svn revert <file>
```

or alternatively

```
cp <file>.r<nn> <file>
```

Keep your changes

If you wish to keep your changes and lose those in the repository, then you should copy the .mine backup file that Subversion created to the original file

```
cp <file>.mine <file>
```

Use parts of both versions

Finally, if you decide to use parts of both revisions, you'll have to do some manual editing. Fire up your favorite editor and make your changes. Do not forget to remove the conflict markers!

Marking the conflict as resolved

Once you've resolved the conflict, you should inform Subversion, otherwise you will not be able to commit the file.

```
svn resolved <file>
```

More information

More information on handling (merge) conflicts can be found in the sections [Resolve Any Conflicts](#) and [More on Merge Conflicts](#) of the book *Version Control with Subversion*. See also the following links for solving very specific issues when merging:

<http://stackoverflow.com/questions/2472249/missing-ranges-error-message-when-reintegrating-a-branch-into-trunk-in-subversion>

Removing a branch

Branches can be removed from the repository using the command:

```
svn delete <URL of branch>
```

References

- [Release Management Procedure](#)
- [LOFAR Subversion Frequently Asked Questions](#)
- [Subversion for CVS users](#)
- [How we migrated from CVS to Subversion](#)
- The [Subversion homepage](#)
- The online book [Version Control with Subversion](#)
- openCollabNet's [Subversion Knowledge Base](#)

From:
<https://www.astron.nl/lofarwiki/> - **LOFAR Wiki**

Permanent link:
https://www.astron.nl/lofarwiki/doku.php?id=public:user_software:documentation:lofarsvn&rev=1486563936

Last update: **2017-02-08 14:25**

