Introduction to REALTA

A User Guide for I-LOFAR Computer System

Aoife Brennan

SUMMER RESEARCH INTERNSHIP, I-LOFAR, BIRR, IRELAND

This user guide was written as part of a larger summer project under the supervision of Dr. Evan Keane with the financial support of the I-LOFAR and BreakThrough Listen Project, Berkeley.

First release, July 2020

Contents

0.1 Overview 3
0.2 Accessing REALTA 4
0.3 I-LOFAR Computer System 4
0.4 REALTA Layout 5
0.5 Using Dockers 7
0.6 Useful Commands 8
0.7 Observing with REALTA 10
0.8 Data Processing 12
0.9 Resources 15
0.10 Acknowledgements 16

0.1 Overview

I-LOFAR is an Irish radio observatory, a part of a more extensive network of telescopes observing at low radio frequency (10 – 240) Mhz, collectively referred to as LOFAR. REALTA is the backend system for I-LOFAR and can be used as a gateway to connect to the I-LOFAR observing array, via (ucc1). REALTA is composed of four nodes (ucc1, ucc2, ucc3, and ucc4), all of which are mounted on. The first node (ucc1) is used for running observing scripts, acting as a connection between the physical observing system and data processing nodes. After observing using (ucc1), data can then be transferred from (ucc1) to the data processing nodes (ucc2, ucc3, ucc4). Many different data files can be produced when observing with I-LOFAR and processed by scripts written in many different languages. Furthermore, many different data processing softwares are installed, frequently in the form of dockers. The following user guide is divided into nine sections, as listed in contents.

Section two describes how to access REALTA, by ssh into the REALTA gateway. Section three describes the general structure of the I-LOFAR computer system, giving an overview of the system's back end. Section four gives a brief guide to navigating processing nodes using simple bash commands. Section five describes the usage of dockers on REALTA. Section six demonstrates useful data processing commands with applications. Section seven gives an example of an observing script, along with a general methodology for observing in (ucc1). Section eight contains a sample data processing script. Section nine gives resources to software guides and command references that are useful when using REALTA.

0.2 Accessing REALTA

The easiest way to access REALTA is to ssh into the gateway from either a Mac or Linux operating system. Before accessing REALTA on a Windows system, a terminal like program must be installed, resources for windows are listed in section five. The command terminal for both a Mac or Linux operating system will work. REALTA can is accessible by using the ssh command to access the Linux gateway @lgc.lofar.dias.ie. First, a whitelisted IvP4 must be obtained the IvP4 used must be stable and pingable. After whitelisting an IvP4, a username and password will be assigned. The command below demonstrates how to ssh into REALTA, where username is the unique username assigned after whitelisting.

```
$ ssh username@160.6.237.10
```

In order to access REALTA in a more timely manner a public key can be generated, this can be used to authenticate the user without entering a password when logging in. An alias can also be used, to shorten the command used to ssh into REALTA from our first command example to a shorten command as follows,

```
$ ssh lgc
```

Resources for creating and using both aliases and public keys and listed in section nine.

0.3 I-LOFAR Computer System

I-LOFAR consists of two different types of array, the low band array (LBA) which observes in the range of 10 MHz to 90 Mhz, and consists of an array of pole like antennas. The second array, the high band array (HBA), observes in the range 110 MHz to 240 Mhz. The (HBA) array consists of a series of tile-like antennas. The input signals from the (LBA) and (HBA) arrays are transported by coaxial cables, to the receiver control unit (RCU) which is located in the I-LOFAR cabinet. The receiver unit performs input selection, then the signal is amplified and filtered by a A/D converter. After this, the signal goes into the remote station processing board (RSP). In the (RSP), the signal goes into a first in first out buffer (FIFO). Then goes through a polyphase filter, which is a fast Fourier transform implementation of a bandpass filter bank. The filter makes the signal bandwidth small enough for beamforming to take place. In addition to beamforming, the RSP boards can calculate subband statistics, beamlet statistics, and array covariances.

The Transient buffer boards (TBB) are not part of the main signal processing chain, but their function is to store a short period of raw voltage data. The first two components (RCU and RSP) are in the I-LOFAR cabinet. The terminal panel located in the control center acts as a control unit for the I-LOFAR system. I-LOFAR, also has a series of switch which act in various ways to connect computes to each other. First, the fibre switch is used to connect the computer data storage to various servers. Next, the Ethernet switch acts as a central station connecting computers, to each other. Lastly, InfiniBand (IB) is a computer networking communications standard used in high-performance computing that features very high throughput and very low latency. It is used for data interconnect both among and within computers. REALTA is shown in figure 1 in yellow, and orange. The first node nuig1 is a storage node which can be accessed by ssh into nuig1 from the gateway. The node ucc1 is the observing node for I-LOFAR, the other node (ucc2, ucc3, and ucc4) are used for data processing. The last two nodes seen in the diagram are apart of the Breakthrough Listen initiative. The first (blh0) is a headnode used for logging in to the data processing node (blc).

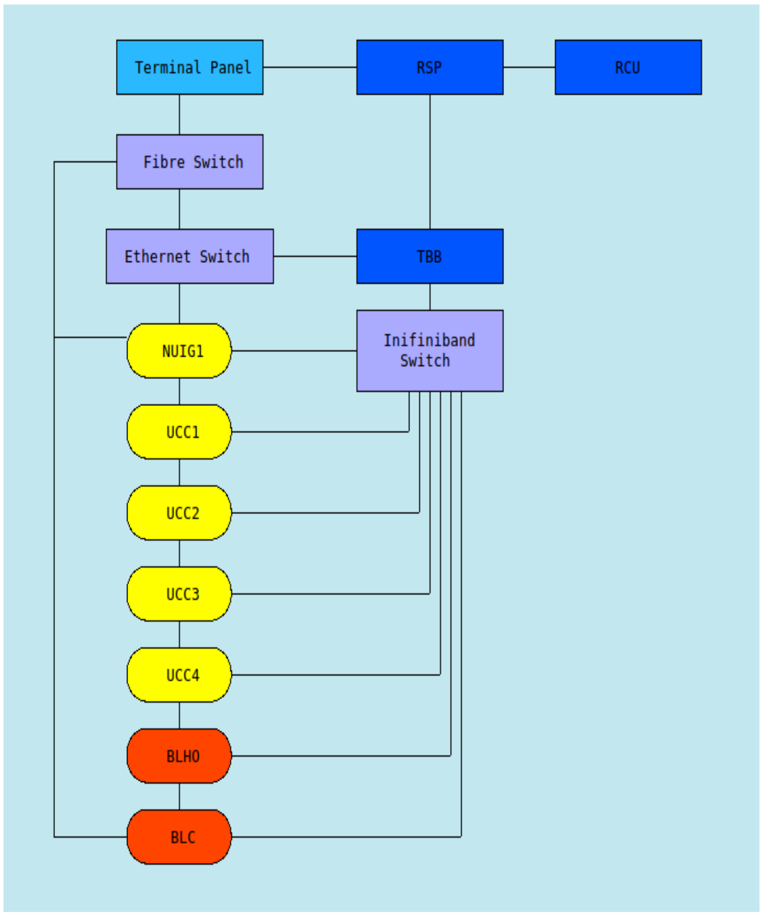

Figure 1: Diagram of I-LOFAR computer system structure

At the time of the writing of this user guide, the Breakthrough Listen computer system has been installed, and is shown in the diagram, operation and usage of breakthrough listen head node and node are not covered in this guide. Resources with greater descriptions on individual components of I-LOFAR are provider in section nine.

0.4 REALTA Layout

REALTA, is a Linux like system, therefore normal bash commands can be used to navigate the system. This section assumes that access to REALTA via a white-listed IvP4 has been obtained, and therefore a user name and password, alias as described in section two are also used. First from your home terminal ssh into the gateway, where lgc is an alias for username@160.6.237.10.

```
$ ssh lgc
```

From this gateway we can ssh into nodes (nuig1, ucc1, ucc2 and ucc4). Please note that nodes can only be accessed from this gateway, it is not possible to ssh from one processing node to another. If unsure of current working directory the command pwd will print out the current directory that the user is in. Next ssh into one of the data processing nodes for this example use ucc2.

```
$ ssh ucc2
```

The command below will ssh into your sub-folder, scripts should be stored here, it is also possible to store a small amount of data. Be very careful as there is a limited amount of space, only keep data in a personal subfolder, if the desired data can not be located elsewhere on the system.

```
$ cd /mnt/ucc2_data1/data/yourusername
```

All data processing and running of scripts should take place in ucc2 data1, if running a script outside a personal directory please consider where the output files are going, later in this section I have discussed the running of scripts and how to redirect the output to a personal sub-folder. As this is your own directory you can make directories and files as you see fit.

```
$ mkdir first_directory
```

```
$ nano first.file
```

Two useful bash commands for managing files are cp and mv. The first command copies the input file and places the output file in the specified directory. The second moves the original file into the specified directory. Please note if you are copying a file from outside your directory, to your own directory please use cp instead of mv. If moving the file inside a personal directory,the output path can be greatly simplified.

```
$ cp input.file /mnt/ucc2_data1/data/yourusername
```

Many different languages can be used to write scripts, but beware in order for the script to be executable a "shebang" must be used. A "shebang" line indicating the language that the script is written in, this must be at the top of the script. Scripts can be created and run using standard bash commands, for example running the a script called script.file.

```
$ source script.file
```

Scripts can be run in any directory in a data processing node, but please be careful if running scripts outside a personal sub-folder. In order to view graphs that are outputted by your script, an x server can be used to generate a display window. First make sure that your current operating system has an x server installed, I personal use x11. In order to use a display window you need to add -x to both ssh commands for example,

```
$ ssh -X lgc
```

```
$ ssh -X ucc2
```

The command xeyes can be used to test if a display window is available in lgc. After ssh into (ucc2), test if a display window is available using the commands

```
$ gnuplot
```

```
$ test
```

If your preferred language is python, python scripts can be used. In general it is far more useful to try write as many scripts as possible in shell. There are many very useful bash packages which can be used as alternatives to the most used python packages. For example Gnuplot is a very useful plotting software, installed on REALTA, and is easy to use.

```
$ gnuplot
```

```
$ set xlabel "x axis"
```

```
$ set ylabel "y axis"
```

```
$ title"first plot"
```

```
$ plot sin(x)
```

0.5 Using Dockers

When data processing in REALTA there are many useful software installed in order to simplify matters. Many of these software are installed on dockers. Docker is a set of platforms, that delivers software in packages called containers. Containers are isolated from one another and bundle their own software. You can use a docker by running the following generic command,

```
$ docker run -v host/directory:/container/directory
```

This command will let the user mount a container, which contains the desired processing software to a file, which contains the data to be processed. The flag -v creates a volume, which are the preferred mechanism for persisting data generated by and used by docker containers. Other useful flags that should be used when using dockers are,

```
$ docker run --rm -it -v host/directory:/container/directory
```

The flag –rm is important as it will close the docker after use, if –rm is not used the docker with keep running even after the user has exited the docker. If this happens multiple dockers can be left running on REALTA. To view all the current dockers that are running use the command docker ps. A word of warning many other user could possible be running dockers, so take care, for this reason its better to use –rm flag so you don't have to worry about closing dockers after you have finished with them.

When mounting containers to files it is advisable to define as general a mount as possible. The following command mounts to home local, which allows the user to access all directories in home local while inside the docker.

```
$ docker run -it -rm -v /mnt:/mnt -v /home:/home_local ‘‘container’’
```

Please note that a container name must be inserted in place of "container" in the above command. When inside the docker, the user can cd into other directories, and run scripts. If the data that you wish to process is in a personal directory after running the docker command, simply run the script as normal. If the data is in another directory in a processing node, cd into that directory run the script, making sure to direct the output back to a personal subfolder. If the data is in another node, a path must be defined to the data inside your script. This script can then be run inside your personal subfolder, where the output of that script should be stored. Multiple files can be mounted to a container at the same time, multiple containers can also use the same volume at the same time. It is also important to note that any file created while inside the docker, can only be edited inside that docker. Further information on dockers can be found in section nine.

0.6 Useful Commands

There are many useful bash commands that can greatly speed up data processing, and avoid the use of lengthy python scripts. With REALTA, data files can be very large, and take many different forms, possibly composed of many different columns. Each bash command listed below can be used to manipulate files in order to carry out data processing in a timely manner. Furthermore all of these commands can be piped together.

1.) Tail

Tail can be used to skip lines in a file, which is very handy for removing headers, the command below skips the first line of the input file.

```
$ tail -n+2 input.file > output.file
```

Tail can be used to print the last "n" number of lines in a file. The command below prints the last three lines in file file.txt. A plus symbol can be inserted in place of the minus symbol to give the first three lines of file.txt. If you wanted to use multiple files you can add the flag -q. By default the command tail followed by a file name will output the last ten lines of a file.

```
$ tail -n 3 input.file > output.file
```

2.) Cat

The command cat can be used to view the context of a single file or multiple files. The command cat followed by the file to read works for convectional file formats, niche file types that are generated while processing data in REALTA might need to be opened in a different way. For example filterbank files and timeseries file can't always be opened with cat only, the command below could be used instead,

```
$ cat filename | od -f
```

The od command in Linux is used to convert an input file into octal format, the flag -f converts to floats. Another useful function of cat is to combined files into one large file, the below command works on many file formats, this can be done as follows,

```
$ cat input1.file input2.file > output.file
```

3.) Head

The command head can be used to view the header of a file, this command works for many file formats, for more niche file formats, that can be generated during data processing a header command from a specialised processing program may be needed. For example filterbank and timeseries files, that are produced using sigproc, which is a pulsar processing program. For these files the sigproc command header will need to be used instead.

```
$ head input.file
```

4.) Grep

The command grep can be used for searching a file for either a word, or string. The following command looks for the word "cat" in the the file file.txt.

```
$ grep ‘‘cat’’ file.txt
```

5.) Awk

The command awk can be used to print columns of a file or extract columns. The first command prints the first column of a file, and creates a new file with only the first column. The command below can be used both if a file containers headers, but also when headers aren't present.

```
$ awk ’{print $1}’ input.file > output.file
```

If a file has two columns of with headers 1 and 2, and you wanted to create a new file with only values in the first column that where greater then four but less then one hundred, the following could be used,

```
$ awk  ’($1 > 8)  && ($1 < 100)’ input.file > output.file
```

The command below pipes together three different commands, this can make searching, sorting and getting desired values for data processing easy. The command below is an example of piping where dollar sign 1 means first column, dollar sign 2 means second column, and dollar sign 0 means all columns. The sort -g sorts numerically (rather than alphabetically) -r means reverse order and -k2 means second column. In the command below, awk ignores header, and values in the third columns that are less then 8000, the values are then sorted, and the top 10 values are printed.

```
awk ’NR!=1{if ($3<8000) print $0}’ input.file  sort -gr -k4 | head -10|
```

6.) Sed

Sed can be used to replace word, numbers or patterns in a file. The following command replaces the word one with the word two.

```
$ sed  ’s/one/two/’ input.file
```

7.) Uniq The uniq command can be used to remove duplicate lines in a file for example,

```
$ uniq input.file
```

Adding the flag -c will print the amount of times the line was repeated, the flag -d will print the repeated lines.

0.7 Observing with REALTA

Before observing, an observation script must be written, which contains information specifying starting parameters, pointing, and recording preferences. Observation scripts can be run in ucc1, after observing the data gathered can be transferred to the relevant processing nodes. Observing with I-LOFAR is quite easy, and a standard script as shown below needs little adjustments. Before writing an observing script it is important to properly choose targets. For example, is the target visible from Birr, how bright is the target, does the target rise high enough and for long enough?

In regards to the brightness and elevation of a target, it is important to note that the projected area of a LOFAR station is different with zenith angle, as the arrays are stationary. Therefore if you are offset by an angle, the length would appear to be shortened by a factor of cosine(zenith). Considering this we might need a dim target to rise to at least a forty-five degree elevation for a considerable time. After choosing a suitable target an observing script can be written. The first block consists of general parameters, these parameters normally don't need to be changed for different targets. Important things to note in this first block are, the swlevel which refers to the software level of the station. The control processes of a LOFAR station are organized into six level. The first three levels are used in local mode when the telescope is observing as a lone system, the last three levels are used when the station is observing as a network, with the other LOFAR stations across Europe.

After changing software level, the script has to wait for a register update (rspctl) to complete. In summary, beam forming requires software level three, while usage of the RCU, RSP boards, and TBBs only requires software level two. All processes of the levels below the current level are also available to the user.

```
echo ‘initialising: SWLEVEL 2’

eval swlevel 2

rspctl  --wg=0

sleep 1

rspctl --bitmode=8

rspctl --bitmode

sleep 1

killall beamctl

sleep 3

echo ‘initialising: SWLEVEL 3’

eval swlevel3
```

```
sleep2

rspctl --splitter=0

sleep 3

rcus='0:83,86:191
```

The next block is the pointing block, the first command gives the pointing coordinates, the second command specific the beamforming. LOFAR carries out beamforming by converting equatorial coordinates to complex beamformer weights, digital and analogue beam pointing directions are updated with regular intervals. The pointing coordinates need to be correctly inputted in radians.

```
pointing="RA(rad(-pi/2,pi/2)),DEC(rad(-pi/2,pi/2)),J2000"

beamctl --antennaset=HBA_JOINED --rcus=$rcus --band=110_190  --beamlets=0:487

--subbands=12:499 --anadir=$pointing --digdir=$pointing &

bash sleepuntil.sh YYYYMMDD HHMMSS.0

killall -9 beamctl

swlevel 0
```

For example if you wanted to observed the pulsar PSR J0250+5854, with ICRS coordinates 02 50 17.78 +58 54 01.3, the pointing command would read,

```
pointing='0.7430579301269316,1.0280052319831219,J2000'
```

The last block that is needed in an observing script is the recording block. The observing start and end time must be inputted by the user, a target name must be added, which can be anything the user chooses. For example if a you wanted observing to start at four in the morning on first of the July in the year 2020, the observing start command would read,

```
obs_start="2020-07-01T04:00.00.0"
```

The sleepunitl.sh time should be set to a few minutes before the observation starts. The remainder of the scripts does not normally need to be changed.

```
obs_start="YYYY-MM-DDTHH:MM:SS.0

obs_end="YYYY-MM-DDTHH:MM:SS.0"

target="name"

bash sleepuntil.sh YYYYMMDD HHMMSS
```

```
starts

echo $target $obs_start $obs_end

logdate=‘date +"%Y%m%d%H%M%S"‘

bash generic_ucc1_timestamps.sh $obs_start $obs_end ’dump_udp_ow_12’ $target 2>&1

tee -a $logdate’_$target.log’
```

In section nine the LOFAR cookbook is linked, which gives great details, on flags used when observing.

0.8 Data Processing

This section demonstrates examples of data reduction, with the aim of providing the user with an idea of how to process different types of data on REALTA. The usage of individual software used in the examples or analysis of results are not covered in detail in this manual. The first two examples use a software called sigproc, which is installed on REALTA, in containers. Sigproc is a pulsar data processing software, but can also be used for fast radio bursts (FRB). The first data reduction exercise uses filterbank file (.fil), that was recorded by Parke.

For our purpose a filterbank file can be thought of as an array of bandpass filters that separates the input signal into multiple components. We can located the filterbank files that we will be using in /mnt/ucc2 data1/data/dmckenna/SMC. If you cd into this directory you can see that there are thirteen files labeled 1-D. Each .fil file corresponded to a different beam, which points at a different location in the sky.

In order to use sigproc, we need to use a container that has sigproc installed, we will use the container pulsar-gpu-dsp2020 for this exercise. The container needs to be mounted to the data file, in order to process the data using sigproc commands, but first a data processing script should be created in a personal subfolder. The simple script shown below will first dedisperse each of the thirteen files for a range of dispersion measurements. Dedispersion is the process by which a filterbank file is corrected for the effects of interstellar dispersion, by the the interstellar medium. The dispersion measurement depend on both the electron density and distance between the target and the observer. Normally a range of dispersion measurements are used, as the correct value is often unknown. The command dedisperse will output time series files, the script below will output five hundred of these files for each beam. For example the outputted time series file,

```
"SMC021_008A1.2.tim"
```

This file is a time series file for the beam A, that has undergone a trial dedispersion of 2.

```
#!/bin/sh

for file in  1 2 3 4 5 6 7 8 9 A B C D; do echo
        for DM in {0..500}
        do
```

```
                dedisperse "SMC021_008""$file""1.fil" -d $DM >
                /mnt/ucc2_data1/data/personal_subfolder/
                "SMC021_008""$file""1.""$DM"".tim"
                seek "SMC021_008""$file""1.""$DM"".tim" -fftw -pulse -s  >
                /mnt/ucc2_data1/data/personal_subfolder/
                "SMC021_008""$file""1.""$DM"".pls"
                rm "SMC021_008""$file""1.""$DM"".tim"
                rm "SMC021_008""$file""1.""$DM"".prd"
                rm "SMC021_008""$file""1.""$DM"".hst"
                rm "SMC021_008""$file""1.""$DM"".top"

        done
done
```

After dedisperse, a search command called seek is used, this commands carries out both a periodicity search and a single pulse search (-pulse). Spectrum files can also be outputted using the flag -s. The command rm is used to remove all files that are generated by the seek command, that won't be used in this exercise. It is always good practice to only keep necessary files in a subfolder. Also note that all files have been directed back to a personal subfolder.

The use of a tmux session is suggested, when scripts like the one above have a long run time. When inside a session, REALTA can be used as normal. After running the script the user can simply leave the session without closing it, which allows the script to run in the background. Create a session before running the script above using the following command,

```
$ tmux new -s first_session
```

When inside the session, login into docker, and run the following command,

```
$ docker login

$ docker run -it -rm -v /mnt:/mnt -v /home:/home_local pulsar-gpu-dsp2020
```

Please note that in order to run a docker, you must have a docker account. The above command preforms a very general mount, as described in section five. When inside the docker containing sigproc, the script which contains sigproc commands can be executed. Next cd into the directory containing the data, and run the script. After this script has finished you should see single pulse search files (.pls), and periodicity search files (.prd), in the personal directory, that was used in the script.

You should observe that (.pls) files have five columns, the first is dispersion measurements, the third is time, and the fourth signal to noise. For this example we will create single pulse graphs, for each of the thirteen beams. Next, we need to sort and merge files, we want to combine all the (.pls) files for one beam together. This can be accomplished by removing the headers using tail. Then merge all the .pls for each beam individually into one large file using the cat command. You should end up with thirteen files.

```
#!/bin/sh
```

```
for DM in {0..500}; do echo

                tail -n+2 "SMC021_008D1.""$DM"".pls" >>
                "SMC021_finished.""$DM"".pls"
                cat "SMC021_finished.""$DM"".pls"  >>
                "SMC021_008D1_finished.pls"
                rm "SMC021_finished.""$DM"".pls"
done
```

Next we can use the awk command to extract the columns we want to plot. Its also important to remove background noise in order to properly observe the signal, for this exercise use a signal to noise value of six. You can use the awk command to only extract columns where the fourth column is greater then 6. You should end up with three .csv files for each of the thirteen beams, which should be very easy to plot.

```
awk '{$4!>6 && $4!<100}'  SMC021_00811_finished.pls  >> SMC021_00811_finished.pls
awk '{print $1}'  SMC021_00811_finished.pls >>  DM.csv
awk '{print $3}'  SMC021_00811_finished.pls >>  time.csv
awk '{print $4}'  SMC021_00811_finished.pls >>  SN.csv
```

Now plot the thirteen beam graphs separately using something like the script below. Examining the plots its clear that most plots look the same, with constant lines of radio frequency interference (RFI). Examining graphs for the beams 6, 7 and D a unique signal that is absent in the other graphs can be observed. This signal is the FRB, we can zoom in to view the (FRB), further analysis can be carried out on the signal to determine, the pulse duration, DM and SNR.

```
#!/usr/bin/python

from glob import glob
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import csv

x = []
y = []
z = []
with open('time.csv','r') as csvfile:
    plots = csv.reader(csvfile)
    for row in plots:
        x.append(int(float(row[0])))
```

```
with open('DM.csv','r') as csvfile:
    plots = csv.reader(csvfile)
    for row in plots:
        y.append(int(float(row[0])))

with open('SN.csv','r') as csvfile:
    plots = csv.reader(csvfile)
    for row in plots:
        z.append(int(float(row[0])))

graph = plt.scatter(x, y, c=z, s = .1, cmap=plt.cm.plasma)
cb = plt.colorbar(graph)
cb.set_label('SN')
plt.xlabel("time")
plt.ylabel("DM")
plt.title("SMC021_008D1.fil")
plt.savefig('/place to save/final.png')
plt.show()
```

0.9 Resources

Resources are divided into three section, the first gives resources for LOFAR, the second gives resources for useful bash data processing commands, the third gives useful astronomical software that can help with observational planning. Resources included in this section are scientific papers, user guide, tutorials, and blogs, all resources are inserted as links.

9.1 Section One (LOFAR resources)

(station cookbook)
(github tutorials for examples of data processing on users local machine)
(LOFAR calculators)
(LOFAR long term archive)
(LOFAR image gallery)
(LOFAR architecture)
(LOFAR general overview)
(docker user guide)
(document on mounting containers to files)

9.2 Section Two (bash commands)

(useful awk commands)
(generating public key)
(setting up alias)
(using grep command)
(using uniq)

9.3 Section Three (astronomical resources)

(target search engine)
(pulsar catalogue)
(pulsar data processing software SIGPROC)
(SIGPROC user guide)
(planetarium software)

0.10 Acknowledgements

I would like to express my very great appreciation to Evan Keane for mentoring me during my internship. I would also like to offer my special thanks to David McKenna, for all the guidance during the internship. I would like to thank Steve Croft and Howard Isaacson for organizing and running the BL internship program. Finally, I would like to thank everyone at I-LOFAR and BL.