



# PALM Revision Control

---

last update: 20. Feb 2018



Institute of Meteorology and Climatology, Leibniz Universität Hannover

## └ Preface

- This presentation explains how to work with svn branches, and how to possibly use git for local revision control.
- These are just hints and recommendations. If you are an experienced svn user, feel free to follow your own best practice.
- Only the minimum set of svn commands is introduced here. For more detailed information, please check the official svn manual (<http://svnbook.red-bean.com/de/1.7/index.html>)
- Please contact Farah Kanani-Sühring ([kanani@muk.uni-hannover.de](mailto:kanani@muk.uni-hannover.de)), if any of the steps remain unclear.

## 1 – Creating an svn branch

<feature\_name> = name of svn branch

### ▪ Repository branch (only for admins)

- @palm server: permissions to be set in /palmdata/conf/svn\_authz file
- `svn copy https://palm.muk.uni-hannover.de/svn/palm/trunk https://palm.muk.uni-hannover.de/svn/palm/branches/<feature_name> -m "Branch for <feature_name> created."`
- (rename: `svn move <path to old branch> <path to new branch> -m „...“`)

### ▪ Local branch copy (for developers)

- Create local working directory where the local copy of branch <feature\_name> is placed, we recommend:

```
mkdir -p ~/palm/branches
```

The directory `branches` is on the same level as the directory `current_version`, where the official PALM releases/revisions (`trunk`) should be located

- You can browse your branch under <https://palm.muk.uni-hannover.de/trac/browser/palm/branches>

## 2a – Checkout of local branch copy

- `cd $HOME/palm/branches/`
- `svn checkout --username <your PALM username>  
https://palm.muk.uni-hannover.de/svn/palm/branches/<feature_name>`

## 2b – Update of local branch copy

If the local branch copy already exists, and you would like to update it with a newer svn revision of this branch:

- `cd $HOME/palm/branches/`
- `svn update <feature_name>`

You can also *update/downdate to* or *checkout* a specific revision:

- `svn update -r<revision number> <feature_name>`
- `svn checkout -r<revision number> --username ...`

## 3 – Code development in local ...

- ...svn branch copy

Versioning happens when committing to the svn repository branch. With every commit, the PALM revision number is incremented.

- ...git branch copy

To avoid too numerous commits to the svn repository branch, local versioning could be done using **git** (→ follow steps on next slide)

**NOTE:** As you know, in order to compile and run PALM, the `PATH` and `PALM_BIN` variables need to be set in your shell environment (e.g. `.profile` or `.bashrc`) as in this example:

```
# PALM SCRIPTS
export PATH=$HOME/palm/current_version/trunk/SCRIPTS:$PATH
export PALM_BIN=$HOME/palm/current_version/trunk/SCRIPTS
```

These [paths need to be adjusted](#) when you run PALM out of your `~/palm/branches/<feature_name>` directory! You might have to newly login to your system in order to activate these changes. You can check the paths by typing `echo $PATH` or `echo $PALM_BIN` to your terminal.

An alternative to this approach is to [use symbolic links](#) to link `~/trunk` to `~/branches/<feature_name>`. This way you don't have to keep changing the `PATH/PALM_BIN` variables, and you run PALM out of your `current_version` directory, with your `SCRIPTS/SOURCE` files from `~/palm/branches/<feature_name>`

## 3a – How to use git for local code versioning

(These are only brief hints, please refer to the git manual for more detailed help)

- Create local repository

```
mkdir -p ~/git_palm/<feature_name>
```
- Rebase from svn feature repository
- Development under git (refer to manual)
- Manual copying of git SOURCE to svn SOURCE (<feature\_name>) necessary
  - Why? → Due to PALM-politics of date-stamping of revision comments (git cannot do that)
  - Note: svn and git copy should have the same svn revision number r####
- Continue with svn procedure (see next slide)

## 4a – Merge repository branch into local branch copy

(This might become necessary if more than one developer simultaneously work on the same PALM-4U feature branch)

- (1) `cd $HOME/palm/branches/<feature_name>`
- (2) `svn update` (takes care of merge here)
- (3)

During update/merge, svn will print an alert in case the same line/block of code has been modified by the different developers. As an example:

```
Conflict discovered in file 'palm.f90'.
```

```
Select: (p) postpone, (df) show diff, (e) edit file, (m) merge,  
        (mc) my side of conflict, (tc) their side of conflict,  
        (s) show all options:
```

We recommend to **select (p)ostpone** for all emerging conflicts, and after that, manually check/merge the contents of these files. **(Details in Chapter 4c)**

- (4) `svn resolved any_file.f90`
- (5) `svn commit` **(Details in Chapter 5)**

**Note:** It might be wise to put a copy of your modified files somewhere else before you do the merge.

## 4b – Merge PALM main repository revision (trunk) into local branch copy

- (1) `cd $HOME/palm/branches/<feature_name>`
- (2) `svn update` (to obtain HEAD (overall latest) repository revision )
- (3) `svn merge -r####:HEAD`  
`https://palm.muk.uni-hannover.de/svn/palm/trunk`  
(####: branch revision at last merge with trunk)

(4)

During update/merge, svn will print an alert in case the same line of code has been modified by the different developers. As an example:

```
Conflict discovered in file 'palm.f90'.
```

```
Select: (p) postpone, (df) show diff, (e) edit file, (m) merge,  
        (mc) my side of conflict, (tc) their side of conflict,  
        (s) show all options:
```

We recommend to **select (p)ostpone** for all possible conflicts, and after that, manually check/merge the contents of these files. **(Details in Chapter 4c)**

- (5) `svn resolved any_file.f90`
- (6) `svn commit` **(Details in Chapter 5)**

**Note:** It might be wise to put a copy of your modified files somewhere else before you do the merge.

## 4c – Resolving conflicts in a merge

You will find following set of files in case of a conflict, e.g.

- `average_3d_data.f90.merge-left.r2046` (your code was used at conflicted line in merged file)
- `average_3d_data.f90.merge-right.r2735` (their code was used at conflicted line in merged file)
- `average_3d_data.f90.working` (your unmerged file version)
- `average_3d_data.f90` (your + their code at conflicted line in merged file)

To resolve the conflict, we recommend to take file `average_3d_data.f90`, and manually work through the conflicted lines, which look something like this:

your code	USE control_parameters,	&
	<<<<<<< .working	
their code	ONLY: air_chemistry, average_count_3d, doav, doav_n, land_surface, urban_surface,	&
	varnamelength	
	=====	
	ONLY: air_chemistry, average_count_3d, doav, doav_n, land_surface,	&
	urban_surface, varnamelength	
	>>>>>>> .merge-right.r2735	

**Carefully check** your & their code lines, and put them together appropriately!

Remove these lines, as well as any doubled code!

You could also use `kdiff3` of files „left“ and „right“ to visualize the changes at conflicted lines.

**The final merged content must be in file `average_3d_data.f90` (without suffix)**

**Now continue with step `svn resolved` in Chapter 4a or 4b!**

## 5 – Commit changes to repository branch

(1)

Add a brief summary of your changes into the header of the file you modified under „Current revisions:“. Add <your PALM username>, and repeat this for every commit of this specific file, e.g.:

```
! Current revisions:
```

```
! - - - - -
```

```
! Change of variable name pt to vpt (<your PALM username>)
```

```
! Bugfix in shortwave radiation calculation (<your PALM username>)
```

(2) `cd ~/palm/branches/<feature_name>/SOURCE`

(3) `svn status`

(to see which files were actually **Modified**, **ATTENTION: all listed files will be committed in step 4**)

(4) `svn commit -m 'Branch <feature_name>: short comment on changes'  
<file1.f90> <file2.f90>`

(5)

Now you should be able to see your committed files, if you browse your branch under <https://palm.muk.uni-hannover.de/trac/browser/palm/branches>

While committing to your branch, please make sure that the revision comments remain under „Current revisions“. Do not use the script `document_changes`, nor manually copy the revision comment to the „Former revisions“ section. This makes merging to the `palm4u` branch or to the trunk more difficult.

## 5a – Commit changes to repository branch → How to add new files (I)

(For how to structure new SOURCE files, please take a look at the template\_newmodule\_mod\_v3.f90 in the Downloads section of the MOSAIK page)

In order to use the “keyword substitution” feature of subversion, a newly created file must contain the Id-keyword

```
! Former revisions:  
! -----  
! $Id$
```

The new file has to be earmarked in order to be added to the svn repository with the next commit. This marking is done by

```
svn add newfile.f90
```

Substitution of the „Id“-keyword has to be activated with svn-command

```
svn propset svn:keywords "Id" newfile.f90
```

## └ 5a – Commit changes to repository branch → How to add new files (II)

After the commit (see Chapter 5), subversion will automatically substitute the “Id” string with the current timestamp:

```
! Former revisions:  
! -----  
! $Id: newfile.f90 1683 2015-10-07 23:57:51Z raasch $
```

## 6 – Some additional commands

(For details about the commands, see <http://svnbook.red-bean.com/de/1.7/index.html>)

- Deleting files or directories from the svn repository  
`svn delete file/directory`  
(Perform command in local branch copy to mark files/directories for deletion. During the commit, the marked files/directories will be deleted in the svn branch)
- Checking, which files have been modified in your local branch copy with respect to the checked-out branch revision  
`svn status <feature_name>`
- Printing revision information about the local branch copy  
`svn info <feature_name>`