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 0.4

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
0.2	fvj	20.12.2018	Major restructure
0.3	fvj	04.02.2019	Changes in conda packages installation section
0.4	fvj	07.02.2019	Changes to highlight the existence of “master” and “dev” branches. Added more specifics on documentation generation

Contents

Figures

Tables

No table of figures entries found.

Chapter 1 What is Pyrad?

Pyrad is a real-time data processing framework developed by MeteoSwiss. The framework is aimed at processing and visualizing data from individual Swiss weather radars both off-line and in real time. It is written in the Python language. The framework is version controlled and automatic documentation is generated based on doc-strings. It is capable of ingesting data from all the weather radars in Switzerland, namely the operational MeteoSwiss C-band radar network, the MeteoSwiss X-band METEOR 50DX radar and the EPFL MXPol radar. It can also read ODIM complying files.

The processing flow is controlled by 3 simple configuration files. Multiple levels of processing can be performed. At each level new datasets (e.g. attenuation corrected reflectivity) are created which can be stored in a file and/or used in the next processing level (e.g. creating a rainfall rate dataset from the corrected reflectivity). Multiple products can be generated from each dataset (i.e PPI, RHI images, histograms, etc.). In the off-line mode, data from multiple radars can be ingested in order to obtain products such as the inter-comparison of reflectivity values at co-located range gates.

The framework is able to ingest polarimetric and Doppler radar moments as well as auxiliary data such as numerical weather prediction parameters (e.g. temperature, wind speed, etc.), DEM-based visibility and data used in the generation of the products such as rain gauge measurements, disdrometer measurements, solar flux, etc.

The signal processing and part of the data visualization is performed by a MeteoSwiss developed version of the **Py-ART** radar toolkit [1] which contains enhanced features. MeteoSwiss regularly contributes back to the main Py-ART branch once a new functionality has been thoroughly tested and it is considered of interest for the broad weather radar community.

The capabilities of the processing framework include various forms of echo classification and filtering, differential phase and specific differential phase estimation, attenuation correction, data quality monitoring, multiple rainfall rate algorithms, etc. In addition, time series of data in points, regions or trajectories of interest can be extracted and comparisons can be performed with other sensors. This is particularly useful when performing measurement campaigns where remote sensing retrievals are validated with in-situ airplane or ground-based measurements. The capabilities of the framework are expanded on an almost daily basis.

A certain degree of parallelization has been included. The user may choose to parallelize the generation of datasets of the same processing level, the generation of all the products of each dataset or both.

Radar volumetric data can be stored in C/F radial format or in ODIM format. Other data is typically stored as csv files. Plots can be output in any format accepted by Matplotlib.

Chapter 2 Installation

IMPORTANT: If you have access to the pre-installed conda environment (For example have access at the CSCS and are part of the msrad group or have access to one of the MeteoSwiss computers where Pyrad is running) the only section that concerns you here is section 2.2.

2.1 Dependencies

Pyrad requires Python 3. The following Python modules are required for Py-ART to work:

- NumPy [2]
- SciPy [3]
- Matplotlib [4]
- Netcdf4-python [5]

Py-ART also has the following optional dependencies:

- NASA TRMM RSL [6]: Adds capability to read other non-standard file formats. Not used by Pyrad
- h5py [7]: Reading of files stored in HDF5 format. Used to read/write ODIM files
- PyGLPK [8]: Linear programming solver if fast LP Phase processing is desired
- Basemap [9]/cartopy [10]: Plotting on geographic maps. Basemap is on the way to be deprecated. Cartopy is its replacement.
- pytest [11]: To run Py-ART unit tests
- gdal [12]: Output of GeoTIFFs from grid objects
- pyproj [13]: A Python interface to PROJ4 library for cartographic transformations
- wradlib [14]: Used to calculate the texture of a differential phase field. Also used to read Rainbow files
- xmltodict [15]: Used to read the Selex-proprietary Rainbow® 5 files.
- metranetlib (only available at MeteoSwiss and CSCS): Used to read the MeteoSwiss radar data files in proprietary METRANET format

Pyrad also has the following optional dependencies:

- pandas [16]: Used for certain applications dealing with time series
- shapely [17]: Used for certain applications to manipulate and analyze geometric objects in the Cartesian plane
- dask (and dependencies) [18]: Used for parallelization
- bokeh [19]: Used to output the profiling results of the parallelization

To automatically create and update the pdf reference manuals sphinx [20], and its dependencies, is used.

Memory profiling in non-parallel processing mode is performed using memory_profiler [21].

To enforce that the code complies with minimum Python style pylint [22] is used.

It is strongly recommended to use Anaconda to manage all the dependencies.

2.2 Getting Pyrad/Py-ART for users or MeteoSwiss developers

Users with access to an already setup conda environment 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 (It is strongly recommended to use your $HOME in order to be able to use some of the Pyrad tools) 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.

Regular users should use the “master” branches of both Pyrad and Py-ART. To check that you use the “master” branch of Pyrad place yourself in the root directory of the project and type:

git branch

And eventually:

git checkout master

And to check that you use the “master” branch of Py-ART go to the directory src/pyart and repeat the procedure above

MeteoSwiss developers should use instead the “dev” branch for both Pyrad and Py-ART.

2.3 Getting Pyrad/Py-ART for developers (external to MeteoSwiss)

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 [32] and the Py-ART submodule [33] and fork them.

Follow the instructions in section 2.2 but with your own username instead of meteoswiss-mdr. Use the “dev” branches of both Pyrad and Py-ART in order to get the most up-to-date code and sync your Pyrad/Py-ART version regularly with the MeteoSwiss one to prevent the drifting of your code.

2.4 Conda installation and pyrad environment creation

Note 1: This section is only necessary for those who do not have access to the pyrad conda environment

Open a shell and get the conda 3 installation file from [23]:

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

Install conda by executing:

bash Anaconda3-x.x.x-Linux-x86_64.sh

and following the instructions.

Create a pyrad environment by typing (currently working with python version 3.5 or higher):

conda create -n pyrad python=3.x

Activate the python environment:

source activate pyrad

Install all the required packages (see section 2.5).

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 with the pathes to the libraries, i.e.:

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
```

2.5 Conda packages installation

Note 1: This section is only necessary for those who do not have access to the pyrad conda environment

Note 2: The paths in the .bashrc/conda environment file here are those for zueub242. 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 [23]. Do not forget to add the path to Anaconda in your .bashrc file. In the case of zueub242 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
```

To avoid conflicts it is recommended to install all the conda packages simultaneously and, whenever possible, from the same conda channel. The TRMM Radar Software Library (RSL) 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 conda environment file):

export RSL_PATH=/opt/anaconda3/

Install the rest of the packages from conda-forge with the following command:

```
Conda install -c conda-forge netcdf4 h5py pytest basemap cartopy gdal
wradlib xmltodict pandas shapely dask bokeh memory_profiler sphinx pylint
```

From these packages netcdf4 is a required dependency for Py-ART, while h5py (to read HDF5 files), pytest (to run unit tests), gdal (to output GeoTIFFS from grid objects), basemap and cartopy (to plot grids on geographic maps) are optional. Basemap is not maintained anymore and the standard has become cartopy. The location of the GDAL data has to be specified by writing in your conda environment file the following command:

export GDAL_DATA=/opt/anaconda3/share/gdal

wradlib is used to read Selex-proprietary Rainbow Rainbow® 5 files and for that it needs the dependency xmltodict.

pandas (to process time series data) and shapely (to extract data in a particular area), dask (for parallel computing), bokeh (to output plots of performance when using dask) and memory_profiler (to check the memory consumption) are optional dependencies of Pyrad. pylint is used to check that the code complies with the Python recommendations and sphinx is used to generate the automatic documentation.

In addition to the standard Py-ART packages, at MeteoSwiss we have created specific libraries to read the ELDES-proprietary format METRANET in which the MeteoSwiss C-band radar network data is stored. For this data, make sure that you have access to the library srn_idl_py_lib.[machine].so and add the path to your conda environment:

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

2.6 ARM-DOE Py-ART installation

Note 1: This section refers to the official Py-ART version from ARM-DOE. We strongly recommend to use the MeteoSwiss version with Pyrad and thus follow the procedure described in section 2.2

Note 2: Make sure to have the latest version of the pyrad repository in your local server.

Note 3: In zueub242 and cscs activate the pyrad environment before installation

Py-ART repository can be found on [24]. A compiled version is available from the conda repository:

conda install -c conda-forge arm_pyart

2.7 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.

2.7.1 ARTView

ARTView is an interactive radar viewing browser. The source code can be found in [25]. The simplest way to install it is using conda:

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

2.7.2 DualPol

DualPol is a package that facilitates dual-polarization data processing. Its source code can be found in [26]. 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 [27]. It provides a set of tools 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 [28]. 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
```

2.7.3 PyTDA

PyTDA is a package that provides functions to estimate turbulence from Doppler radar data. Its source code can be found in [29]. To install it simply download the source code, go to the main directory and type:

```
python setup.py install
```

2.7.4 SingleDop

SingleDop is a package that retrieves two-dimensional low-level winds from Doppler radar data. It can be found in [30]. 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
```

2.7.5 PyBlock

PyBlock estimates partial beam blockage using methodologies based on the self-consistency of polarimetric radar variables in rain. It can be found in [31]. It requires DualPol (see section 2.7.2) 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
```

Chapter 3 Using Pyrad/Py-ART

3.1 Compilation

For the initial compilation of the software activate the conda environment, i.e.:

conda activate pyrad

Then go to pyrad/src and execute:

make_all.sh

This command takes care of compiling both Py-ART and Pyrad. To compile them separately you can use the scripts make_pyart.sh and make_pyrad.sh or see the sections below.

3.1.1 Py-ART compilation instructions

Note: Activate the pyrad environment before installation

To compile 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 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 conda environment file. For example:

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

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

3.1.2 Pyrad_proc compilation instructions

Note: Activate the pyrad environment before installation

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 compile 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 purposes it exists 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 fully implement the changes made by the the developer the built installation has to be completely clean up. 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
```

To compile Pyrad one has always to remember to first compile Py-ART and then Pyrad. If you only modified code in pyrad_proc you do not need to recompile Py-ART 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.

3.2 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.

3.3 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 an ending point in time.

main_process_data_period.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).

There are a number of tools to automatize the fetching of the data, processing, etc. in the CSCS. Have a look at pyrad/tools to see what is useful to you.

3.4 Getting help

3.4.1 Bug reporting and request for new functionalities

To report a bug in Pyrad/Py-ART use the Issues page of the Pyrad repository in github: https://github.com/meteoswiss-mdr/pyrad/issues

Use that page also to report issues with the MeteoSwiss Py-ART. You can also use the Issues page to request new functionalities.

If you would like to add a new functionality by yourself it is strongly recommended to use that page also so that we can coordinate the development and see how it fits to the whole program.

3.4.2 Other documentation

For specific information about the functions implemented in Pyrad/Py-ART have a look at the automatically generated pdfs contained in pyrad/doc:

- pyart-mch_library_reference_dev.pdf
- pyart-mch_library_reference_users.pdf
- pyrad_library_reference_dev.pdf

- pyrad_library_reference_users.pdf

For an overview of the monitoring functions implemented with Pyrad you can read the document pyrad_monitoring_fvj.pdf also available in pyrad/doc.

3.4.3 Developers contact

Pyrad is maintained by the RadarV team of the Radar, Satellite and Nowcasting Division of MeteoSwiss. The current points of contact are:

- Jordi Figueras i Ventura: jordi.figuerasiventura@meteoswiss.ch
- Jacopo Grazioli: jacopo.grazioli@meteoswiss.ch
- Zaira Schauwecker: zaira.schauwecker@meteoswiss.ch
- Martin Lainer: martin.lainer@meteoswiss.ch

Chapter 4 Developing Pyrad

4.1 The Pyrad git architecture

A schematic of the Pyrad git architecture can be seen in Fig. 1. The Pyrad project contains 5 main directories: config stores the configuration files, doc contains relevant documentation about the project, tools contain useful tools for data management, docs contain the html pages of the online documentation 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.

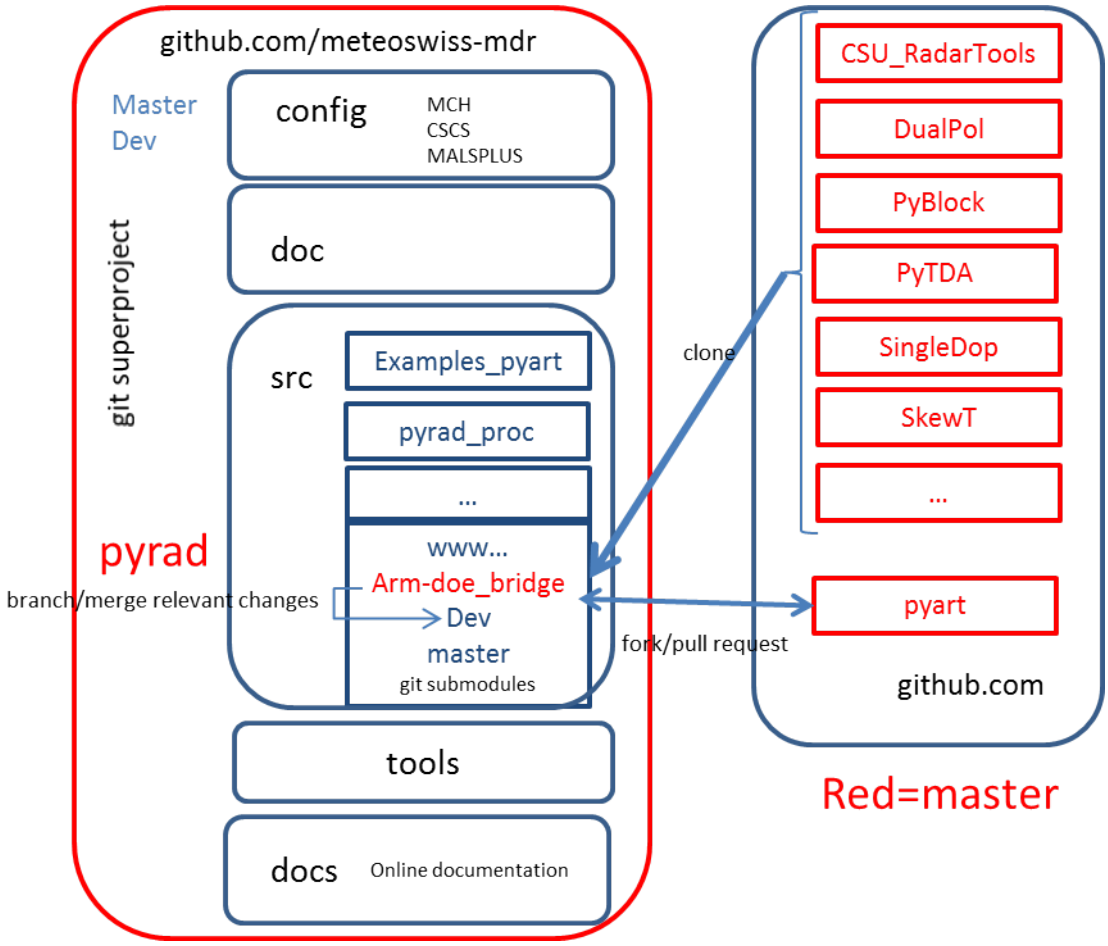

Fig. 1 The Pyrad superproject architecture

The Pyrad project is stored in a repository in github [32]. It has two branches:

- The “master” branch: is the stable branch that should be used operationally and for users that do not wish to develop new projects.

- The “dev” branch: is used to test new Pyrad features.

The MeteoSwiss Py-ART submodule was forked from the Py-ART repository [24] and is placed in the github repository [33]. Regularly it has three branches:

- The “master” branch: is the one used by the stable branch of Pyrad. Operational uses and for users that do not wish to develop new projects should make sure to use this branch

- The “arm-doe_bridge” branch: is used to sync the ARM-DOE Py-ART with the MeteoSwiss Py-ART. This branch is intended for use only by the Principal Investigator (PI) of the Pyrad project.

- The “dev” branch: is used to test new features. This should be the branch used by internal MeteoSwiss developers.

New ad-hoc branches may be created to push new features to the ARM-DOE Py-ART.

4.2 Code style

Pyrad and its submodels follow the PEP8 standard [34]. To make sure that your code formally complies with the standard make use of the pylint tool. The simplest use is to type:

pylint [your_file.py]

A list of errors and their location will appear.

4.3 Developing the Pyrad git superproject by internal MeteoSwiss collaborators

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 (in our case Py-ART) 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 dev)

   git branch

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

   git checkout dev

5. Add or remove the files you want to commit with the regular commands git add and 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 (in our case Pyrad) 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.

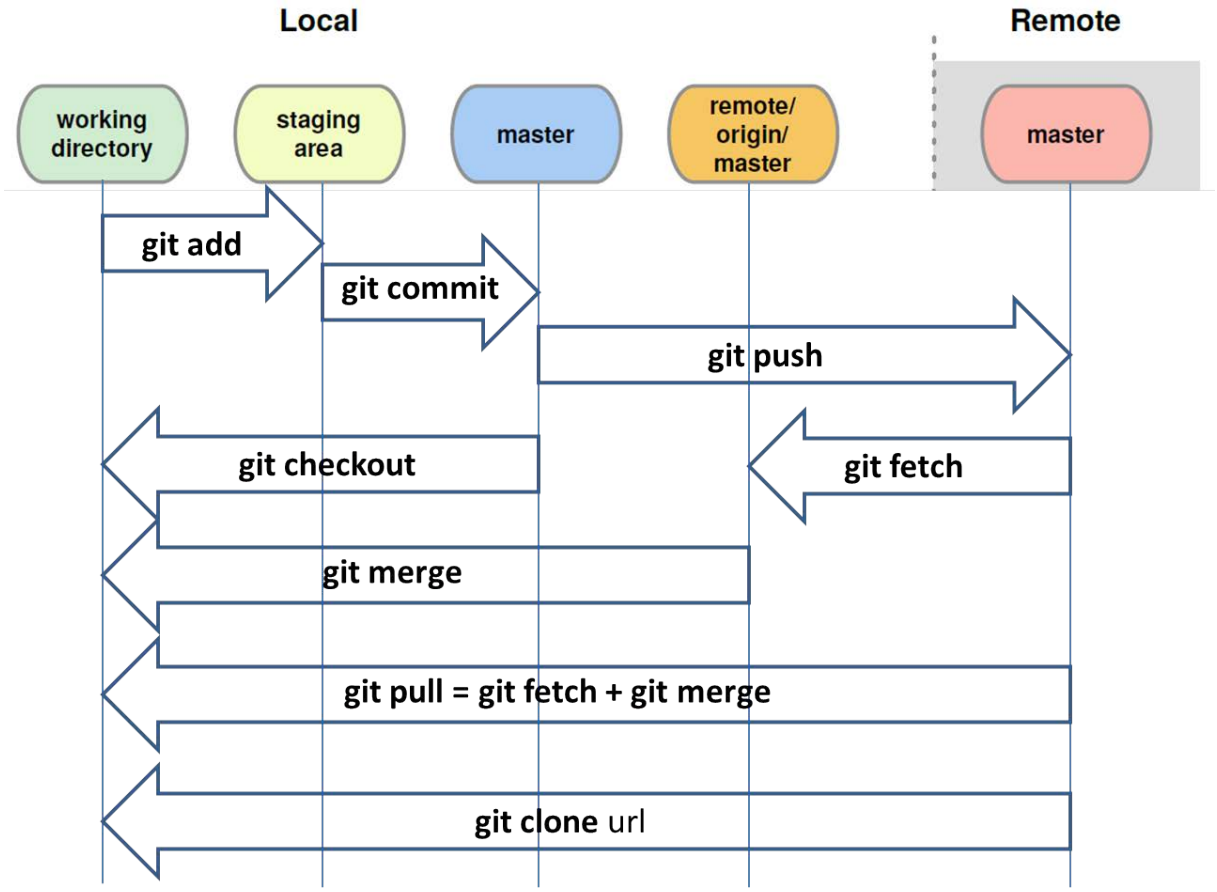

Fig. 2 Git flow diagram

4.4 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. Before even modifying the code we recommend to use the Pyrad Issues pages and tell us what you would like to do so that we can coordinate our actions.

To develop your local version of Pyrad and its submodules the instructions on section 4.3 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 4.5.2 and 4.5.3 respectively. **It is strongly recommended that you create a branch specific for the changes you would like to submit to the Pyrad superproject.**

4.5 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.

4.5.1 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 two new local branches of the forked version, “dev” and “arm-doe_bridge”. “dev” is the branch where local developments will be made. “arm-doe_bridge” will be use to sync our modules with the public modules:

   git checkout –b dev

4. Add the information of your working branch into your git config file. If in Pyrad master:

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

   If you are in the Pyrad dev use dev. This is another good point to commit to the repository.

4.5.2 Updating the local submodule dev branch with changes in the master public library

1. Place yourself in the superproject directory (Pyrad) and “dev” branch 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 arm-doe_bridge

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 “bridge” branch and change branch otherwise and pull to update the local branch with the changes in the public library:

git branch

git checkout arm-doe_bridge

git pull

3. Synchronize 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 dev:

git checkout dev

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

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

9. Now to update your “dev” branch simply place yourself in the submodule directory and merge the “dev” with it:

git merge arm-doe_bridge

10. Solve any possible conflicts that arise and test the various functionalities, commit and push the result.

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

4.5.3 Transferring changes from the local submodule dev 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 dev branch:

git branch

If you are not in the master branch change it:

git checkout dev

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 dev branch you can use git checkout and the path to the new file, for example:

git checkout dev pyart/correct/noise.py

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

git checkout --patch dev 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 checkout to the regular pyart branch but do not forget to switch branches if changes in 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

4.6 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.

4.7 Automatic Generation of Documentation

4.7.1 Sphinx config file creation

To automatically generate documentation you have first to make sure the package Sphinx 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 [37].

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

4.7.2 Pyrad/Py-ART documentation

There are four reference documents that need to be created/updated. The Pyrad reference manual for users, the Pyrad reference manual for developers, the Py-ART MCH reference manual for users and

the Py-ART MCH reference manual for developers. For all those documents a .pdf version is generated.

For the Py-ART and Pyrad reference manual for users, when in the master branch, an html version is also used. The html Pyrad reference manual version is located in pyrad/docs and the html Py-ART reference manual is located in pyrad/src/pyart/docs. In this way the documentation is shown in the github pages.

The process to generate/update documentation has been automatized:

- To generate the pyrad documentation go to pyrad/doc/pyrad, activate the pyrad environment and execute the file make_pyrad_doc.sh

- To generate the Py-ART documentation go to pyrad/doc/pyart-mch, activate the pyrad environment and execute the file make_pyart-mch_doc.sh

Chapter 5 References

[1] https://arm-doe.github.io/pyart/

[2] http://www.numpy.org/

[3] https://docs.scipy.org/doc/

[4] https://matplotlib.org/

[5] http://unidata.github.io/netcdf4-python/

[6] https://trmm-fc.gsfc.nasa.gov/trmm_gv/software/rsl/

[7] http://www.h5py.org/

[8] http://tfinley.net/software/pyglpk/

[9] https://matplotlib.org/basemap/

[10] https://scitools.org.uk/cartopy/docs/latest/

[11] https://docs.pytest.org/en/latest/

[12] https://www.gdal.org/

[13] http://jswhit.github.io/pyproj/

[14] https://wradlib.org/

[15] https://github.com/martinblech/xmltodict

[16] https://pandas.pydata.org/index.html

[17] https://shapely.readthedocs.io/en/latest/

[18] https://dask.org/

[19] https://bokeh.pydata.org/en/latest/

[20] http://www.sphinx-doc.org/en/master/

[21] https://pypi.org/project/memory-profiler/

[22] https://www.pylint.org/

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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