Schweizerische Eidgenossenschaft
Confédération suisse
Confederazione Svizzera
Confederaziun svizra

Eidgenössisches Departement des Innern EDI
Bundesamt für Meteorologie und Klimatologie MeteoSchweiz

PYRAD

User Manual

	In progress	In validation	Approved
Document status	☒	☐	☐

Version 0.1

Document History

Responsible People

Creation/Edition	Jordi Figueras i Ventura (fvj)
Revision	
Approval	
Further information	

Version Control

Version	**Edited by**	**Date**	**Activity**
0.1	fvj	28.06.2016	Creation

Contents

Figures

Tables

No table of figures entries found.

Chapter 1 Installation

1.1 Conda installation and Pyrad environment creation

Open a shell and get the conda installation file from [1]:

wget https://repo.continuum.io/archive/Anaconda3-4.3.1-Linux-x86_64.sh

Install conda by executing:

bash Anaconda3-4.3.1-Linux-x86_64.sh

and following the instructions.

Create a pyrad environment by typing:

conda create --n pyrad python=3.5

Activate the python environment:

source activate pyrad

Install all the required packages (see section 1.2).

Create the file with the environment variables:

cd [conda_path]/envs/pyrad

mkdir -p ./etc/conda/activate.d

mkdir -p ./etc/conda/deactivate.d

touch ./etc/conda/activate.d/env_vars.sh

touch ./etc/conda/deactivate.d/env_vars.sh

Edit the two files :

File /activate.d/env_vars.sh :

```
#!/usr/bin/sh

# path to py-art configuration file
export PYART_CONFIG=$HOME/pyrad/config/pyart/mch_config.py

# RSL library path
export RSL_PATH="/home/cirrus/anaconda3/envs/pyrad"

# path to library that reads METRANET data
export METRANETLIB_PATH="/home/cirrus/idl/lib/radlib4/"

# gdal library for wradlib5
export GDAL_DATA="/home/cirrus/anaconda3/envs/pyrad/share/gdal"
```

File /deactivate.d/env_vars.sh:

```
#!/usr/bin/sh

unset PYART_CONFIG
unset RSL_PATH
unset METRANETLIB_PATH
unset GDAL_DATA
```

1.2 Py-ART installation instructions

1.2.1 Pre-installation requisites

Note 1: If you are using the zueub222 MeteoSwiss server or CSCS the package pre-installation has already been performed for you. Simply activate the right conda environment (root at zueub222 and pyrad at CSCS), i.e.:

source activate pyrad

Note 2: The pathes in the .bashrc file here are those for zueub222. If you are working in another server modify them accordingly

A version of Anaconda supporting Python 3.5 or higher should be installed in the server. It can be found in [1]. Do not forget to add the path to Anaconda in your .bashrc file. In the case of zueub222 is:

```
export PATH=/opt/anaconda3/bin/:$PATH
```

The following default packages in the Anaconda installation are necessary to run Py-ART: NumPy, SciPy and matplotlib. Before installing additional packages, depending on the configuration of your server, you may need to switch off ssl verification:

```
conda config --set ssl_verify false
```

On addition, the package netCDF4 has to be installed using the following command:

```
conda install netcdf4
```

There are a number of optional dependencies available from conda that can be installed as well: h5py (to read HDF5 files), basemap (to plot grids on geographic maps), nose (to run Py-ART unit tests), gdal (to output GeoTIFFS from grid objects) and some sort of linear programming solver to use the LP phase processing method such as CyLP. They can all be installed simply with the command:

```
conda install basemap nose h5py gdal
```

In addition the TRMM Radar Software Library (RSL) (See for the webpage [2]) can be installed to read radar files in particular formats. The installation is performed from the jjhelmus channel:

```
conda install -c https://conda.binstar.org/jjhelmus trmm_rsl
```

WARNING: Be aware that there are other versions of the library in other channels but, if installed with conda, only this channel should be used because otherwise the library is not working properly due to issues with the pathes.

The location of the library (where the lib and include directories are) should be specified with the following command (typically on your .bashrc file):

```
export RSL_PATH=/opt/anaconda3/
```

Finally, wradlib (see [3]) is used to compute the texture of a differential phase field. It can be installed from conda adding the conda-forge channel:

```
conda config --add channels conda-forge
```

And then installing the package:

```
conda install wradlib
```

Finally, to have the capacity to read rainbow5 files you have to add the xmltodict library:

```
pip install --trusted-host pypi.python.org xmltodict
```

The location of the GDAL data has to be specified by writing in your .bashrc file the following command:

```
export GDAL_DATA=/opt/anaconda3/share/gdal
```

WARNING: Additional packages or different versions of the packages may be necessary to run wradlib property. You can check if the installation has been performed properly by typing

```
python
```

and then:

```
import wradlib
```

If there is any error you should download the missing library. A good practice may be to update all the installed packages in anaconda by typing:

```
conda update --all
```

In addition to the standard Py-ART packages at MeteoSwiss we have created specific libraries to read rad4alp and DX50 radar data. For the rad4alp data, make sure that you have access to the library srn_idl_py_lib.[machine].so and add the path to your .bashrc:

```
export METRANETLIB_PATH=/proj/lom/idl/lib/radlib4/
```

For the rainbow data files (i.e. the file type used by the DX50) there exists a reader in the wradlib library but we have also created an alternative reader using a c-based library which is located in the libRainbow directory of the pyrad repository. To use that one add the path to the Python pathes in your .bashrc. For example:

```
export PYTHONPATH=[Pyrad_path]/src/libRainbow/:$PYTHONPATH
```

1.2.2 Py-ART installation

Note 1: Make sure to have the latest version of the pyrad repository in your local server.

Note 2: In zueub222 and cscs activate the pyrad environment before installation

Py-ART repository can be found on [4]. A compiled version is available from the conda repository:

```
conda install -c https://conda.anaconda.org/jjhelmus pyart
```

We have our own (modified) version of py-ART in the pyrad repository which can be cloned using the procedure described in section 3.3.1.

To install Py-ART in your personal repository enter into the directory pyart-master and simply type:

```
python setup.py install --user
```

Optionally, if you have the rights for this you can install it for all users by typing:

```
python setup.py build
```

```
sudo python setup.py install
```

To check whether the library dependences have been installed properly type:

```
python -c "import pyart; pyart._debug_info()"
```

Important: Type the aforementioned command outside the pyart directory

Py-ART has a default config file called default_config.py and located in folder pyart. If you would like to work with a different config file you have to specify the location in the variable PYART_CONFIG in your .bashrc file. For example:

```
export PYART_CONFIG= [Pyrad_path]/config/pyart/mch_config.py
```

Note: In zueub222 or cscs the variable is already defined in the conda environment

The Pyrad library has its own config file in the aforementioned path.

1.2.3 Py-ART extensions

Several extensions build over Py-ART are available. In the following we will show how to install the ones available in the pyrad repository.

1.2.3.1 ARTView

ARTView is an interactive radar viewing browser. The source code can be found in [5]. The simplest way to install it is using conda:

```
conda install -c jjhelmus artview
```

1.2.3.2 DualPol

DualPol is a package that facilitates dual-polarization data processing. Its source code can be found in [6]. Apart from Py-ART it is built on the libraries CSU_RadarTools and SkewT, which have to be installed first.

SkewT can be found in [7]. It provides a set of tool for plotting and analysis of atmospheric data. To install it simply download the source code, go to the main directory and type:

```
python setup.py install
```

CSU_RadarTools can be found in [8]. It provides a set of tools to process polarimetric radar data developed by the Colorado State University. To install it simply download the source code, go to the main directory and type:

```
python setup.py install
```

Finally you can install DualPol by downloading the source code, going to the main directory and typing:

```
python setup.py install
```

1.2.3.3 PyTDA

PyTDA is a package that provides functions to estimate turbulence from Doppler radar data. Its source code can be found in [9]. To install it simply download the source code, go to the main directory and type:

```
python setup.py install
```

1.2.3.4 SingleDop

SingleDop is a package that retrieves two-dimensional low-level winds from Doppler radar data. It can be found in [10]. It requires PyTDA to be installed in order to work. To install it simply download the source code, go to the main directory and type:

```
python setup.py install
```

1.2.3.5 PyBlock

PyBlock estimates partial beam blockage using methodologies base on the self-consistency of polarimetric radar variables in rain. It can be found in [11]. It requires DualPol to be installed in order to work properly. To install it simply download the source code, go to the main directory and type:

```
python setup.py install
```

1.3 Pyrad_proc installation instructions

Note: In zueub222 and cscs activate the pyrad environment before installation

For certain applications the pandas package has to be installed:

conda install pandas

Pyrad_proc is the container for the MeteoSwiss radar processing framework. The core radar processing functions are based on Py-ART. Therefore Py-ART should be correctly installed before running Pyrad_proc.

To install pyrad_proc, simply go to the main directory and type:

```
python setup.py install --user
```

This setup command will build and install your pyrad code. The build output is stored in the directory “build” in your pyrad_proc directory. The installation process with the option “- -user” will store the output in your home local directory (e.g. $HOME/.local/lib/python3.5/site-packages/pyrad/).

The previous procedure has the disadvantage that every time you change a single line of your code, you have to recompile and reinstall your code. For development purpose there exist a mode where the active code is directly in your working directory. Thus, your changes are active immediately without recompiling and reinstalling. To activate the development mode:

```
python setup.py develop --user
```

Cleaning up the code:
To clean up the installed code go to the installation directory (e.g. $HOME/.local/lib/python3.5/site-packages/) and remove the whole “pyrad” directory and all “mch_pyrad-*” files.

To clean up the “build” directory, run:

```
python setup.py clean --all
```

Chapter 2 Using Pyrad

2.1 Compilation

To compile Pyrad one has always to remember to first compile Py-ART and then Pyrad. The first installation has to be performed manually. First you have to activate the conda pyrad environment:

source activate pyrad

Then you place yourself into the pyart directory (pyrad/src/pyart) and compile pyart with the command:

python setup.py install --user

Then you place yourself into the pyrad directory (pyrad/src/pyrad_proc) and compile pyrad with the command:

python setup.py install --user

After this very first compilation, the subsequent compilations can be performed using the scripts aimed at it placed in pyrad/src, which have the advantage to delete the objects created by previous compilations. To compile both Py-ART and Pyrad use the script make_all.sh. To compile either Py-ART or Pyrad use the scripts make_pyart.sh or make_pyrad.sh respectively.

If you only modified code in pyrad_proc you do not need to recompile pyart for the changes to take effect but if you modify code in Py-ART you have to compile both Py-ART and Pyrad to make effective the changes.

2.2 Configuration files

Pyrad uses 3 different configuration files which are typically stored in the folder:

pyrad/config/processing/

The first file specifies the input data, output data and configuration files packages, the second specifies radar related parameters (radar name, scan name and frequency, etc.) and the general configuration of the various image output, the last file specifies the datasets and products to be produced.

The easiest way to start is to copy one of the available config files and modify it according to your needs.

2.3 Running the programs

To run the programs first you need to activate the conda pyrad environment

source activate pyrad

Then go to directory:

pyrad/src/pyrad_proc/scripts/

and type:

python [name_of_the_program] [variables]

At the moment there are two main programs:

main_process_data.py will process (and optionally post-process) data from a starting point in time to and ending point in time.

main_process_data.py will process (and optionally post-process) data over several days starting the processing at a given starting and ending time (default 00:00:00 for start and 23:59:59 for the end).

Chapter 3 Developing pyrad

3.1 The pyrad git architecture

A schematic of the Pyrad git architecture can be seen in Fig. 1. The Pyrad project contains three main directories: config stores the configuration files, doc contains relevant documentation about the project and finally src contains all the source code related to the project. Within the src directory there is the main program, which is contained inside the pyrad_proc directory and a set of auxiliary software tools and example programs. The main program controls the workflow of the processing framework and the datasets and products generated. The actual signal processing is intended to be performed by the auxiliary software and in particular by Py-ART. Since MeteoSwiss wants to contribute to the development of Py-ART it has been set as a submodule of Pyrad.

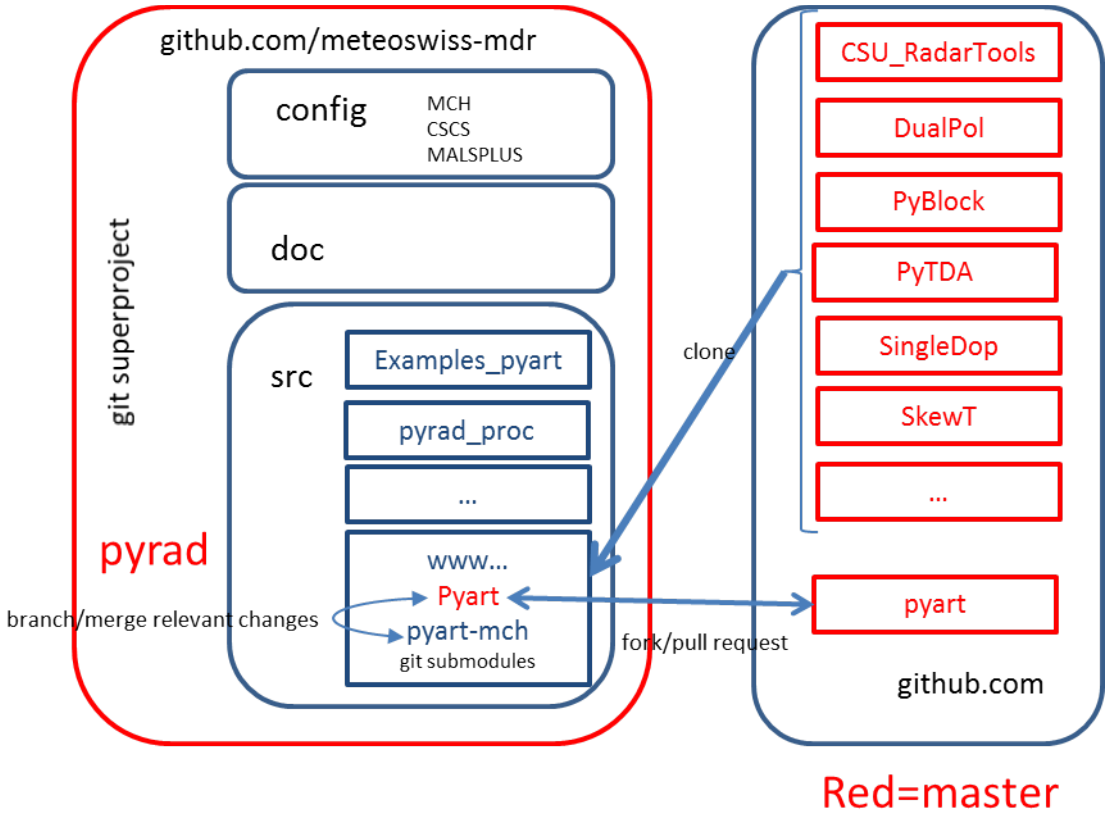

Fig. 1 The Pyrad superproject architecture

The Pyrad project is stored in a repository in github [12]. The MeteoSwiss Py-ART submodule was forked from the Py-ART repository [4] and is placed in the github repository [13]. It has two branches: The master branch is used exclusively to introduce code developed by MeteoSwiss into the main Py-ART project. This branch is intended for use only by the PI of the Pyrad project. The branch named pyart-mch is the one used to develop new code. People wishing to contribute to the MeteoSwiss Py-ART project should work with this second branch.

3.2 Code style

Pyrad and its submodels follow the PEP8 standard [14]. To make sure that your code formally complies with the standard make use of the pycodestyle tool [15]. You can install it from conda simply typing:

conda install –c conda-forge pycodestyle=2.0.0

For users of zueub222 the package has already been installed. Detailed instructions on how to use the tool can be found in [16]. The simplest use is simply to type:

pycodestyle [your_file.py]

A list of errors and their location will appear.

3.3 Installing and developing the pyrad git superproject by internal MeteoSwiss collaborators

3.3.1 Obtaining Pyrad and its submodules

Internal developers can work directly with the pyrad and Py-ART MeteoSwiss repositories. To get a copy of the Pyrad superproject simply place yourself in the desired working directory and type:

git clone --recursive https://github.com/meteoswiss-mdr/pyrad.git

The recursive keyword fetches automatically all the submodules depending on the main superproject. You should get the MeteoSwiss working branch of Py-ART so place yourself in the pyart directory (pyrad/src/pyart/) and type:

git checkout pyart-mch

3.3.2 Developing Pyrad and its submodules

The regular git commands summarized in Fig. 2 apply. However one has to remember that the Pyrad project contains submodules and those have to be pushed first to the submodule repository before commiting the super-project.

To push changes in the submodule go to the main folder of the submodule and do the following:

1. Check the status of the module:

   git status

2. Check to which remote you are connected:

   git remote -v

3. Check in which branch are you working in (for regular Py-ART developers should be pyart-mch)

   git branch

4. If the branch is not the desired one change it:

   git checkout pyart-mch

5. Add or remove the files you want to commit with the regular commands git add git rm.

6. Commit your changes:

   git commit --a --m “explanation of my changes”

7. Pull the remote and deal with possible conflicts. If necessary commit again:

   git pull

8. Once satisfied, push the changes to the remote:

   git push

   You will be asked to input your user name and password.

Once this is done, you can push the changes in the super-project by going to the main folder of the super-project and repeating steps 4 to 8. Do not forget to add the submodule before you commit.

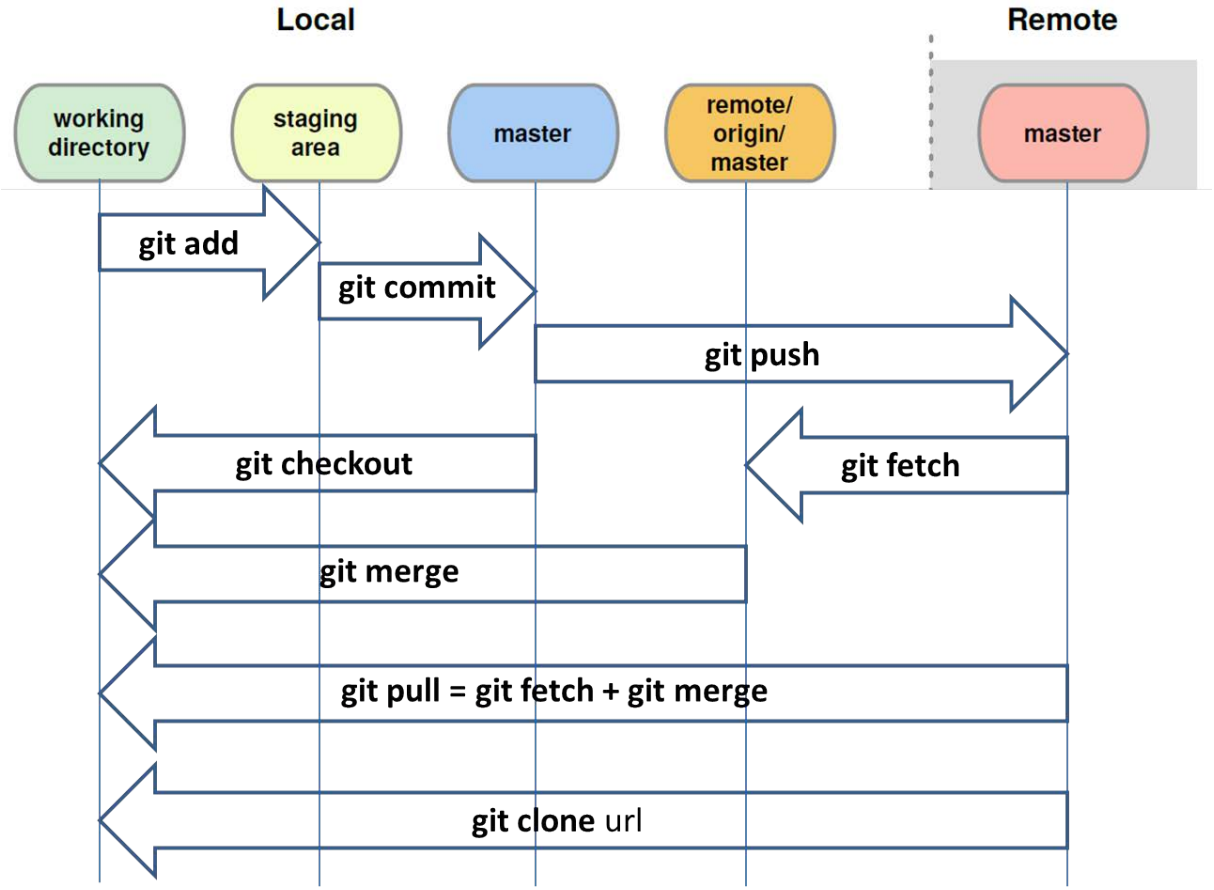

Fig. 2 Git flow diagram

3.4 Installing and developing the Pyrad git superproject by external MeteoSwiss partners

If you are not an internal MeteoSwiss collaborator you do not have direct write access to the Pyrad superproject and its submodules. However you can still propose changes and additions to the code that will be evaluated and eventually accepted by the PI.

3.4.1 Obtaining the Pyrad superproject and its submodules

1. Sign in into Github (create a user account if you do not have it).

2. Go to the web page of the Pyrad super-project [12] and the Py-ART submodule [13] and fork them. IS IT NECESSARY TO CLONE Py-ART submodule??

3. Follow the instructions in section 3.3.1 but with your own username instead of meteoswiss-mdr.

3.4.2 Developing Pyrad and its submodules

To develop your local version of Pyrad and its submodules the instructions on section 3.3.2 apply. To update your forked version with the changes from the MeteoSwiss repository or contribute to the Meteoswiss repository follow the procedures described in sections 3.5.2 and 3.5.3 respectively. **It is strongly recommended that you create a branch specific for the changes you would like to submit to the Pyrad superproject.**

3.5 Installing and developing Pyrad by the principal investigator (PI)

WARNING: The underlying philosophy is that there should be a single development leader in charge of the interaction between pyrad and its public submodules so regular developers should not be concerned by this section.

3.5.1 Installing a git submodule

The pyrad superproject contains a number of Open Source public libraries. In some of them, namely Py-ART, we wish to have an active collaboration and therefore we should be able to interact with the project using the git commands. This requires several steps. In the following we will describe them taking Py-ART as an example. For other products the steps would be analogous:

1. fork the project in the github.com repository (simply register as user and click fork in the main page in https://github.com/ARM-DOE/pyart). A copy of the master program will be created in your personal github space, i.e. https://github.com/meteoswiss-mdr/pyart)

2. In your local copy of Pyrad, from the directory where you want to keep the submodule (i.e. pyrad/src/) add the submodule **from the forked version** of the library:

   git submodule add https://github.com/meteoswiss-mdr/pyart.git src/pyart

A file .gitmodules will be created in the main directory of the Pyrad repository. This is a good point where to commit the submodule to the repository.

3. Create a new local branch of the forked version. This is going to be the branch where local developments will be made:

git checkout –b pyart-mch

4. Add the information of your working branch into your git config file:

git config --file=.gitmodules submodule.src/pyart.branch pyart-mch

This is another good point to commit to the repository.

3.5.2 Updating the local submodule working branch with changes in the master public library

1. Place yourself in the superproject directory and change the information on url and branch contained in the .gitmodules file. Do not forget to synchronize everything:

git config --file=.gitmodules submodule.src/pyart.url https://github.com/ARM-DOE/pyart.git

git config --file=.gitmodules submodule.src/pyart.branch master

git submodule sync

Where [pyart] is the name of the submodule and the url is the url of the master public library.

2. Place yourself in the submodule directory, check that you are using the master branch and change branch otherwise and pull to update the local branch with the changes in the public library:

git branch

git checkout master

git pull

3. Syncronize the changes in the submodule with the superproject:

git submodule sync

4. Now your local master branch is updated with the additions of the main public library. You should commit these changes to your forked version in github. First place yourself in the main directory of the superproject and change back your url in your .gitmodules file:

git config --file=.gitmodules submodule.src/pyart.url https://github.com/meteoswiss-mdr/pyart.git

5. As usual you have to sync the submodule:

git submodule sync

6. Finally you should push the changes to your fork by placing yourself in your submodule project and:

git push

7. Now change the working branch back to the regular pyart-mch:

git checkout pyart-mch

8. And place yourself in the superproject main to change the branch in the .gitmodules file back to your working branch:

   git config --file=.gitmodules submodule.src/pyart.branch pyart-mch

9. Now to update your working branch simply place yourself in the submodule directory and merge the master with it:

   git merge master

10. Solve any possible conflicts that arise and commit and push the result.

11. Place yourself in the superproject directory and commit all the changes

3.5.3 Transferring changes from the local submodule working branch to the master public library

Ideally this should be the responsibility of a single person.

1. Place yourself in the local submodule directory and make sure you are using the master branch:

   git branch

If you are not in the master branch change it:

git checkout master

2. Create a new branch where you will place the changes you desire to make public. Try to use a branch name that relates to the new development, i.e.:

   git checkout –b pyart-vulpiani-fix

3. Push the newly created local branch so that it is available remotely:

   git push --set-upstream origin pyart-vulpiani-fix

4. Make all the changes you desire to make public in this local branch.

   a. If you want to add a completely new file to the master from the pyart-mch branch you can use git checkout and the path to the new file, for example:

      git checkout pyart-mch pyart/correct/noise.py

   b. If the file already exists and you want to selectively apply some changes use:

      git checkout --patch pyart-mch pyart/correct/noise.py

   It will allow to interactively go through the differences between the files at each branch and apply the changes you desire.

5. Commit all the changes you have performed and push them to your forked public repository

6. In your forked public repository (https://github.com/meteoswiss-mdr/pyart) select the branch you created for your development and click on “New pull request”. Select ARM-DOE:master as your target and make sure that meteoswiss-mdr: pyart-vulpiani-fix is the origin. Once a pull request is open all new commits will be directly visible so there is no need to open a pull request for each new commit.

If you want to keep working in a new development while waiting for the outcome of the pull request you can check out to the regular pyart branch but do not forget to switch branches if changes the pulled code are requested. Once the pull request has been accepted you can delete the temporary branch you created. To delete the remote branch:

git push origin --delete pyart-vulpiani-fix

To delete the local branch:

git branch -d pyart-vulpiani-fix

3.6 Manage a pull request

It is recommended to always create a new branch to test the changes locally:

git checkout –b [name_of_test_branch] [name_of_pull_request]

git pull https://github.com/[forker]/pyrad.git master [name_of_pull_request]

Check all the new functionalities of the pull request. If you make any changes commit them locally.

When it is ready merge it to the MeteoSwiss master:

git checkout master

git merge --no-ff name_of_test_branch

git push origin master

After a period remove the test branch.

3.7 Automatic Generation of Documentation

To automatically generate documentation you have first to make sure the package Sphinx ([17] available in conda) is installed. It is also recommended you install the Sphinx extension numpydoc. A good tutorial on how to create documentation with Sphinx can be found in [18].

Create the directory where you want to keep the documentation. Place yourself inside this directory and execute the program:

sphinx-quickstart

Answer all the questions. Once the program has been executed it will have created a source directory with a conf.py and index.rst files and a MakeFile. Inside the conf.py add extension ‘numpydoc’ in extensions lists and import the package you want to comment. For example:

```
import os

import sys

sys.path.insert(0, os.path.abspath('../../../../src/pyrad_proc/pyrad/'))

import pyrad
```

The sys.path.insert is necessary so that sphinx knows where to look for your package. Have a look at the contents of the file and modify it at your convenience.

Create a .rst file for each module you want to include in the documentation and name them (without the extension) in the allocated space in the index.rst file. If you want to document only the high level functions available to the user the module.rst file should look like that:

```
:mod: `pyrad.flow`

==================


.. automodule:: pyrad.flow

  :members:

  :undoc-members:

  :private-members:

  :special-members:

  :inherited-members:

  :show-inheritance:
```

If you want to document all the functions in the package you should specify the path to all the files, i.e.:

```
:mod: `pyrad.io`

================


.. automodule:: pyrad.io.read_data_radar

  :members:

  :undoc-members:

  :private-members:
```

```
   :special-members:
   :inherited-members:
   :show-inheritance:
.. automodule:: pyrad.io.read_data_other
   :members:
   :undoc-members:
   :private-members:
   :special-members:
   :inherited-members:
   :show-inheritance:
.. automodule:: pyrad.io.write_data
   :members:
   :undoc-members:
   :private-members:
   :special-members:
   :inherited-members:
   :show-inheritance:
.. automodule:: pyrad.io.io_aux
   :members:
   :undoc-members:
   :private-members:
   :special-members:
   :inherited-members:
   :show-inheritance:
```

After having provided all the desired content you can generate the documentation by simply executing the MakeFile. For example, in case of pdf generation:

make latexpdf

Chapter 4 References

[1] https://www.continuum.io/downloads

[2] http://trmm-fc.gsfc.nasa.gov/trmm_gv/software/rsl/index.html

[3] http://wradlib.org/

[4] https://github.com/ARM-DOE/pyart

[5] https://github.com/nguy/artview

[6] https://github.com/nasa/DualPol

[7] https://github.com/tjlang/SkewT

[8] https://github.com/CSU-Radarmet/CSU_RadarTools

[9] https://github.com/nasa/PyTDA

[10] https://github.com/nasa/SingleDop

[11] https://github.com/nasa/PyBlock

[12] https://github.com/meteoswiss-mdr/pyrad

[13] https://github.com/meteoswiss-mdr/pyart

[14] https://www.python.org/dev/peps/pep-0008/

[15] https://pypi.python.org/pypi/pycodestyle

[16] https://pycodestyle.readthedocs.io/en/latest/

[17] http://www.sphinx-doc.org

[18] http://hplgit.github.io/teamods/sphinx_api/html/index.html