gscpy Documentation

Release 22.11.2018

Ismail Baris, Nils v. Norsinski

Nov 30, 2018

Contents

CHAPTER 1

Introduction

The goal is the development of Python GRASS modules for the automatic download, processing and analysis of Sentinel-1 data. All modules can be executed in GRASS or in a terminal. The aim of this work was that a user can process all data without much effort and import them directly into GRASS GIS for further analysis. By entering certain metadata you can use this module to search for newly processed files and import them into a database.

Here is an overview of the content:

- A simple module that import Scripts from a package to GRASS GIS script directory.
- Database management modules where it is possible to create create entire databases or mapsets.
- Data download including basic adjustments for Sentinel-1 with sentinelsat.
- A SAR pre-processing add-on for GRASS GIS based on SNAP processing workflow which uses pyroSAR.
- Modules to import all files in a directory by considering a certain pattern. Moreover, it is possible to import these data in different `mapsets`.
- A module that can import `pyroSAR` dataset in a directory based on their metadata.
- Creation of space-time cube.

The package pyroSAR and sentinelsat is used for the pre-processing and download of sentinel data.

1.1 Modules

This packages include the following modules:

- i.script: A simple module that import Scripts from a package to GRASS GIS script directory.
- g.database: Create a GRASS GIS Database.
- g.c.mapset: Create a mapset in a GRASS GIS Database if it is not existent.
- s1.download: Data download including basic adjustments for Sentinel-1 with sentinelsat.
- i.dr.import: Import data into a mapset from a file by considering a certain pattern.

- i.fr.import: Import pyroSAR datasets in a directory based on their metadata.
- pr.geocode: Wrapper function for geocoding SAR images using pyroSAR.
- t.c.register: Creation of Sentinel-1 space-time cube.

CHAPTER 2

Installation

After you have received the *gscpy* package, you can install it with

```
$ python setup.py install
```

After this process it is recommended to use the script `i_script` with GRASS GIS. This is necessary because some modules from this package call other modules from this package that are only present if they are located in the script folder of GRASS GIS. It is possible that some of these modules require administration rights. The reason for this is that, for example, when downloading data to the hard disk, any write permissions must be present.

To launch a Python script from GUI, use File -> Launch Python script and select /path/to/gscpy/i_script.py.

Now you can launch the following modules:

- i.script: A simple module that import Scripts from a package to GRASS GIS script directory.
- g.database: Create a GRASS GIS Database.
- g.c.mapset: Create a mapset in a GRASS GIS Database if it is not existent.
- s1.download: Data download including basic adjustments for Sentinel-1 with sentinelsat.
- i.dr.import: Import data into a mapset from a file by considering a certain pattern.
- i.fr.import: Import pyroSAR dataset in a directory based on their metadata.
- pr.geocode: Wrapper function for geocoding SAR images using pyroSAR.
- t.c.register: Creation of Sentinel-1 space-time cube.

CHAPTER 3

Examples

Here are some examples of how you can use the gscpy package with GRASS GIS. In the examples most of the modules are executed within the GRASS terminal. It is also possible to do all these steps with the graphical GUI. Most of the examples are from FOSS4G 2014 workshop

3.1 Data

These data are from here. These data are already in a GRASS GIS database. To show the import routines I exported the files within the mapset *climate_2000_2012* with *out.l.gdal*. This mapset contains temperature and precipitation series for North Carolina from State Climate Office of North Carolina.

3.2 Contents

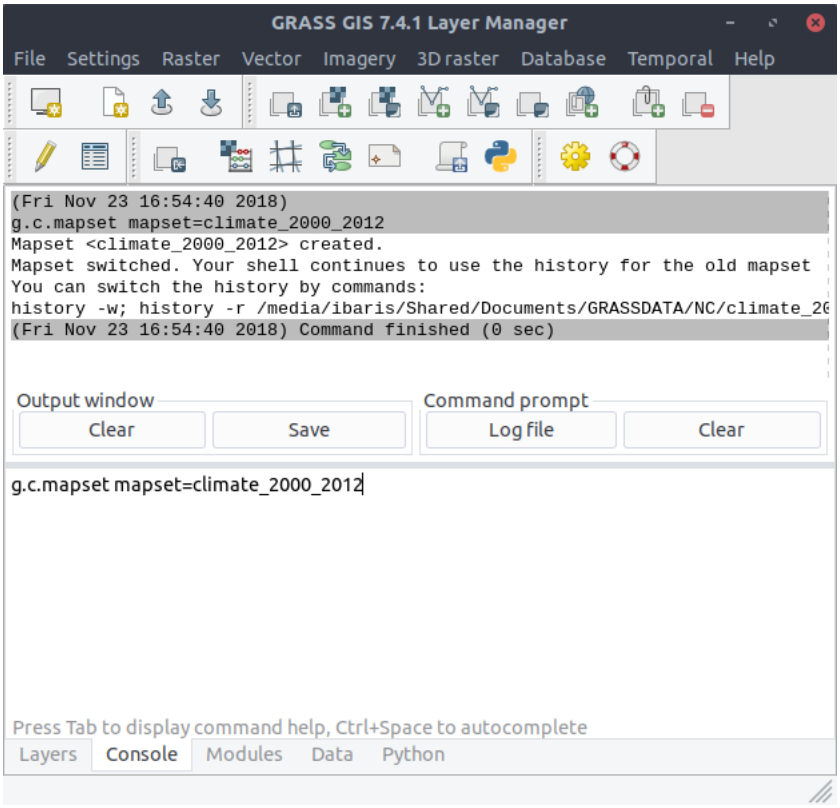
3.2.1 Data

The data is from here. The data is already in a GRASS GIS database. To show the import routines I exported the files within the mapset *climate_2000_2012* with *out.l.gdal*. This mapset contains temperature and precipitation series for the whole North Carolina from State Climate Office of North Carolina.

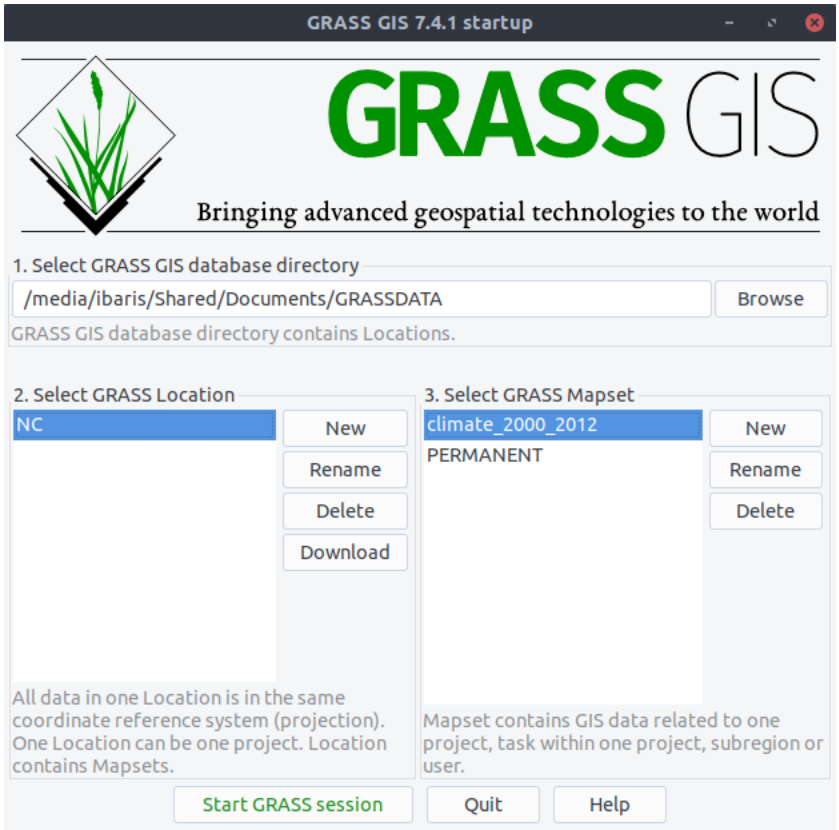
3.2.2 Create a Mapset

Start GRASS with a location and create a mapset *climate_2000_2012*. Therefore, you can use `g.c.mapset`:

```
$ g.c.mapset mapset=climate_2000_2012
```



After we start GRASS GIS new the created mapset is present:



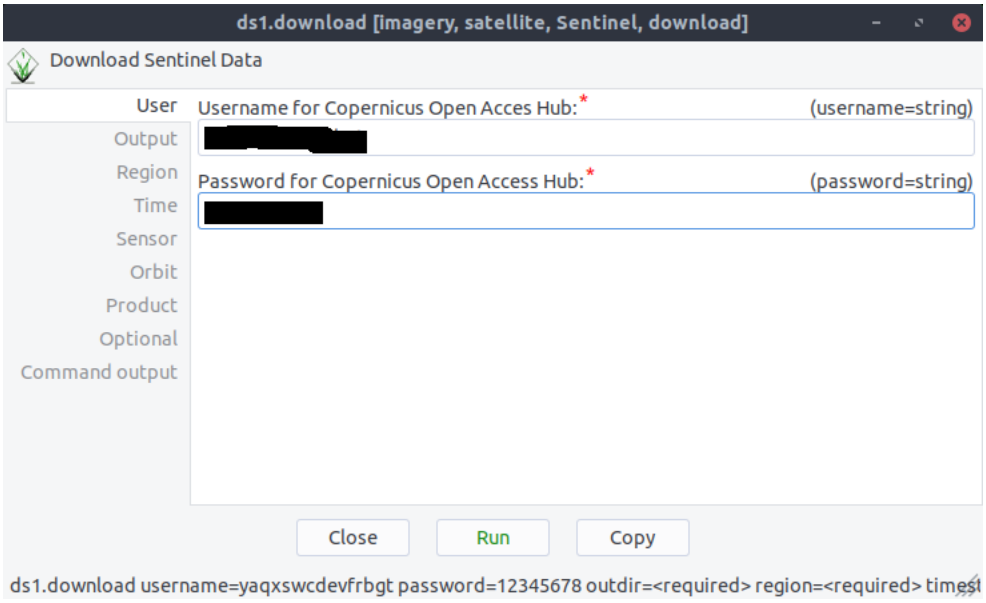
3.2.3 Sentinel 1 Downloader

Here is an example of how to use the Sentinel downloader, to map all available data *(flag: -p)*:

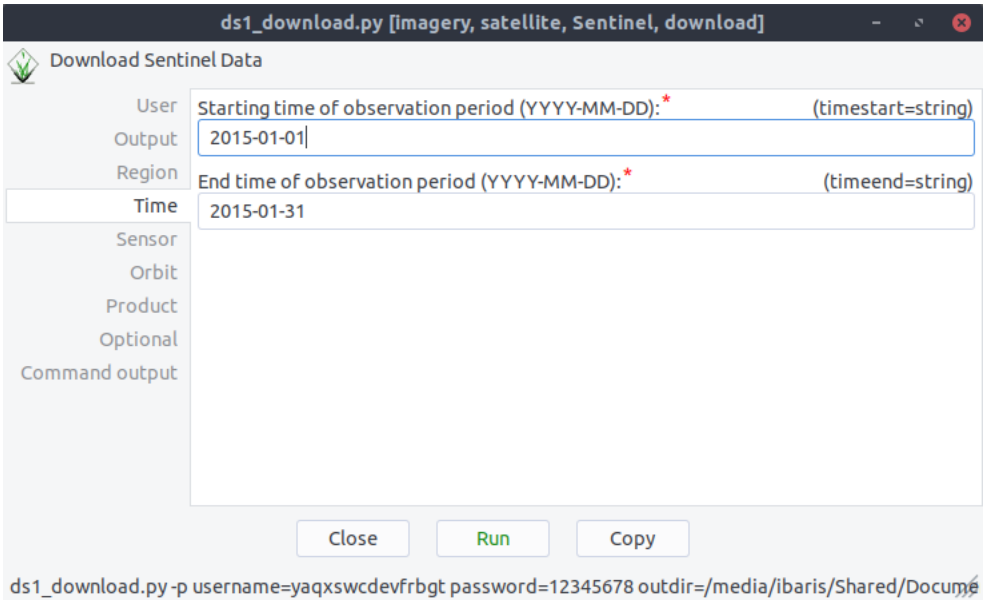
```
$ ds1.download -p username=DALEK password=exterminate region=myGEoJsOnFile.geojson␣
↪timestart=2015-01-02
  timeend=2015-01-12 outdir='home/usr/data' producttype=SLC
```

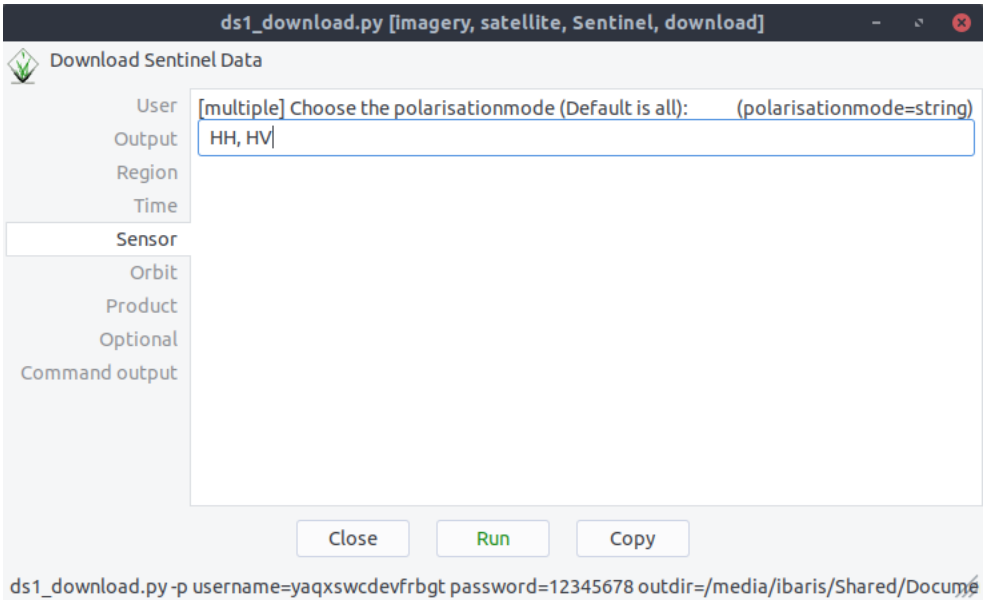
Another way is to use the GUI:

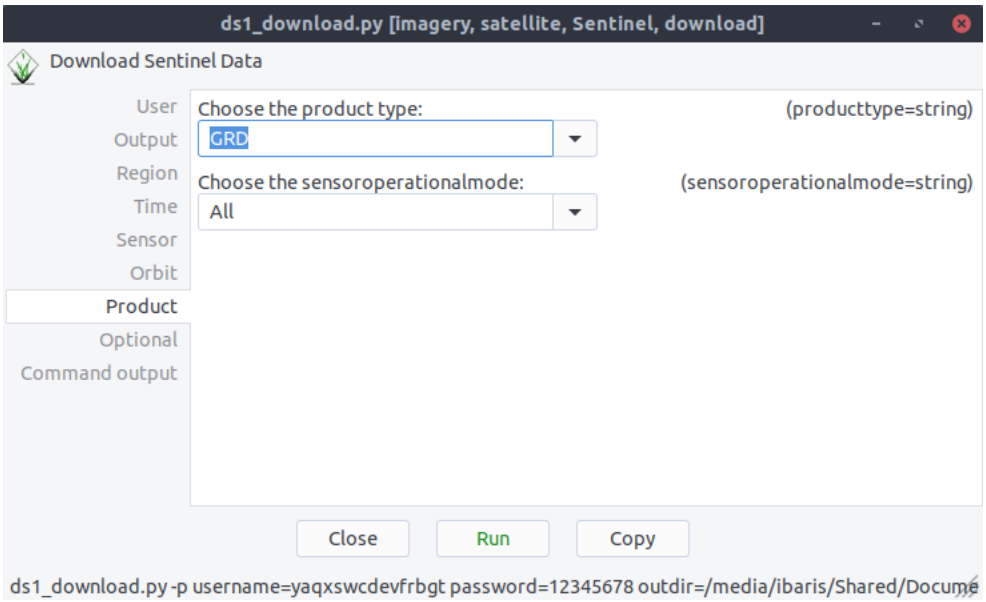
Type your Username and Password:



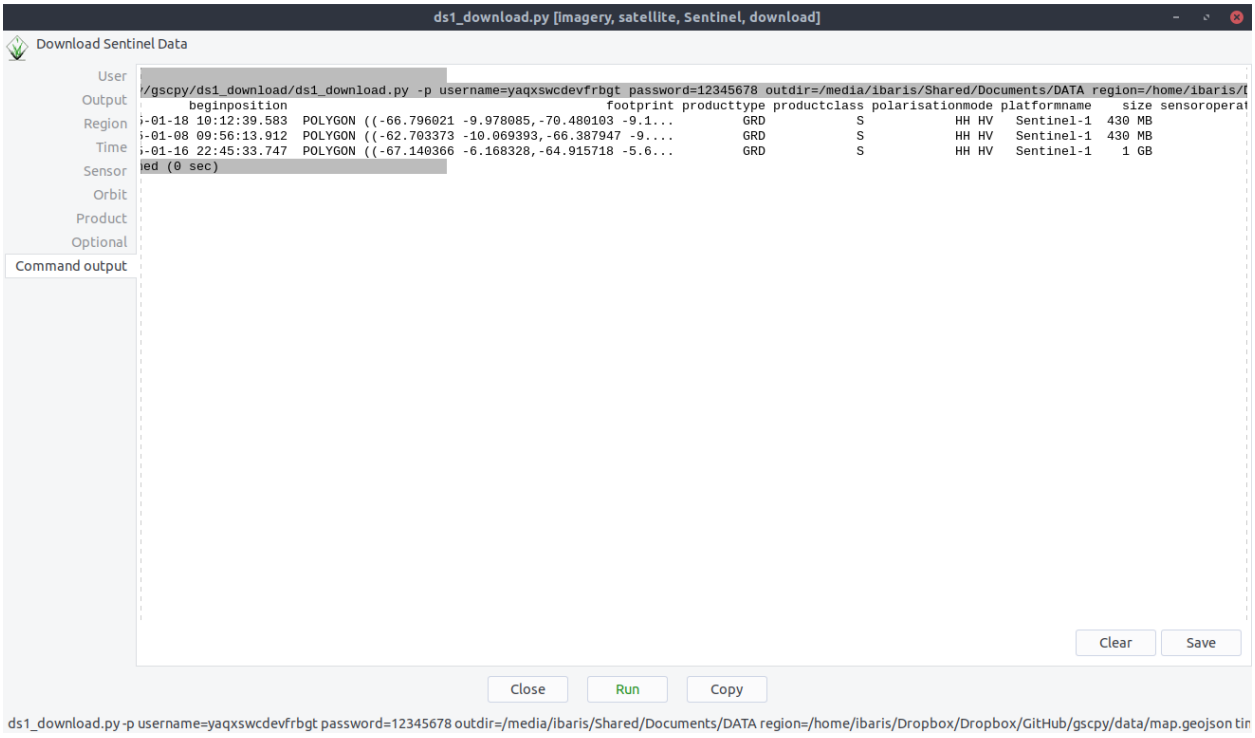
After the definition of a region with a geojson file you can specify the sensing period, polarization and the product type:





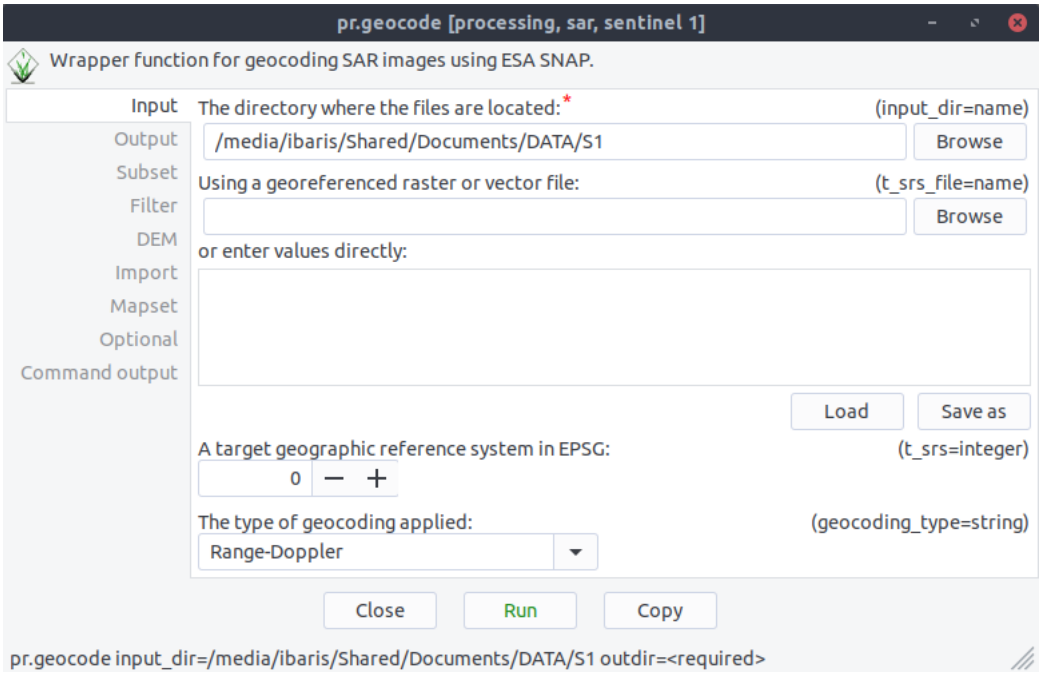


With *flag -p* you can print all available products. If the flag is missing all data will be downloaded:

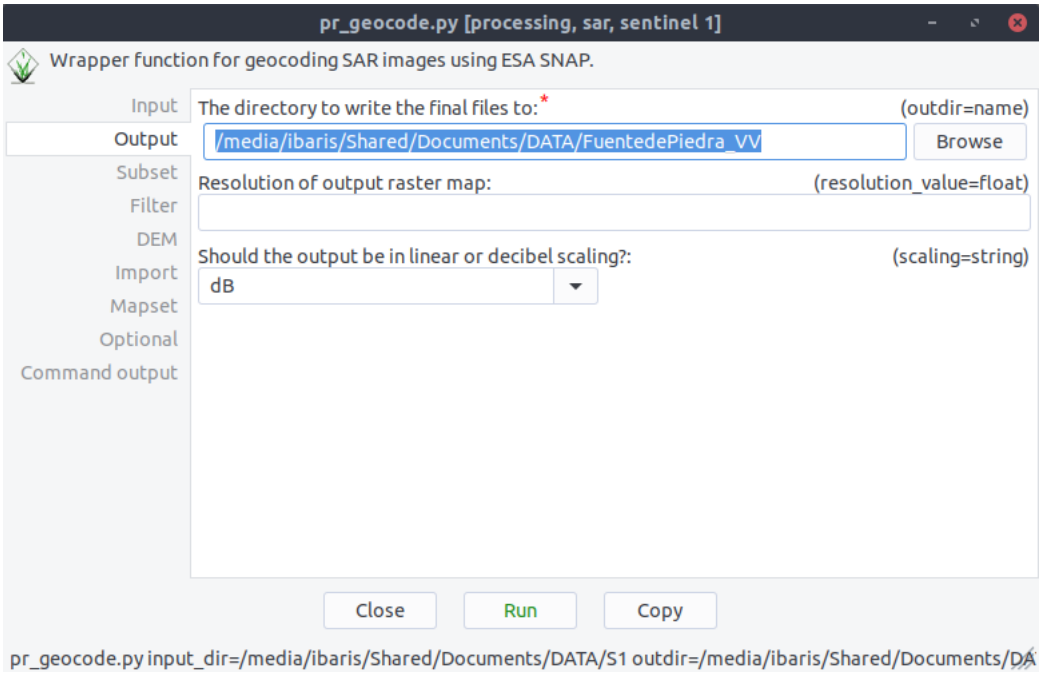


3.2.4 Geocoding Example

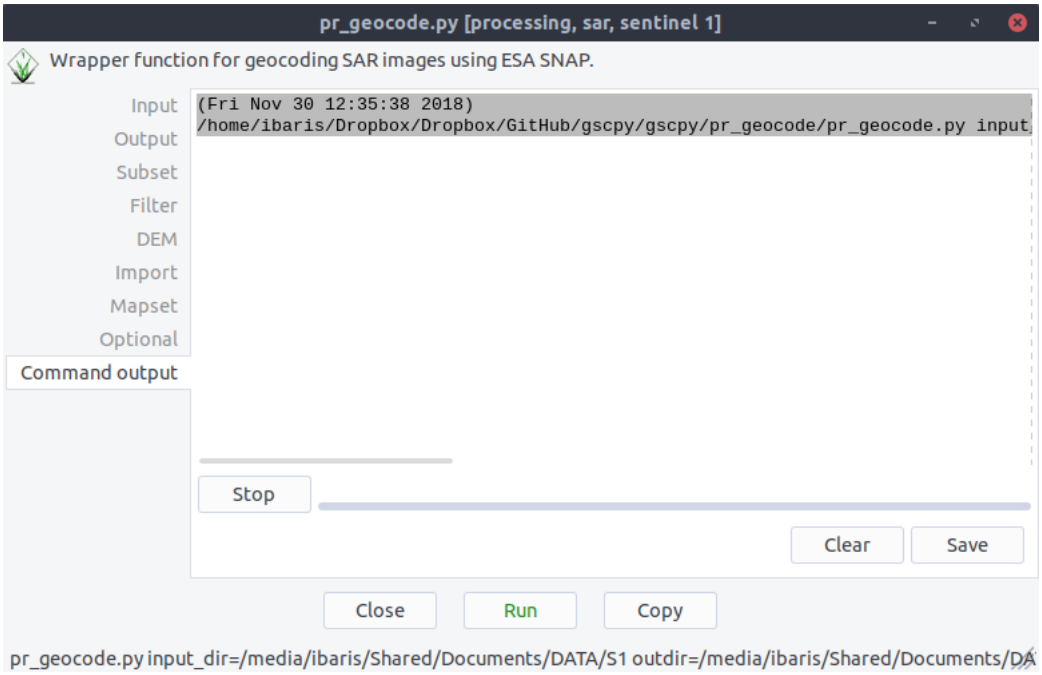
After we downloaded the Sentinel 1 Files with `ds1.download` we want to geocode all the files automatically. Thus, we start with command `pr.geocode` the geocode GUI. Now we can specify our directory where the sentinel data are:



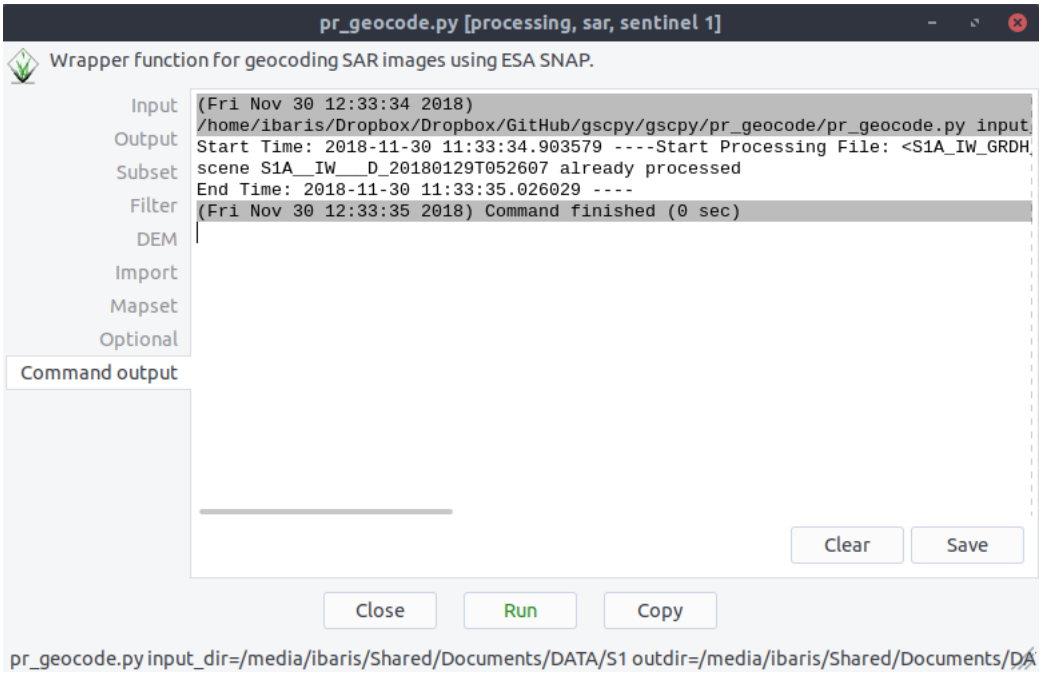
After these we specify our output directory:



if we click on `Run` now the geocode processing will run with ESA´s SNAP software:

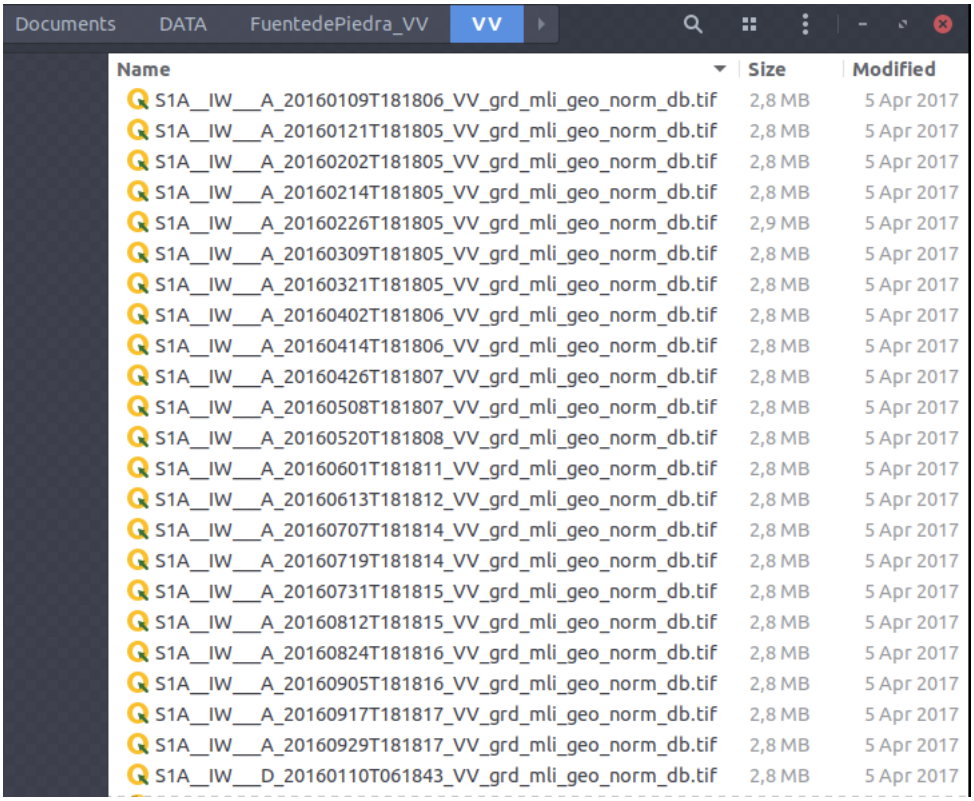


If there is any scene that is already processed the `pr.geocode` module will skip these files:



3.2.5 Find Processed Scenes

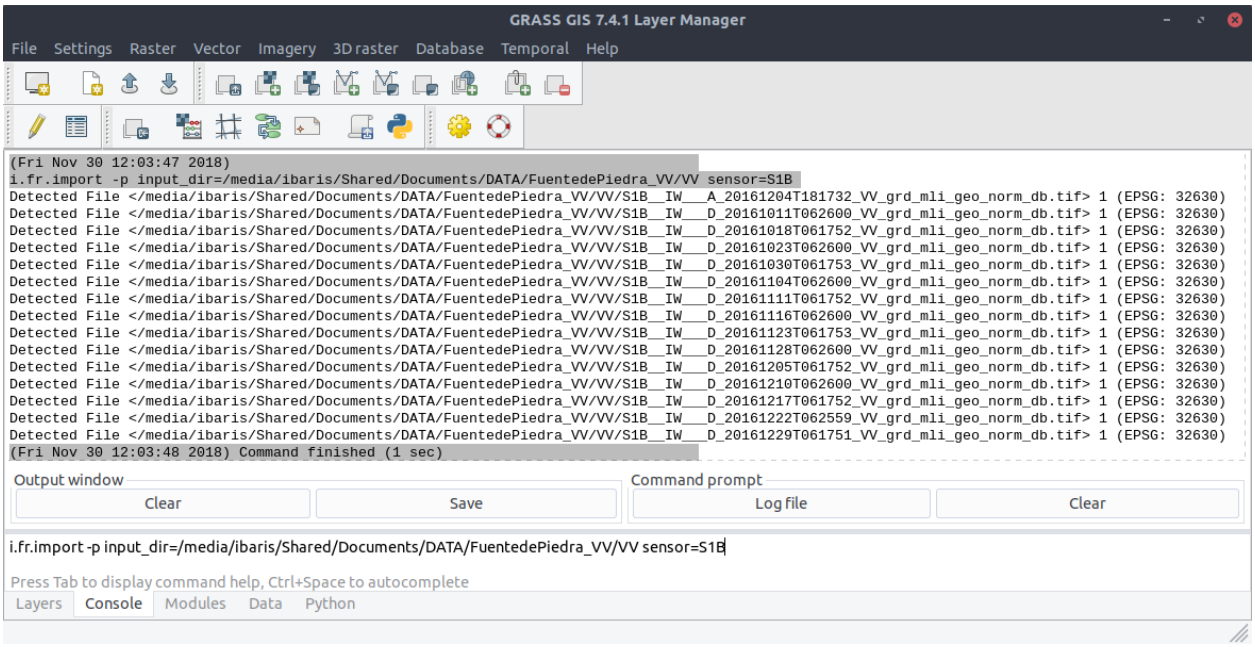
After we downloaded the Sentinel 1 Files with `pr.geocode` all the processed files are find in the output directory:



In this directory their are S1A scenes as well as S1B scenes. Consider a case where we want to find and import all the sentinel 1B scenes. Thus, we can run the command:

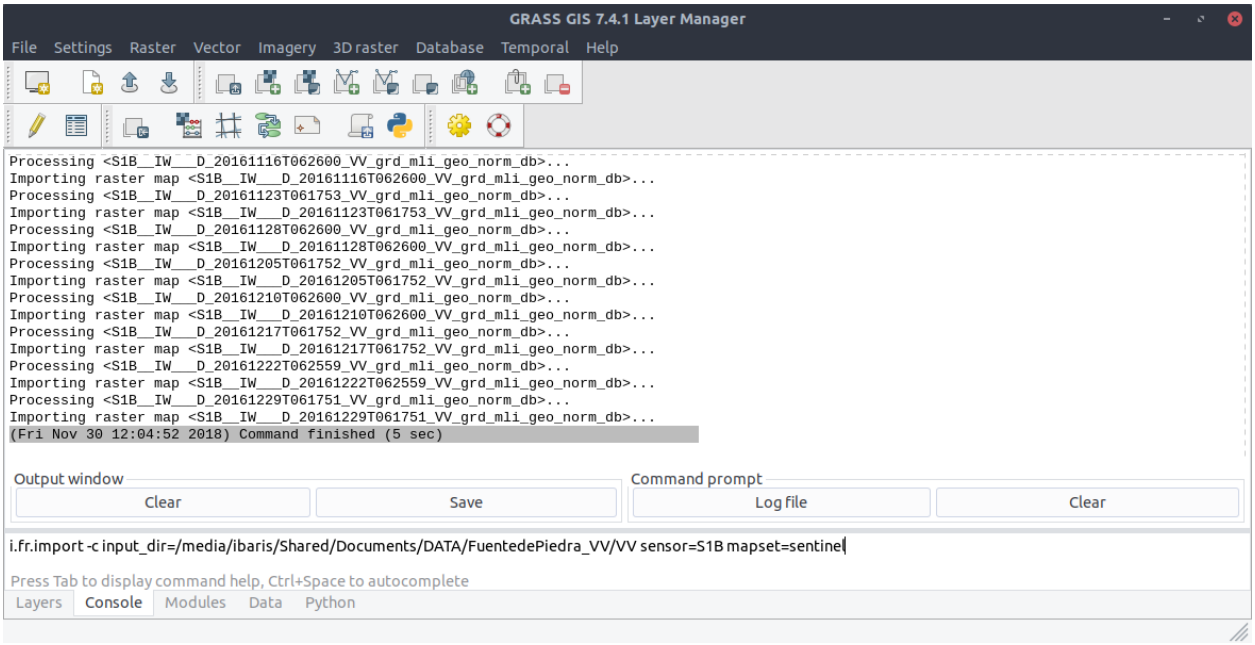
```
$ i.fr.import -p input_dir=/media/ibaris/Shared/Documents/DATA/FuentedePiedra_VV/VV␣
↪sensor=S1B
```

to print all the detected scenes:

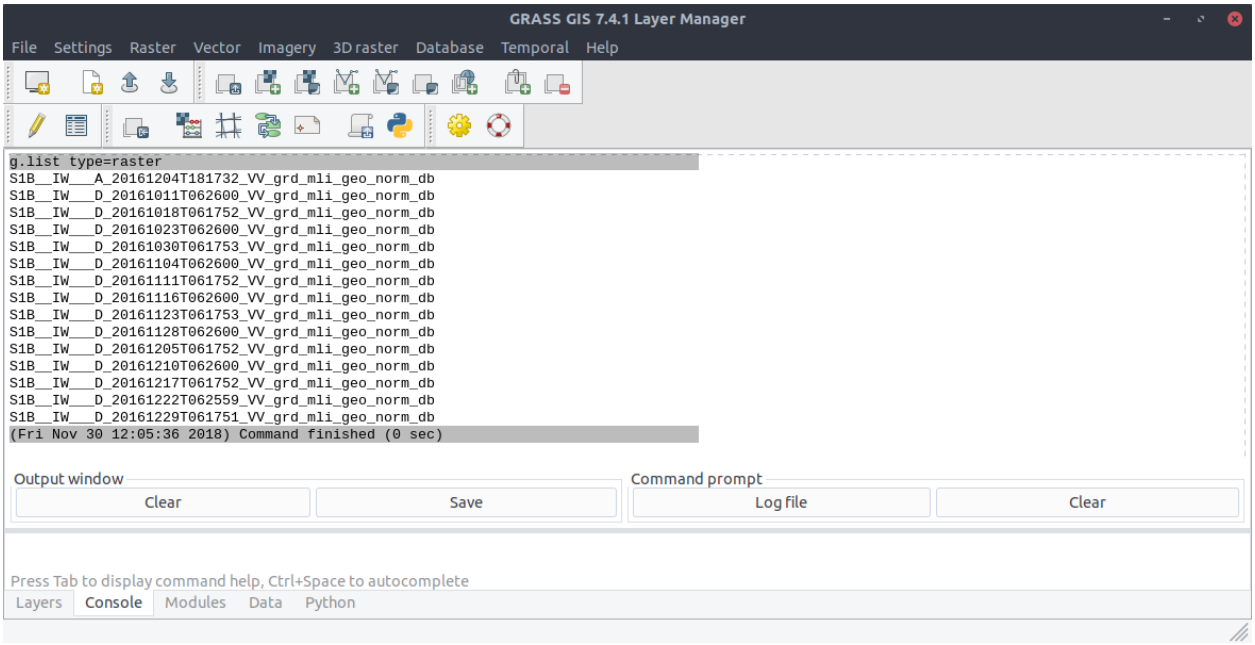


We can import all the detected scenes if we delete the flag `-p`:

```
$ i.fr.import input_dir=/media/ibaris/Shared/Documents/DATA/FuentedePiedra_VV/VV␣
↪sensor=S1B
```



To be sure we can run the command `g.list type=raster`:

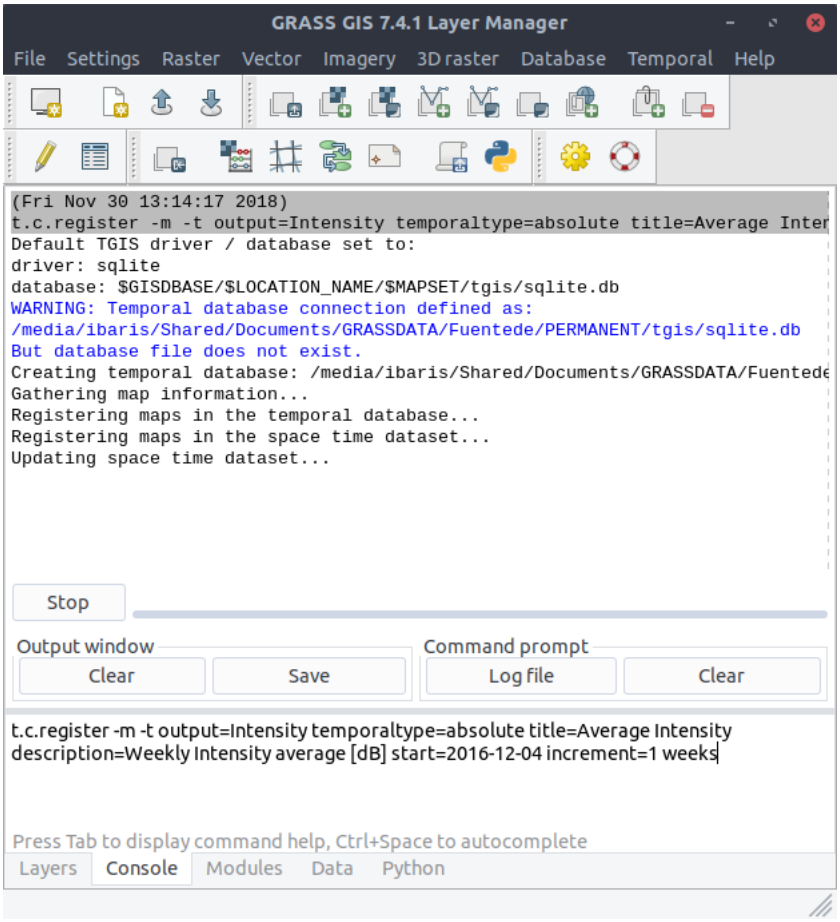


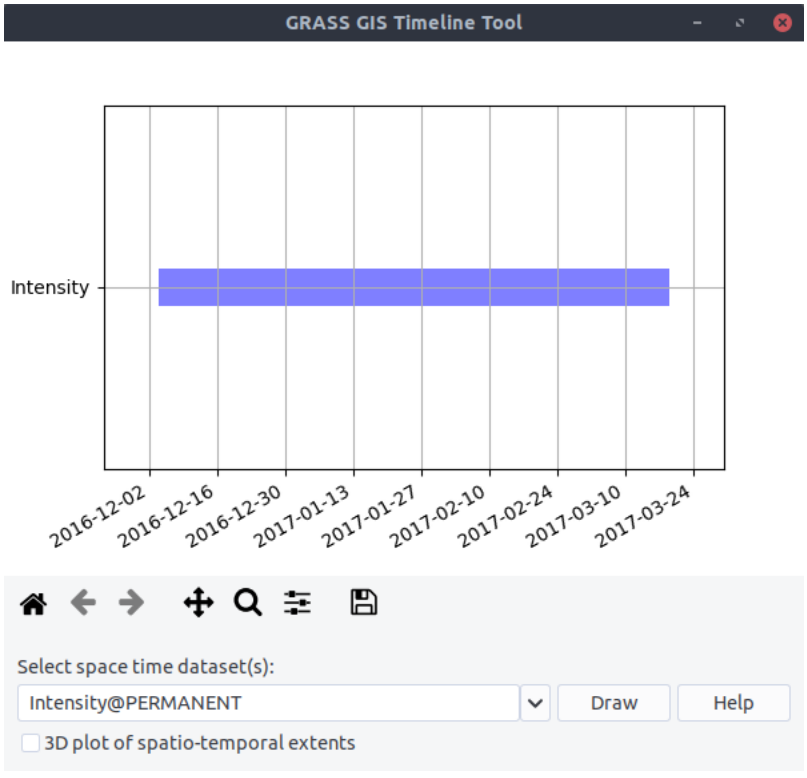
3.2.6 Spatio-temporal data handling with Sentinel Data

After the process of geocoding and import of the scenes we create a huge amount of data as shown in section *Find Processed Scenes*. To better handle the long time series of maps, we create temporal datasets which serve as containers for the time series and we will further manipulate them instead of individual maps. Usually, we create empty datasets of type strds (space-time raster dataset) and after that we register the raster files into the strds. With `t.c.register` we can combine these two steps and with the *flag -m* we will visualize the temporal extents of the dataset (Note, that we use absolute and weekly time.):

```
$ t.c.register -m -t output=Intensity temporaltype=absolute title="Average Intensity"
  description="Weekly Intensity average in [dB]" start=2016-12-04
  increment="1 weeks"
```

Look at the temporal extents:


GRASS GIS 7.4.1 Layer Manager
File Settings Raster Vector Imagery 3D raster Database Temporal Help
(Fri Nov 30 13:14:17 2018)
t.c.register -m -t output=Intensity temporaltype=absolute title=Average Inter
Default TGIS driver / database set to:
driver: sqlite
database: $GISDBASE/$LOCATION_NAME/$MAPSET/tgis/sqlite.db
WARNING: Temporal database connection defined as:
/media/ibaris/Shared/Documents/GRASSDATA/Fuentede/PERMANENT/tgis/sqlite.db
But database file does not exist.
Creating temporal database: /media/ibaris/Shared/Documents/GRASSDATA/Fuentede
Gathering map information...
Registering maps in the temporal database...
Registering maps in the space time dataset...
Updating space time dataset...
Stop
Output window
Clear
Save
Command prompt
Log file
Clear
t.c.register -m -t output=Intensity temporaltype=absolute title=Average Intensity
description=Weekly Intensity average [dB] start=2016-12-04 increment=1 weeks
Press Tab to display command help, Ctrl+Space to autocomplete
Layers Console Modules Data Python


GRASS GIS Timeline Tool
Intensity
2016-12-02
2016-12-16
2016-12-30
2017-01-13
2017-01-27
2017-02-10
2017-02-24
2017-03-10
2017-03-24
Select space time dataset(s):
Intensity@PERMANENT
Draw
Help
3D plot of spatio-temporal extents

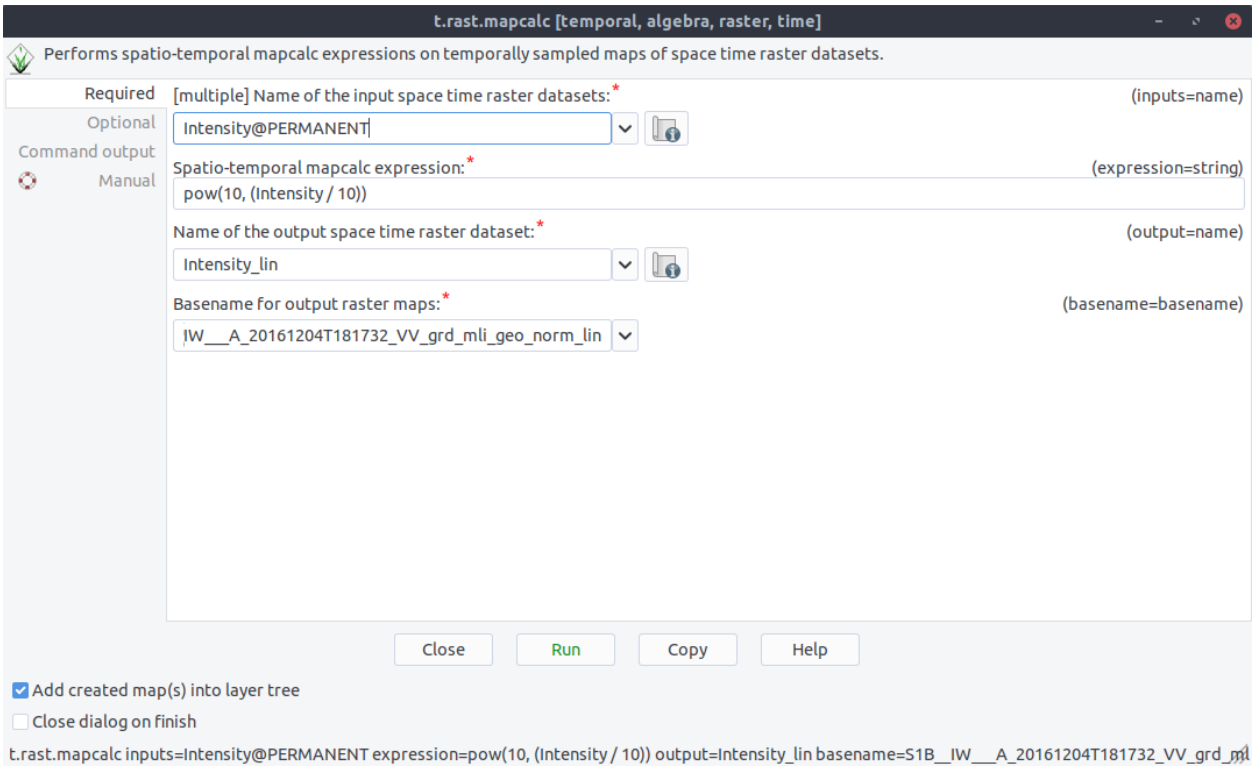
3.2.7 Spatio-temporal Calculation with Sentinel Data

In the previous step we create a spatio-temporal dataset in unit dB. Now, we will convert the dB units to linear unit with `t.raster.mapcalc`. To do this we launch the mapcalc GUI with:

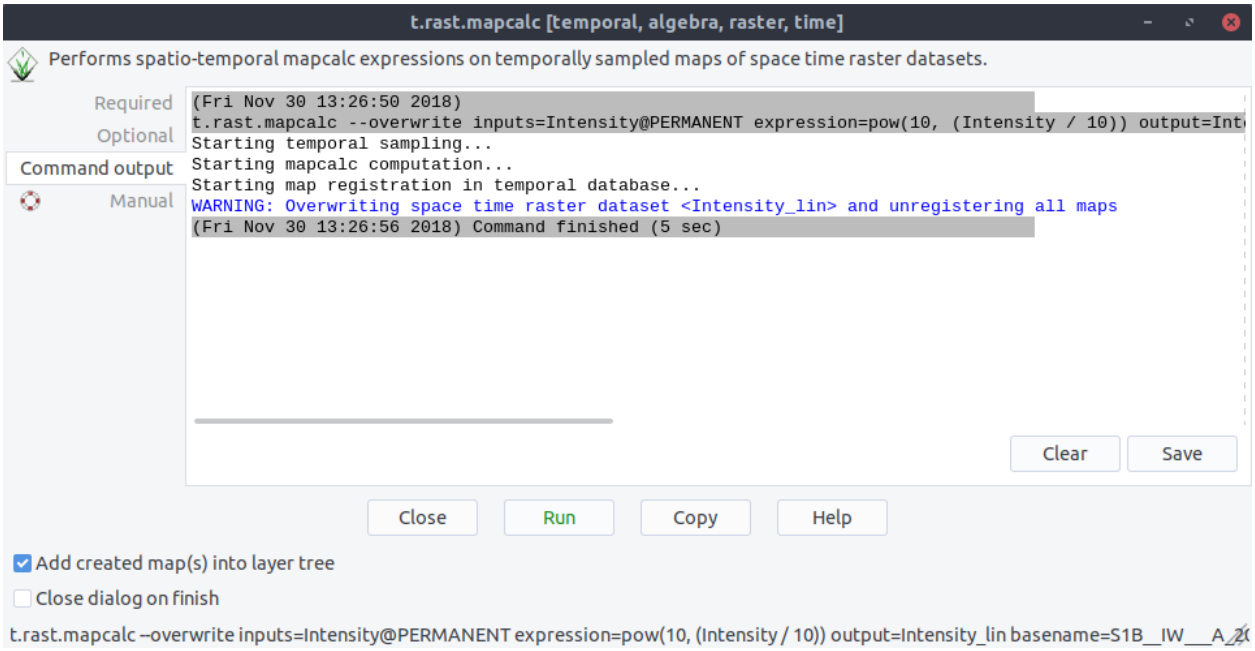
```
$ t.rast.mapcalc
```

We choose the space time raster dataset that we created in the previous step *Intensity*. To convert the unit [dB] in [linear] we use the formula:

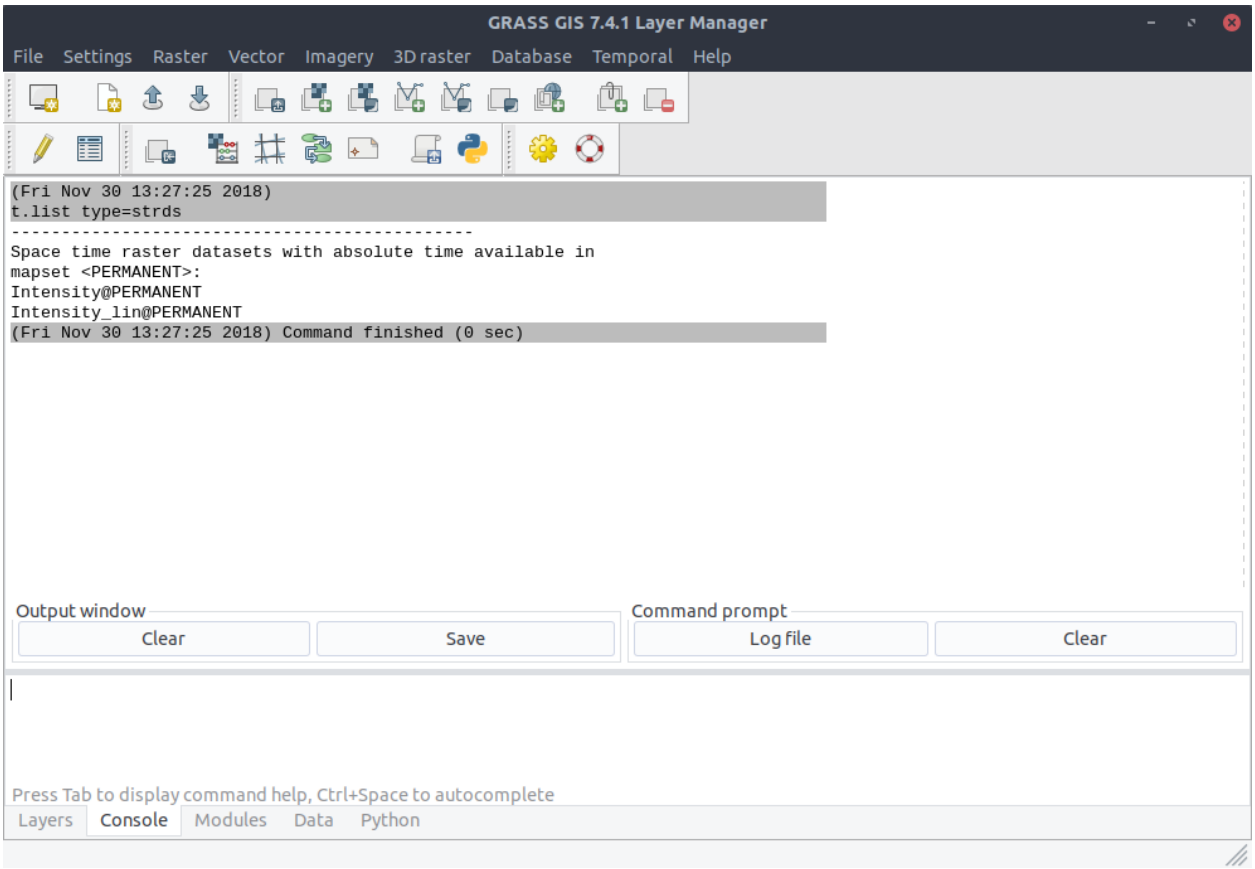
```
pow(10, (dB/10))
```



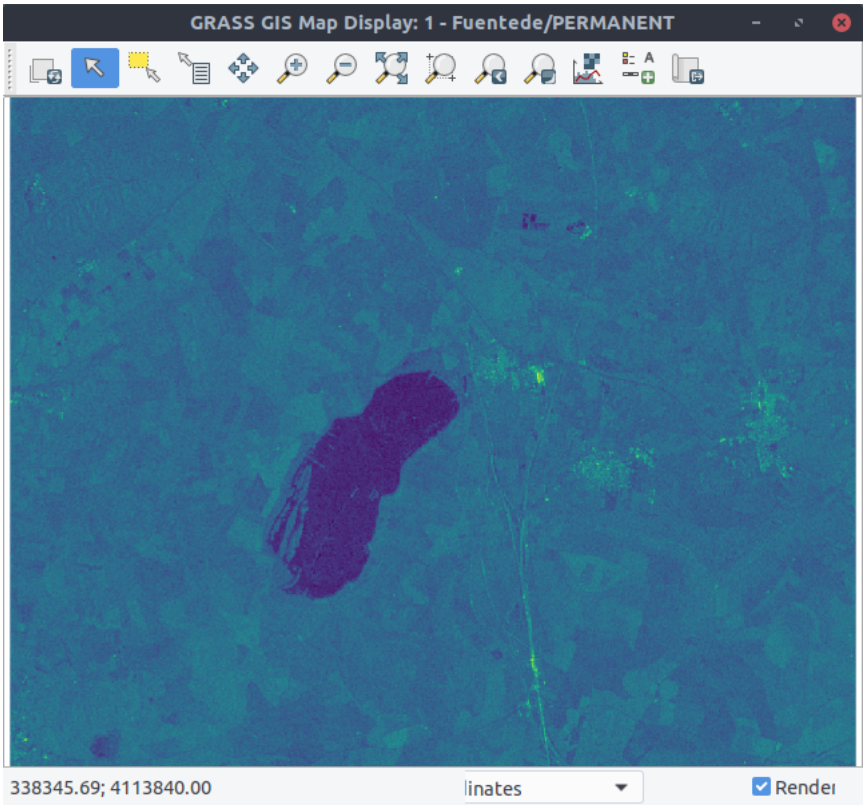
Now, we will click `Run`:



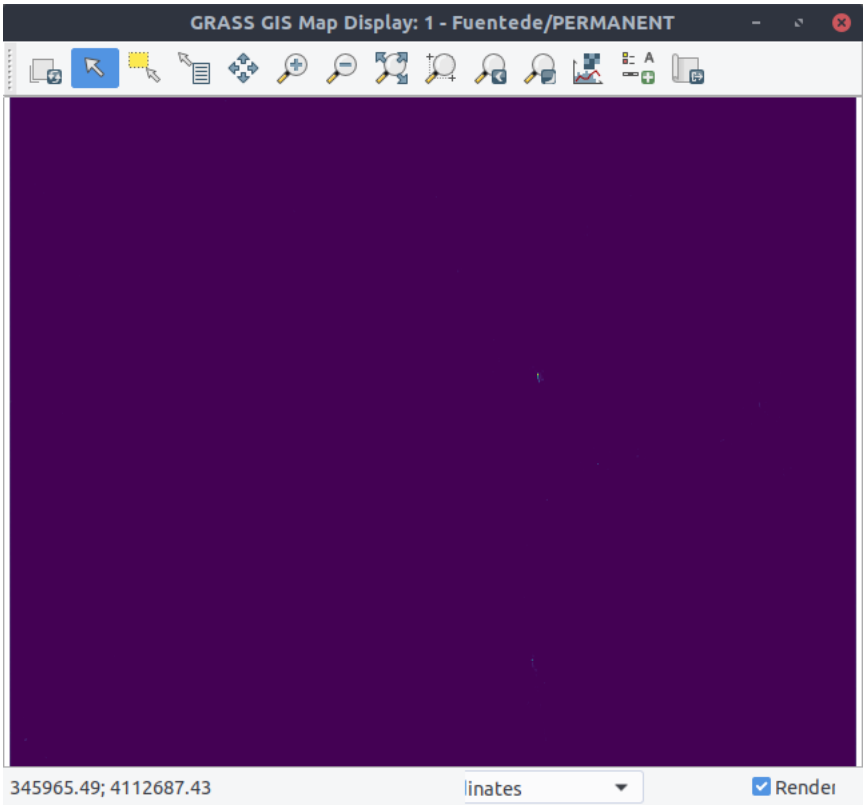
If we now list all spatio-temporal dataset we see two entries: The one for `Intensity` and the another one `Intensity_lin` that we created in with `t.raster.mapcalc`:



We can also import the dataset into the display. Here is the dataset in unit [dB]:



And here is the dataset in unit [linear]:

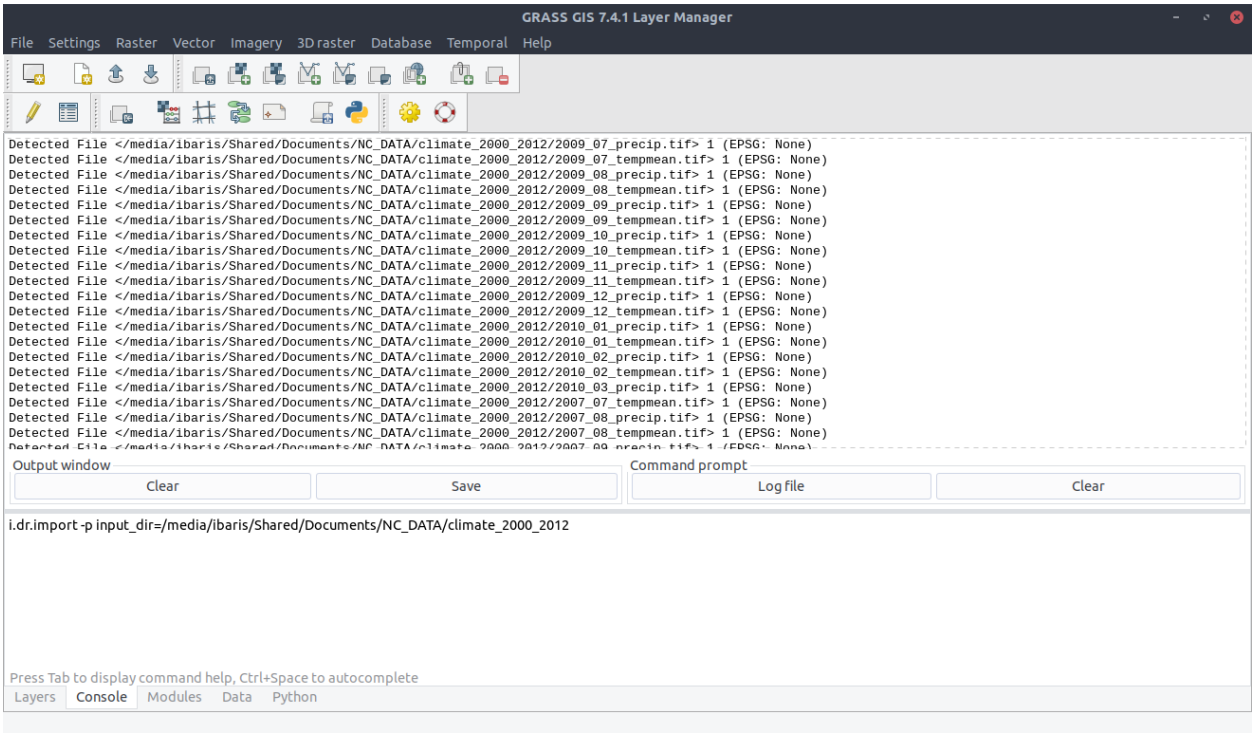


3.2.8 Import / Export Raster Files

After downloading the data (See section Data), you can list all available raster files within the directory *climate_2000_2012*, by using the module `i.dr.import`:

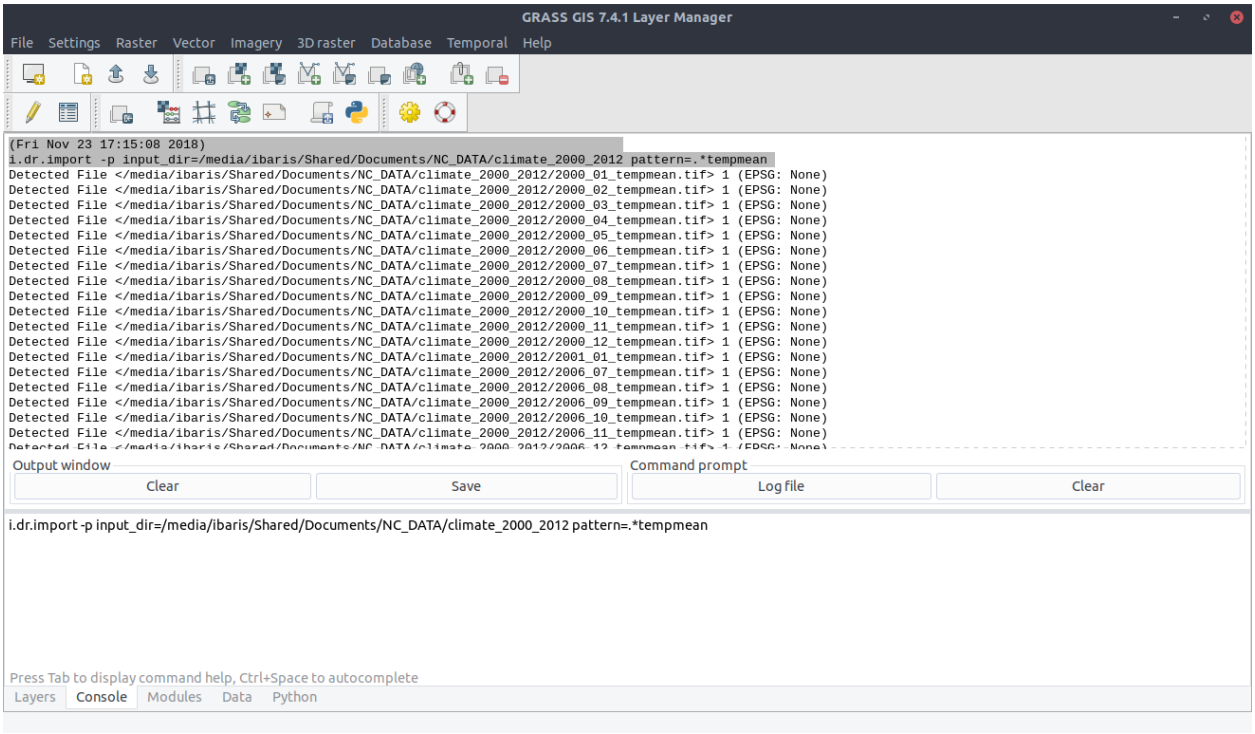
```
$ i.dr.input -p input_dir=/media/ibaris/Shared/Documents/NC_DATA/climate_2000_2012
```

Now you can see all files:



To show the `pattern` parameter of the module `i.dr.import` we want to consider only files that has the string *tempmean* in their filenames:

```
$ i.dr.input -p input_dir=/media/ibaris/Shared/Documents/NC_DATA/climate_2000_2012␣
↪pattern=.*tempmean
```

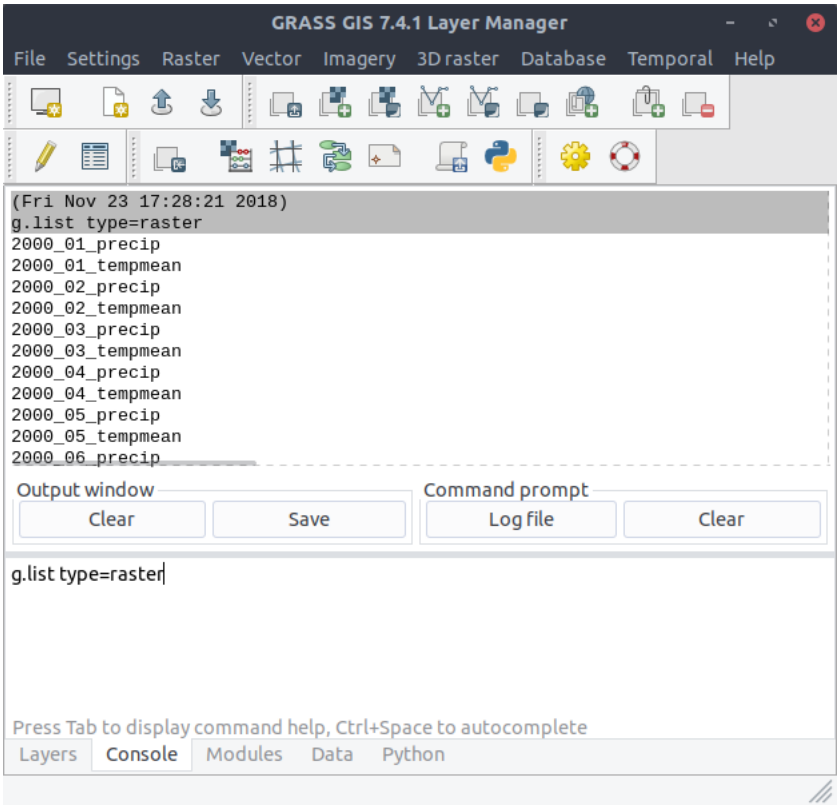


With the command:

```
$ i.dr.input input_dir=/media/ibaris/Shared/Documents/NC_DATA/climate_2000_2012
```

you can import all raster files located in */media/ibaris/Shared/Documents/NC_DATA/climate_2000_2012*. To be sure you can use the command:

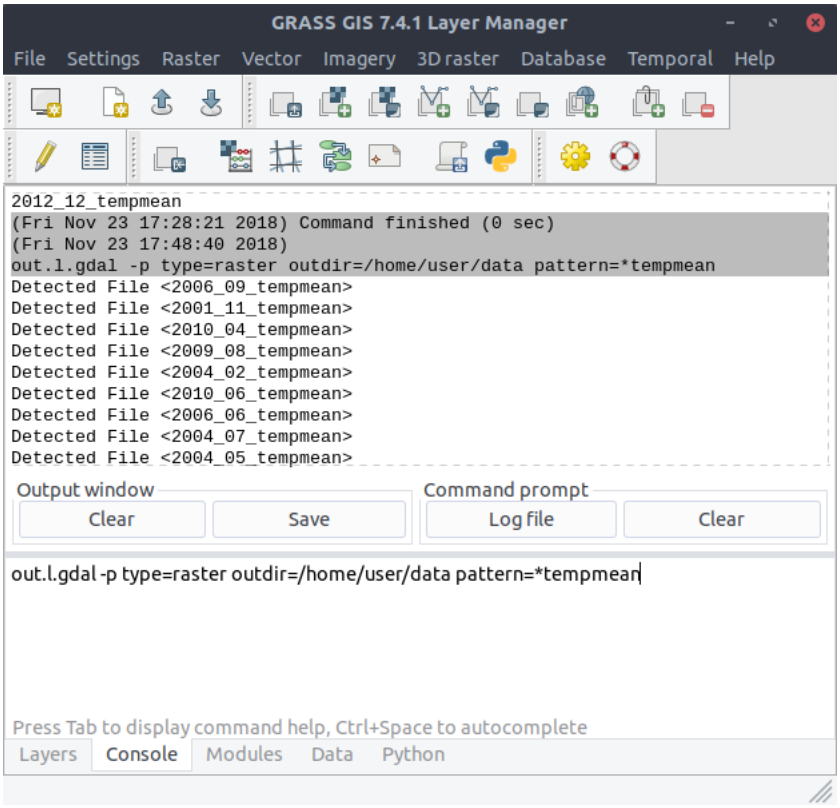
```
$ g.list type=raster
```



You could export all files with the command `out.l.gdal` like:

```
$ out.l.gdal type=raster outdir=/media/ibaris/Shared/Documents/NC_DATA/climate_2000_
↪2012
```

With the *flag: -p* you can see the files that will be exported:



3.2.9 Spatio-temporal data handling in General

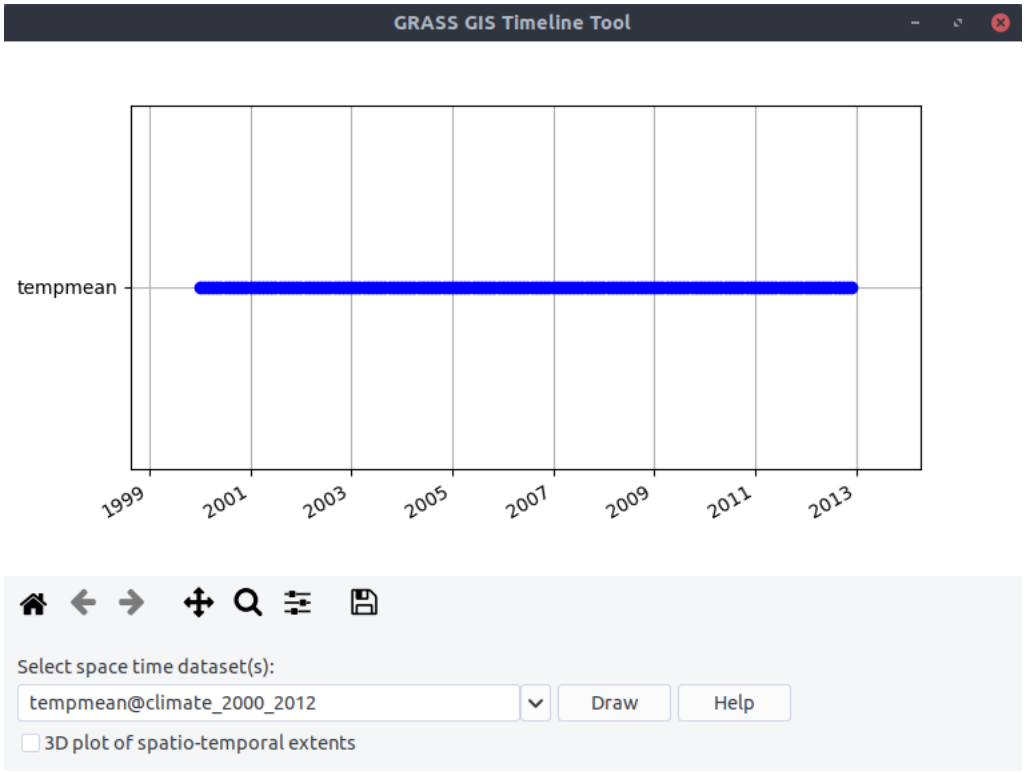
To better handle the long time series of maps, we create temporal datasets which serve as containers for the time series and we will further manipulate them instead of individual maps. Usually, we create empty datasets of type strds (space-time raster dataset) and after that we register the raster files into the strds. With `t.c.register` we can combine these two steps and with the *flag -m* we will visualize the temporal extents of the dataset (Note, that we use absolute time.):

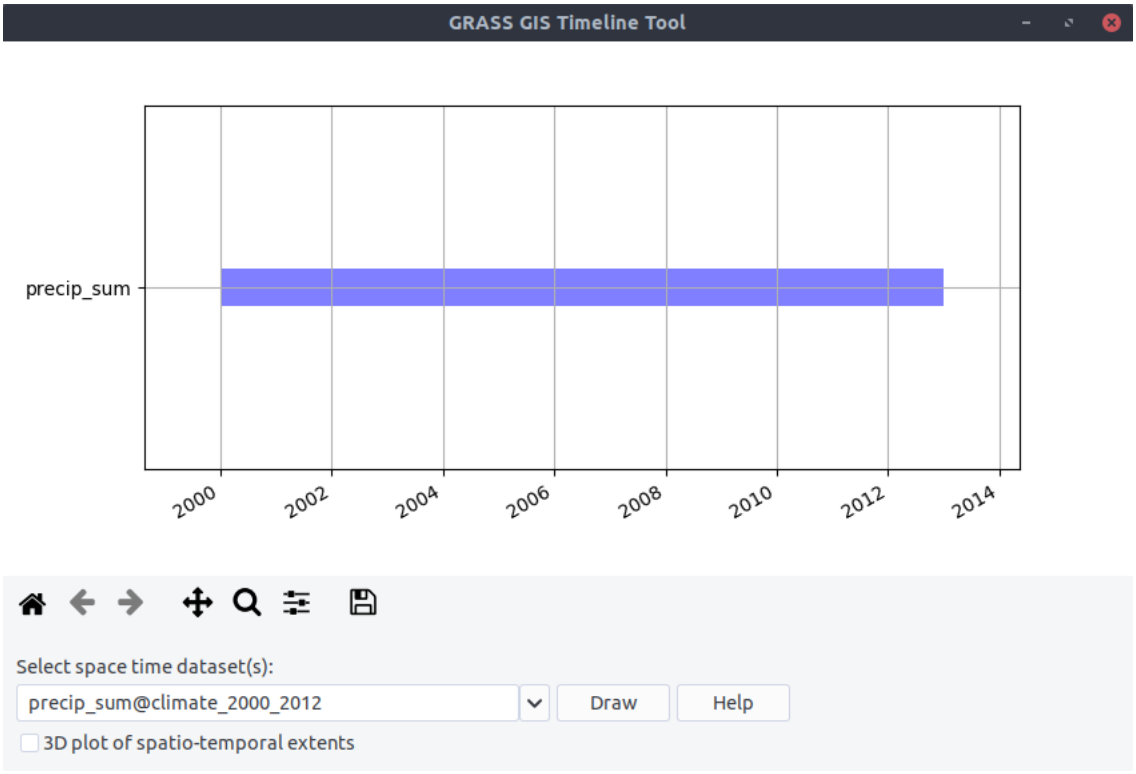
```
$ t.c.register -m -t output=tempmean temporaltype=absolute title="Average temperature"
  description="Monthly temperature average in NC [deg C]" pattern="*tempmean"␣
↪start=2000-01-01
  increment="1 months"
```

For the preciptation dataset:

```
$ t.c.register -m -t output=precip_sum title="Preciptation"
  description="Monthly precipitation sums in NC [mm]" pattern="*precip" start=2000-01-
↪01
  increment="1 months" semantictype=sum
```

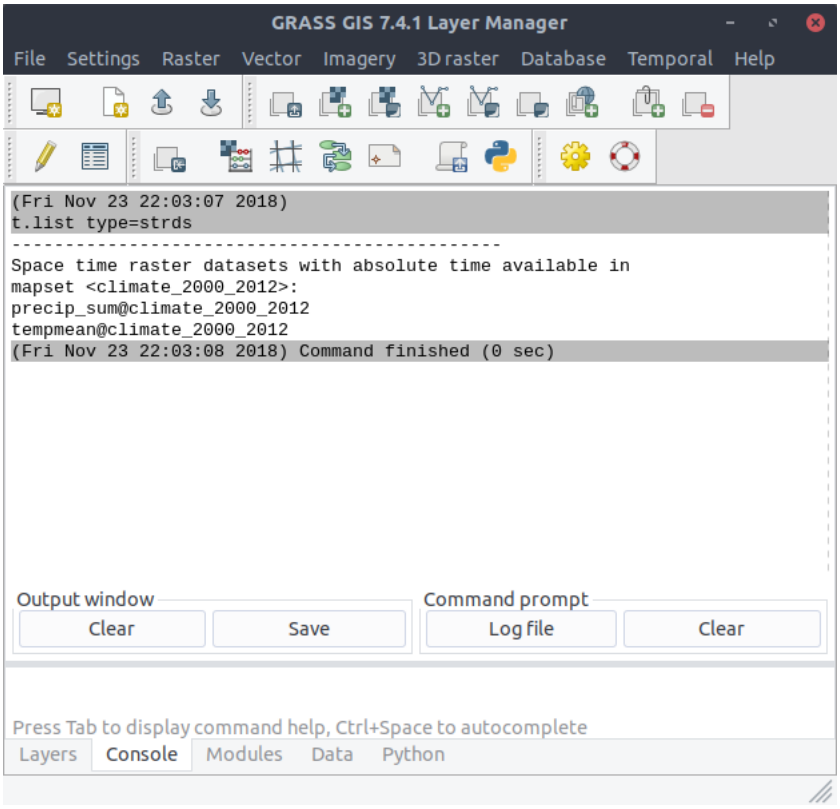
Look at the temporal extents:





With `t.list type=strds` we can see the new created datasets:

```
$ t.list type=strds
```

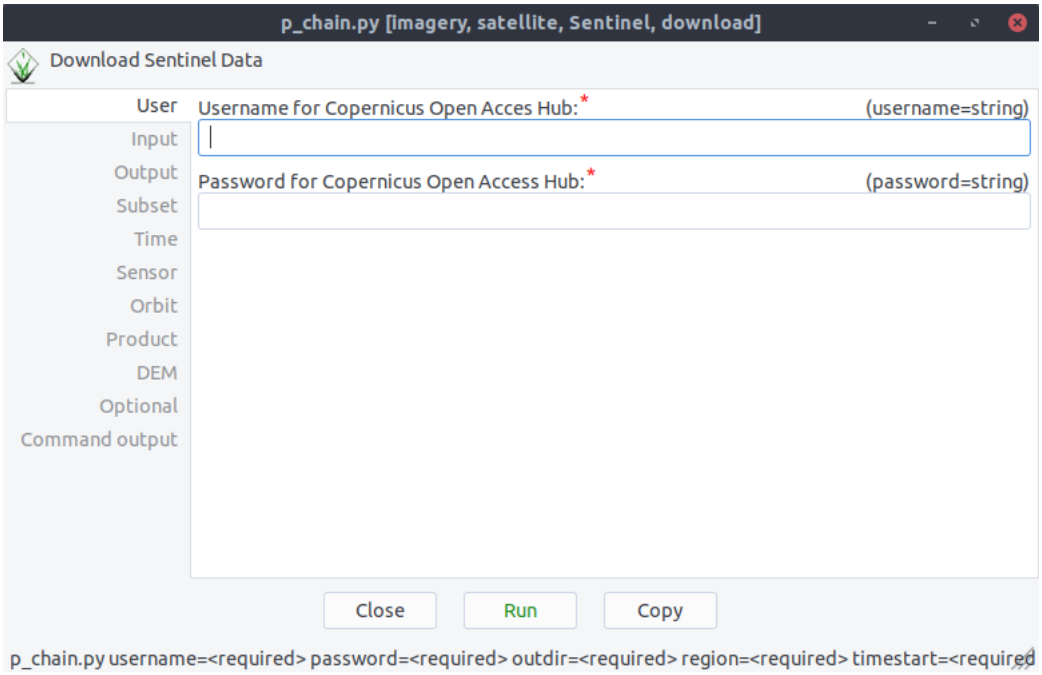


3.2.10 Download, Geocode and Import Chain

It is also possible to run a processing chain with `p.chain`:

```
$ p.chain
```

Now you can input all the same things described in the single modules `ds1.download` and `pr.geocode`


p_chain.py [imagery, satellite, Sentinel, download]
Download Sentinel Data
User
Input
Output
Subset
Time
Sensor
Orbit
Product
DEM
Optional
Command output
Username for Copernicus Open Acces Hub:*
(username=string)
Password for Copernicus Open Access Hub:*
(password=string)
Close
Run
Copy
p_chain.py username=<required> password=<required> outdir=<required> region=<required> timestart=<required

CHAPTER 4

Technical documentation

Here are the technical documentation which contains all classes and functions.

Contents:

4.1 Import Scripts

Import Scripts from a package to GRASS GIS.

This class will copy any suitable python file like 'i_dr_import.py' into the GRASS script folder, without the '.py' extension and changes the name to 'i.dr.import'. This class will exclude such files like '__init__.py' or 'setup.py'. For more exclusions the parameter *exclusion* can be used.

class gscpy.i_script.**Grassify**(*input_dir*, *export_path=None*, *pattern=None*, *exclusion=None*)

Import Scripts from a package to GRASS GIS.

This class will copy any suitable python file like 'i_dr_import.py' into the GRASS script folder without the '.py' extension and changes the name to 'i.dr.import'. This class will exclude such files like '__init__.py' or 'setup.py'. For more exclusions the parameter *exclusion* can be used.

Parameters

- **input_dir** (*str*) – Directory of python files
- **export_path** (*str, optional*) – Script directory of GRASS GIS (automatically in Linux systems).
- **pattern** (*str, optional*) – The pattern of file names.
- **exclusion** (*str*) – Which files or pattern should be excluded?.

extension

list – A list which contains all supported GRASS GIS candidates.

exclusion

str

import_path
str – Dir parameter.

export_path
str

filter_p
str – Combines pattern and extension.

files
list – All detected files.

copy(*replace=False*)
Copy files.

print_products()
Print all detected files.

Examples

The general usage is

```
$ i.script [-r-p] input_dir=string [pattern=string] [exclusion=string] [export_
↪path=string] [--verbose] [--quiet]
```

Import all suitable python files from a directory into the GRASS script folder

```
$ i.script input_dir=/home/user/package
```

Import all suitable python files from a directory into the GRASS script folder and exclude all files that include the string 'test'

```
$ i.script input_dir=/home/user/package exclude=test.*
```

Note: This class copies all files and replaces all '_' with '.'.

Notes

Flags:

- r : Overwrite file if it is existent **(be careful padawan!)**
- p : Print the detected files and exit.

4.2 Database Related Modules

In this section, modules are listed which relate to GRASS GIS databases. The modules listed below can create databases and mapsets.

The modules work with GRASS GIS versions `['grass70', 'grass71', 'grass72', 'grass73', 'grass74']`. However, this can easily be extended. To extend the versions, add new GRASS GIS versions to `self.candidates` in `gscpy.g_db.g_c_database`.

The name of the modules was chosen to ensure conformity with the GRASS GIS conventions. The addition c was added to module `gscpy.g_db.g_c_mapset` to signalize that the already existing module `g.mapset` is passed with the flag `-c` (*create*).

class gscpy.g_db.g_database.**Database**(*db_dir*, *db_name*, *t_srs=None*, *t_srs_file=None*, *launch=False*)

Create a GRASS GIS Database.

Create a new location, including it's default PERMANENT mapset, with or without entering the new location.

Parameters

- **db_dir** (*str*) – Location of GRASS GIS database
- **db_name** (*str*) – Name of the database.
- **t_srs** (*int, optional*) – A EPSG Code for georeferencing purposes.
- **t_srs_file** (*str, optional*) – If t_srs is not used, a georeferenced file can be here uploaded.
- **launch** (*bool, optional*) – If True, GRASS GIS will start with the new created mapset.

db_dir
str

db_name
str

t_srs
str or NoneType

t_srs_file
str or NoneType

launch
bool

create_database()
Create a GRASS GIS Database.

Examples

The general usage is

```
$ g.database [-l] db_dir=string db_name=string [t_srs=integer] [t_srs_
↪file=string] [--verbose] [--quiet]
```

Create a new location, including it's default PERMANENT mapset, without entering the new location using a EPSG code:

```
$ g.database db_dir=/home/user/grassdata db_name=germany t_srs=32630
```

Create a new location, including it's default PERMANENT mapset, without entering the new location using a georeferenced raster file:

```
$ g.database db_dir=/home/user/grassdata db_name=germany t_srs_file=myFile.tiff
```

Create new mapset within the new location and launch GRASS GIS within that mapset

```
$ g.database -l db_dir=/home/user/grassdata db_name=germany t_srs=32630
```

Notes

It is mandatory that t_srs OR t_srs_file is set.

This class trys to find *['grass70', 'grass71', 'grass72', 'grass73', 'grass74']* commands. This list can easily be extended for other versions of GRASS GIS.

Flags:

- l : Launch mapset with GRASS GIS.

create_database()
Create a GRASS GIS Database.

Returns

Return type None

class gscpy.g_db.g_c_mapset.**Mapset**(*mapset*, *dbase=None*, *location=None*)
Create a mapset in a GRASS GIS Database if it is not existent. This will changes the current working MAPSET, LOCATION, or GISDBASE. This is a fairly radical action to run mid-session, take care when running the GUI at the same time.

In GRASS GIS there is a similar function (*g.mapset*). This function shortens the flags and creates directly a new mapset if it is not existent.

Parameters

- **mapset** (*str, optional*) – Name of mapset.
- **dbase** (*str, optional*) – Location of GRASS GIS database
- **mapset** – Name of the mapset that will be created.

mapset
str

dbase
str

location
str

create_mapset()
Create a mapset in a GRASS GIS Database if it is not existent.

Examples

The general usage is

```
$ g.c.mapset [] mapset=string [dbase=string] [location=string] [--verbose] [--
↪quiet]
```

Creation of a mapset within a GRASS GIS session

```
$ g.c.mapset mapset=Goettingen
```

Creation of a mapset within another GRASS GIS database

```
$ g.c.mapset mapset=Goettingen dbase=/home/user/grassdata/germany
```

By default, the shell continues to use the history for the old mapset. To change this behaviour the history can be switched to record in the new mapset's history file as follows:

```
$ g.c.mapset mapset=Goettingen
history -w
history -r /"$GISDBASE/$LOCATION/$MAPSET"/.bash_history
HISTFILE=/"$GISDBASE/$LOCATION/$MAPSET"/.bash_history
```

Notes

By default, the shell continues to use the history for the old mapset. To change this behaviour the history look at the examples.

create_mapset()
Create a mapset in a GRASS GIS Database if it is not existent.

Returns

Return type None

4.3 Download Related Modules

This module makes searching, downloading and retrieving metadata of Sentinel 1 satellite images, from the Copernicus Open Access Hub, easy.

class gscpy.ds1_download.ds1_download.**S1Download**(*username*, *password*, *region*, *timestart*, *timeend*, *outdir*, *producttype=None*, *polarisationmode=None*, *sensoroperationalmode=None*, *orbitnumber=None*, *orbitdirection=None*)
This module makes searching, downloading and retrieving metadata of Sentinel-1 satellite images from the Copernicus Open Access Hub easy.

Parameters

- **username** (*str*) – Username for Copernicus Open Access Hub
- **password** (*str*) – Password for Copernicus Open Access Hub
- **region** (*str*) – A geojson file.
- **timestart** (*str*) – Start time like "YYYY-MM-DD"
- **timeend** (*str*) – End time like "YYYY-MM-DD".
- **outdir** (*str*) – Output directory.
- **producttype** (*{'SLC', 'GRD', 'OCN', ''RAW}*) – Product type. If None, all types will be recognized.
- **tuple or str** (*polarisationmode*) – A combination of V and H like ('VH', 'HV') or simple 'VH'.
- **sensoroperationalmode** (*{'SM', 'IW', 'EW', 'WV'}*) – Sensor operational mode. If None, all types will be recognized.

- **orbitnumber** (*int*) – Orbit number
- **orbitdirection** (*{DESCENDING, ASCENDING}*) – Orbit direction. If None, all types will be recognized.

api
object – Sentinelsat API object.

outdir
str

region
wkt – A geojson to WKT object.

kwargs
dict – Dictionary with setted attributes.

files
DataFrame – Pandas DataFrame with detected files.

download()
Download all files.

print_products()
Print all detected files.

Examples

The general usage is

```
$ ds1.download [-p] username=string password=string region=string␣
↪timestart=string timeend=string outdir=sting
[*attributes=string] [--verbose] [--quiet]
```

For *attributes the following parameters can be used

```
>>> ["producttype", "polarisationmode", "sensoroperationalmode", "orbitnumber",
↪"orbitdirection"]
```

Print all Sentinel 1 data with product type GRD between 2015-01-02 and 2015-01-12:: $ ds1.download -p username=USER password=PASSWORD region=myGEoJsOnFile.geojson timestart=2015-01-02 timeend=2015-01-12 outdir='home/usr/data' producttype=SLC

Download the last query

```
$ ds1.download username=USER password=PASSWORD region=myGEoJsOnFile.geojson␣
↪timestart=2015-01-02
timeend=2015-01-12 outdir='home/usr/data' producttype=SLC
```

Notes

Flags:

- p : Print the detected files and exit.

print_products()
Print all detected files.

Returns

Return type None

4.4 Pre-Processing Related Modules

Wrapper function for geocoding SAR images using pyroSAR:

The purpose of the pyroSAR package is to provide a complete solution for the scalable organization and processing of SAR satellite data:

* Reading of data from various past and present satellite missions
* Handling of acquisition metadata
* User-friendly access to processing utilities in SNAP and GAMMA Remote Sensing software
* Formatting of the preprocessed data for further analysis

class gscpy.pr_geocode.pr_geocode.**Geocode**(*input_dir*, *outdir*, *pattern=None*, *t_srs=None*, *t_srs_from_file=None*, *resolution_value=20*, *polarizations='all'*, *shapefile=None*, *scaling='dB'*, *geocoding_type='Range-Doppler'*, *removeS1BoderNoise=True*, *offset=None*, *external_dem_file=None*, *external_dem_nan=None*, *externalDEMApplyEGM=True*, *basename_extensions=None*, *test=False*, *verbose=False*)

Wrapper function for geocoding SAR images using pyroSAR.

Parameters

- **input_dir** (*str*) – Directory where the Sentinel-Data is.
- **outdir** (*str*) – The directory to write the final files to.
- **t_srs** (*int, str or osr.SpatialReference*) – A target geographic reference system in WKT, EPSG, PROJ4 or OPENGIS format. See function `spatialist.auxil.crsConvert()` for details. Default: 4326.
- **resolution_value** (*int or float, optional*) – The target resolution in meters. Default is 20
- **polarizations** (*list or {'VV', 'HH', 'VH', 'HV', 'all'}, optional*) – The polarizations to be processed; can be a string for a single polarization e.g. 'VV' or a list of several polarizations e.g. ['VV', 'VH']. Default is 'all'.
- **shapefile** (str or `Vector`, optional) – A vector geometry for subsetting the SAR scene to a test site. Default is None.
- **scaling** (*{'dB', 'db', 'linear'}, optional*) – Should the output be in linear or decibel scaling? Default is 'dB'.
- **geocoding_type** (*{'Range-Doppler', 'SAR simulation cross correlation'}, optional*) – The type of geocoding applied; can be either 'Range-Doppler' (default) or 'SAR simulation cross correlation'
- **removeS1BoderNoise** (*bool, optional*) – Enables removal of S1 GRD border noise (default).

- **offset** (*tuple, optional*) – A tuple defining offsets for left, right, top and bottom in pixels, e.g. (100, 100, 0, 0); this variable is overridden if a shapefile is defined. Default is None.
- **external_dem_file** (*str or None, optional*) – The absolute path to an external DEM file. Default is None.
- **external_dem_nan** (*int, float or None, optional*) – The no data value of the external DEM. If not specified (default) the function will try to read it from the specified external DEM.
- **externalDEMApplyEGM** (*bool, optional*) – Apply Earth Gravitational Model to external DEM? Default is True.
- **basename_extensions** (*list of str*) – names of additional parameters to append to the basename, e.g. ['orbitNumber_rel']
- **test** (*bool, optional*) – If set to True the workflow xml file is only written and not executed. Default is False.

files
list – All detected files.

geocode()
Start the geocoding process.

import_products(*pattern=None*, *mapset=None*, *dbase=None*, *location=None*, *flags=None*)
Import detected files.

print_products()
Print all detected files.

Examples

The general usage is

```
$ pr.geocode [-e -i -r -l -c -p -t -b] input_dir=string outdir=string␣
↪[pattern=string] [t_srs=string] [t_srs_from_file=string]
    [resolution_value=integer] [polarizations=string] [shapefile=string]␣
↪[scaling=string]
    [geocoding_type=string] [offset=string] [external_dem_file=string] [external_
↪dem_nan=integer]
    [basename_extensions=string] [mapset=string] [dbase=string] [location=string]␣
↪[--verbose] [--quiet]
```

Import Sentinel 1A files geocode and import them in current mapset and reproject it

```
$ pr.geocode -r input_dir=/home/user/data outdir=/home/user/data/processed t_
↪srs=43265
```

Import Sentinel 1A files geocode and import them in a new mapset within another GRASS GIS database

```
$ pr.geocode -r input_dir=/home/user/data outdir=/home/user/data/processed t_
↪srs=43265 mapset=Goettingen dbase=/home/user/grassdata/germany
```

Note: If only one polarization is selected the results are directly written to GeoTiff. Otherwise the results are first written to a folder containing ENVI files and then transformed to GeoTiff files (one for each polarization).

If GeoTiff would directly be selected as output format for multiple polarizations then a multilayer GeoTiff is written by SNAP which is an unfavorable format

Notes

Flags:

- e : Apply Earth Gravitational Model to external DEM.
- i : Import processed files in a mapset.
- r : Reproject raster data (using r.import if needed).
- l : Link raster data instead of importing.
- c : Create a new mapset.
- p : Print the detected files and exit.
- t : Write only the workflow in xml file
- b : Enables removal of S1 GRD border noise.

geocode()
Start the geocode process.

Returns

Return type None

import_products(*pattern=None*, *mapset=None*, *dbase=None*, *location=None*, *flags=None*)
Import all detected files in a mapset.

Parameters

- **pattern** (*str, optional*) – The pattern of file names. If not specified all files with selected extension will be imported.
- **mapset** (*str, optional*) – Name of mapset.
- **dbase** (*str, optional*) – Location of GRASS GIS database
- **mapset** – Name of the mapset that will be created.
- **flags** (*str*) – Available flags are '-c-r-l'

Returns

Return type None

print_products()
Print all detected files.

Returns

Return type None

4.5 Import Related Modules

These modules are for importing data. Unlike the existing modules, they can import all files in a directory by considering a certain pattern. Moreover, it is possible to import these data in different `mapsets`. In addition, the module `i_fr_import` can import `pyroSAR` datasets in a directory based on their metadata.

The name of the modules was chosen to ensure conformity with the GRASS GIS conventions. The addition `r` stands for *raster* where the addition `d` and `f` stand for *directory* and *finder* respectively.

Note, it is important for the parameter 'pattern' that the asterisk('*') contains a dot (see examples).

class gscpy.i_import.i_dr_import.**DirImport**(*input_dir*, *pattern=None*, *extension=None*)

Import data into a mapset from a file with considering certain patterns.

Parameters

- **input_dir** (*str*) – The directory where the files are located.
- **pattern** (*str, optional*) – The pattern of file names. If not specified all files with selected extension will be imported.
- **extension** (*{'ENVI', 'GEOTIFF'}, optional*) – Which extensions should be recognized? Default is 'GEOTIFF'

input_dir
str

extension
str

filter_p
str – Combines pattern and extension.

files
list – All detected files.

import_products(*reproject=False*, *link=False*)
Import detected files.

create_mapset(*mapset*, *dbase=None*, *location=None*)
Create a new mapset.

print_products()
Print all detected files.

Examples

The general usage is

```
$ i.dr.import [-r -l -c -p] input_dir=string [pattern=string] [extension=string]␣
↪[mapset=string] [dbase=string] [location=string] [--verbose] [--quiet]
```

Import files that starts with 'S1' from a directory in current mapset and reproject it

```
$ i.dr.import -r input_dir=/home/user/data pattern=S1.*
```

Import files that starts with 'S1' from a directory in a new mapset and reproject it

```
$ i.dr.import -c -r input_dir=/home/user/data pattern=S1.* mapset=Goettingen
```

Notes

Note, it is important for the parameter *pattern* that the asterisk('*') contains a dot (see examples).

Flags:

- r : Reproject raster data (using r.import if needed).
- l : Link raster data instead of importing.
- c : Create a new mapset.
- p : Print the detected files and exit.

create_mapset(*mapset*, *dbase=None*, *location=None*)
Create a new mapset calling the module *g.c.mapset*.

Parameters

- **mapset** (*str, optional*) – Name of mapset.
- **dbase** (*str, optional*) – Location of GRASS GIS database
- **mapset** – Name of the mapset that will be created.

Returns

Return type None

import_products(*reproject=False*, *link=False*)
Import detected files.

Parameters

- **reproject** (*bool*) – Reproject raster data (using r.import if needed).
- **link** (*bool*) – Link raster data instead of importing.

Returns

Return type None

print_products()
Print all detected files.

Returns

Return type None

class gscpy.i_import.i_fr_import.**FinderImport**(*input_dir*, *recursive=False*, *sensor=None*, *projection=None*, *orbit=None*, *polarization=None*, *acquisition_mode=None*, *start=None*, *stop=None*, *product=None*, *spacing=None*, *sample=None*, *lines=None*)
Import pyroSAR dataset in a directory based on their metadata.

Parameters

- **input_dir** (*str*) – The directory where the files are located.
- **recursive** (*bool, optional*) – Recursive search. Default is False.
- **sensor** (*str or tuple, optional*) – Sensor.
- **projection** (*str or tuple, optional*) – Projection.
- **orbit** (*str or tuple, optional*) – Orbit.

- **polarization** (*str or tuple, optional*) – Polarization.
- **acquisition_mode** (*str or tuple, optional*) – Acquisition_mode.
- **start** (*str or tuple, optional*) – Start.
- **stop** (*str or tuple, optional*) – Stop.
- **product** (*str or tuple, optional*) – Product.
- **spacing** (*str or tuple, optional*) – Spacing.
- **sample** (*str or tuple, optional*) – Sample.
- **lines** (*str or tuple, optional*) – Lines.

input_dir
str

recursive
bool

kwargs
dict – Selected attributs (sensor, polarization etc.) in a dictionary.

files
list – All detected files.

find_products()
Find all files that matches the input attributes.

import_products(*reproject=False*, *link=False*)
Import detected files.

create_mapset(*mapset*, *dbase=None*, *location=None*)
Create a new mapset.

print_products()
Print all detected files.

Examples

The general usage is

```
$ i.fr.import [-r -l -c -p -e] input_dir=string [*attributes=string]␣
↪[mapset=string] [dbase=string] [location=string] [--verbose] [--quiet]
```

For *attributes the following parameters can be used

```
>>> ['sensor', 'projection', 'orbit', 'polarization', 'acquisition_mode', 'start',
↪ 'stop', 'product', 'spacing', 'samples', 'lines']
```

Import Sentinel 1A files with polarization VV and VH from a directory in current mapset and reproject it

```
$ i.fr.import -r input_dir=/home/user/data sensor=SA1 polarization=VV, VH
```

Import Sentinel 1A and 1B files with polarization VV from a directory in a new mapset and reproject it

```
$ i.fr.import -c -r input_dir=/home/user/data sensor=S1A, S1B polarization=VV␣
↪mapset=Goettingen
```

Notes

Flags:

- r : Reproject raster data (using r.import if needed).
- l : Link raster data instead of importing.
- c : Create a new mapset.
- p : Print the detected files and exit.
- e : Recursive search.

create_mapset(*mapset*, *dbase=None*, *location=None*)
Create a new mapset calling the module *g.c.mapset*.

Parameters

- **mapset** (*str, optional*) – Name of mapset.
- **dbase** (*str, optional*) – Location of GRASS GIS database
- **mapset** – Name of the mapset that will be created.

Returns

Return type None

find_products()
Find all files that matches the input attributes. :returns: :rtype: list

import_products(*reproject=False*, *link=False*)
Import detected files.

Parameters

- **reproject** (*bool*) – Reproject raster data (using r.import if needed).
- **link** (*bool*) – Link raster data instead of importing.

Returns

Return type None

print_products()
Print all detected files.

Returns

Return type None

4.6 Export Related Modules

Exports GRASS raster maps from a selection into formats which GDAL supports.

class gscpy.out_l_export.out_l_gdal.**OutLGdal**(*type*, *outdir*, *pattern=None*, *exclude=None*, *mapset=None*, *region=None*, *output=None*, *createopt=None*, *metaopt=None*, *nodata=None*, *suffix=False*)
Exports GRASS raster maps from a selection into GDAL supported formats.

Parameters

- **type** (*{'raster', 'raster_3d', 'vector', 'label', 'region', 'group', 'all'}*) – Data type(s).
- **outdir** (*str*) – The directory to write the final files to.
- **pattern** (*str, optional*) – The pattern of file names.
- **exclude** (*str, optional*) – Which files or patterns should be excluded?
- **mapset** (*str, optional*) – Name of mapset to list (default: current search path); '*' for all mapsets in location.
- **region** (*str, optional*) – Name of saved region for map search (default: not restricted); '*' for default region
- **output** (*str, optional*) – Suffix or prefix to the filename (see parameter suffix.
- **createopt** (*str, optional*) – Creation option(s) to pass to the output format driver.
- **metaopt** (*str, optional*) – Metadata key(s) and value(s) to include
- **nodata** (*float, optional*) – Assign a specified nodata value to output bands
- **suffix** (*bool, optional*) – If True, the parameter output is used as suffix. If False (Default) it will be used as prefix.

lkwargs
dict – Attributes for g.list module.

ekwargs
dict – Attributes for out.gdal module.

list_files(*i=False*, *r=False*, *e=False*, *t=False*, *m=False*, *f=False*)

List ass detected files. Note, that only flag i works!

export_files(files) Export all detected files.

print_products()
Print all detected files.

Examples

The general usage is

```
$ out.l.gdal [-i-x-p] type=string outdir=string [pattern=string] [exclude=string]␣
↪[mapset=string] [region=string]
[output=string] [createopt=string] [nodata=float] [--verbose] [--quiet]
```

List raster files that ends with 'tempmean' from current mapset to a directory

```
$ out.l.gdal -p type=raster outdir=/home/user/data pattern=*tempmean
```

Export raster files that ends with 'tempmean' from current mapset to a directory

```
$ out.l.gdal type=raster outdir=/home/user/data pattern=*tempmean
```

Export raster files that ends with 'tempmean' from current mapset to a directory and add the suffix '_export' to them:

```
$ out.l.gdal -x type=raster outdir=/home/user/data pattern=*tempmean output=_
↪export
```

Export all raster files

```
$ out.l.gdal type=raster outdir=/home/user/data
```

Notes

Note, it is important for the parameter *pattern* that the asterisk('*') contains a dot (see examples).

Flags:

- x : Consider output name as suffix. Otherwise it is considered as prefix.
- i : Ignore case.
- p : Print the detected files and exit.

print_products (*files*)
Print all detected files.

Returns

Return type None

4.7 Space Time Related Modules

Spatio-temporal data handling and visualization in GRASS GIS.

class gscpy.t_c_register.t_c_register.**CRegister**(*output*, *title*, *description*, *start*, *type='raster'*, *semantictype='mean'*, *end=None*, *temporaltype='absolute'*, *separator='comma'*, *pattern=None*, *exclude=None*, *mapset=None*, *region=None*, *unit=None*, *increment=None*)

Create and register a space time dataset.

Parameters

- **output** (*str*) – Name of the output space time dataset.
- **title** (*str*) – Title of the new space time dataset.
- **description** (*str*) – Description of the new space time dataset
- **semantictype** (*{"min", "max", "sum", "mean"}*) – Semantic type of the space time dataset. Default is mean.
- **type** (*{'raster', 'raster_3d', 'vector', 'label', 'region', 'group', 'all'}*) – Data type(s). Default is raster.
- **start** (*str*) – Valid start date and time of the first map. Format for absolute time: "yyyy-mm-dd HH:MM:SS +HHMM", relative time is of type integer.
- **end** (*str*) – Valid end date and time of last map. Format for absolute time: "yyyy-mm-dd HH:MM:SS +HHMM", relative time is of type integer.

- **temporaltype** (*{"absolute", "relative"}*) – The temporal type of the space time datase. Default is absolute.
- **separator** (*{"pipe", "comma", "space", "tab", "newline"}*) – Field separator character of the input file. Default is comma.
- **unit** (*{"years", "months", "days", "hours", "minutes", "seconds"}*) – Time stamp unit. Unit must be set in case of relative timestamps.
- **increment** (*str*) – Time increment, works only in conjunction with start option. Time increment between maps for creation of valid time intervals (format for absolute time: NNN seconds, minutes, hours, days, weeks, months, years; format for relative time is of type integer: 5)
- **pattern** (*str, optional*) – The pattern of file names.
- **exclude** (*str, optional*) – Which files or patterns should be excluded?
- **mapset** (*str, optional*) – Name of mapset to list (default: current search path); '*' for all mapsets in location.
- **region** (*str, optional*) – Name of saved region for map search (default: not restricted); '*' for default region

lkwargs
dict – Attributes for g.list module.

ckwargs
dict – Attributes for t.create module.

rkwargs
dict – Attributes for t.register module.

cregister(*self*, *t=False*)
Create and register a space time dataset.

print_products()
Print all detected files.

list()
List Files for current space time dataset.

plot()
Visualize the temporal extents of the dataset.

Examples

The general usage is

```
$ t.c.register [-i-t-p-l-m] output=string title=string description=string␣
↪start=string [type=string]
[semantictype=string] [end=string] [temporaltype=string] [separator=string]␣
↪[pattern=string]
[exclude=float] [mapset=string] [region=string] [unit=string] [unit=increment] [--
↪verbose] [--quiet]
```

Create a mapset that named 'tempmean' and register all raster files that contains *tempmean*

```
$ t.c.register output=tempmean temporaltype=absolute title="Average temperature"
description="Monthly temperature average in NC [deg C]" pattern="*tempmean"␣
↪start=2000-01-01
increment="1 months"
```

Create a mapset that named 'tempmean' and register all raster files that contains *tempmean*. Show a plot after registration

```
$ t.c.register -m -t output=tempmean temporaltype=absolute title="Average␣
↪temperature"
description="Monthly temperature average in NC [deg C]" pattern="*tempmean"␣
↪start=2000-01-01
increment="1 months"
```

Create a mapset that named 'precip_sum' and register all raster files that contains *precip*. Show a plot after registration

```
$ t.c.register -m -t output=precip_sum title="Preciptation"
description="Monthly precipitation sums in NC [mm]" pattern="*precip" start=2000-
↪01-01
increment="1 months" semantictype=sum
```

Notes

Flags:

- i : Ignore case.
- t : Create an interval (start and end time) in case an increment and the start time are provided
- p : Print the detected files and exit.
- l : List files after registration.
- m : Plot files after registration.

cregister(*t=False*)

Create and register a space time dataset.

Parameters **t** (*bool*) – Create an interval (start and end time) in case an increment and the start time are provided.

Returns

Return type None

list()

List Files for current space time dataset.

Returns

Return type None

plot()

Visualize the temporal extents of the dataset.

Returns

Return type None

print_products()
Print all detected files.

Returns

Return type None

4.8 Indices and tables

- genindex
- modindex
- search

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

g

Index

K

L

M

O

P

R

S

T