GRASS GIS Development APIs

Lifting the fog on the different ways
to develop with and for GRASS

Moritz Lennert
Member of the GRASS GIS
Project Steering Comittee

Over 30 years of development

- A C-Project
- Shell scripts integrated very early
- Appearance of
  - New generations of developers
  - New languages
- Today different APIs exist
- **Aim of this presentation : clarification of objectives and use-cases for each API**

GRASS GIS = API

- GRASS GIS = toolkit following UNIX philosophy
- Each module can be considered a 'function' with its parameters
  - Module map output stored in GRASS database
  - Other types of output can either be stored in files or directly sent to standard output
  - Chaining of modules = programming

GRASS GIS = API

- Identify points lying in possible flood risk area :

```
ELEVATION_MAP=elevation
POINT_MAP=camping_grounds

g.region rast=$ELEVATION_MAP
r.watershed elevation=$ELEVATION_MAP threshold=10000
stream=raster_streams
r.to.vect input=raster_streams output=vector_streams
type=line
v.buffer input=vector_streams output=stream_buffers
distance=500
v.select a_input=$POINT_MAP b_input=stream_buffers
operator=within output=places_at_risk
```

GRASS GIS = API

- Memory handling (garbage collection) by system
- Individual error handling by each module
- Modules can be called through system calls from other languages = instant integration of GRASS GIS

Python

```
import subprocess
subprocess.call('r.watershed elevation=elevation
threshold=10000 stream=raster_streams', shell=True)
```

C

```
void main()
{
   system("r.watershed elevation=elevation
threshold=10000 stream=raster_streams");
}
```

GRASS GIS Python Scripting Library

- Module calls from Python not always easy to implement
  - Specific handling of shell expansion
  - Handling module output other than maps
- Python scripting library elaborated (mostly by Glynn Clements) to ease these calls
  - Wrapper functions around subprocess.Popen().
  - Specific functions for handling different types of IO :
    - Reading output line by line
    - Reading output as dictionary
    - Feeding info to a module via stdin
    - etc

GRASS GIS Python Scripting Library

```
>>> grass.run_command('g.region', flags='g')

n=228500
s=215000
w=630000
e=645000
nsres=10
ewres=10
rows=1350
cols=1500
cells=2025000

>>> grass.read_command('g.region', flags='g')

'n=228500\ns=215000\nw=630000\ne=645000\nnsres=10\newres=10\n
rows=1350\ncols=1500\ncells=2025000\n'

>>> grass.parse_command('g.region', flags='g')

{'rows': '1350', 'e': '645000', 'cells': '2025000', 'cols':
'1500', 'n': '228500', 's': '215000', 'w': '630000', 'ewres':
'10', 'nsres': '10'
```

GRASS GIS Python Scripting Library

- Makes calls to GRASS GIS modules very easy, including output handling
- Also implements a series of wrappers for often-used module calls. Ex :
  - `grass.create_location()`
  - `grass.list_pairs('vector', mapset='PERMANENT')`
  - `grass.db_select(sql = 'SELECT cat,CITY FROM myfirestations WHERE cat < 4')`
  - `grass.raster_what('elevation', [[640000, 228000]])`
  - `Etc`
- In GRASS 7, all scripts were rewritten from bash to Python using scripting library

http://grass.osgeo.org/grass71/manuals/libpython/script.html

PyGRASS

- GRASS GIS Python scripting library
  - wrapper around module calls
  - no low-level acces to GRASS GIS data and functions
  - not very pythonic
- Google Summer of Code project by Pietro Zambelli: PyGRASS
  - Two layers
    - replacement of scripting library's module call functions
    - lower-level access to GRASS GIS (via ctypes)
  - integrate philosophies of both GRASS GIS and Python => more pythonic

PyGRASS – Access to modules

```
>>> from grass.pygrass.modules import Module
>>> gregion=Module('g.region')
>>> gregion.description
'Manages the boundary definitions for the geographic region.'
>>> gregion.flags.g = True
>>> gregion.run()
n=318500
s=10500
[…]
rows=616
cols=1613
cells=993608
>>> gregion.inputs.raster = 'elevation'
>>> gregion.run()
n=228500
s=215000
[…]
rows=1350
cols=1500
cells=2025000
```

PyGRASS – Access to modules

- Shortcuts to module calls almost identical to command line

```
>>> from grass.pygrass.modules.shortcuts import general as g
>>> g.region(flags='g')
>>> from grass.pygrass.modules.shortcuts import vector as v
>>> v.info('schools')
```

- Modules treated as objects
- Output handling not as easy => more programming skills necessary
- Currently, Python scripting library and PyGRASS modules package co-exist.
- Simplified, subjectif differentiation :
  - Scripting library for GRASS GIS users who want to begin coding scripts
  - PyGRASS modules more for Python users who want to use GRASS GIS

PyGRASS – Low-level access to data and functions

- Directly access geometry features and a selection of low-level C-functions
- Uses ctypes
- Allows programming complex GRASS GIS applications in Python, combining
  - Ease of Python programming
  - Performance of GRASS C-API
- Different packages, notably
  - raster
  - vector
  - gis

PyGRASS – Low-level access to data and functions

```
>>> from grass.pygrass.vector import VectorTopo
>>> schools = VectorTopo('schools')
>>> schools.open(mode='rw')
>>> schools.num_primitive_of('point')
167

>>> from grass.pygrass.vector.geometry import Point
>>> new_school = Point()
>>> new_school.x = 643550
>>> new_school.y = 216200
>>> schools.write(new_school)
>>> schools.close()

>>> schools.open(mode='r')
>>> schools.num_primitive_of('point')
168
```

PyGRASS – Low-level access to data and functions

```
>>> for school in schools:
...     print(school.id, school)
...
(1, Point(633649.285674, 221412.944348))
(2, Point(628787.129283, 223961.620521))
[...]
(167, Point(650870.509540, 247064.343249))
(168, Point(643550.000000, 216200.000000))
>>> schools.close()
>>> recrutement_area=new_school.buffer(1000)
```

http://grass.osgeo.org/grass71/manuals/libpython/pygrass_index.html

GRASS GIS C-API

- Over 30 years of development
- Parts have remained stable over entire period
- Some fundamental additions on the way :
  - Floating-point and null support (GRASS 5)
  - New vector library including network tools (GRASS 6)
  - Large-file support, significant performance optimization and much better cross-platform usability (GRASS 6-7).

GRASS GIS C-API

- Three main core libraries:
  - gis: fundamental operations, data structures and GRASS database management
  - raster: creation, handling and processing of raster data
  - vector: creation, handling and processing of vector data
- Many other libraries, including functions for different mathematical calculations, the treatment of satellite imagery, displaying maps, projection handling, 3D-support, etc.

GRASS GIS C-API

- Consistent naming scheme :
  - G_* = gis library
  - Rast_ = raster handling
  - Vect_* = vector handling
  - I_* = satellite imagery
  - etc.
- Host of data structures for dealing with specific GIS-related data and GRASS GIS data formats

http://grass.osgeo.org/programming7/

GRASS GIS C-API

- Reading all features in a vector map and selecting only the points for further treatment :

```
nlines = Vect_get_num_lines(&Map);
for (line = 1; line <= nlines; line++) {
    type = Vect_read_line(&Map, Points, Cats, line);
    if (!(type & GV_POINT))
        ontinue;
}
```

- Getting size of raster map and reading row by row :

```
inrast = Rast_allocate_buf(data_type);

/* Allocate output buffer, use input map data_type */
nrows = Rast_window_rows();
ncols = Rast_window_cols();

for (row = 0; row < nrows; row++) {
    G_percent(row, nrows, 2);
    Rast_get_row(infd, inrast, row, data_type);
}
```

Easy (G)UI creation

- GRASS GIS offers really easy automatic user interface creation. Ex v.db.addcolumn :

```
#%option G_OPT_V_MAP
#%end

#%option G_OPT_V_FIELD
#% label: Layer number where to add column(s)
#%end

#%option
#% key: columns
#% type: string
#% label: Name and type of the new column(s) ('name type
[,name type, ...]')
#% description: Data types depend on database backend, but
all support VARCHAR(), INT, DOUBLE PRECISION and DATE
#% required: yes
#%end
```

Easy (G)UI creation

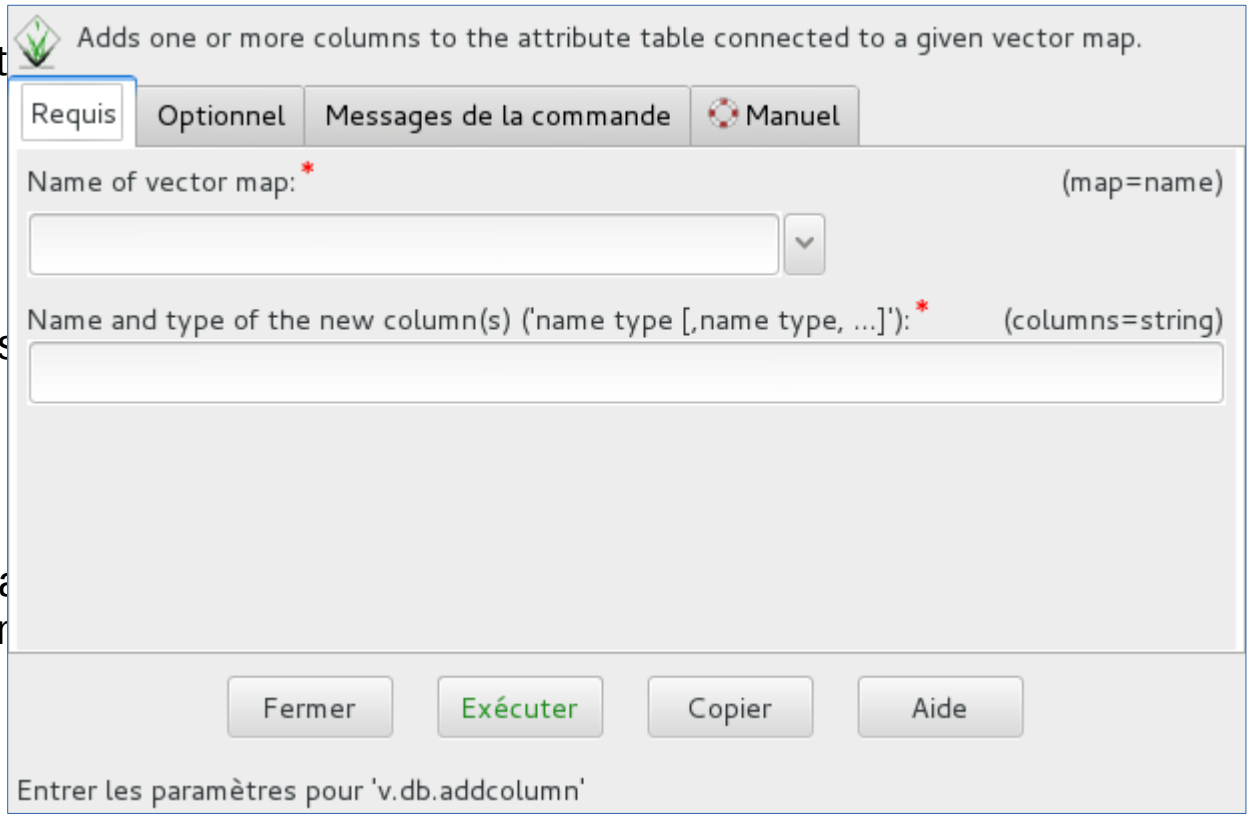
> v.db.addcolumn --help

Description:
 Adds one or more columns to the at

Keywords:
 vector, attribute table, database

Usage:
 v.db.addcolumn map=name [layer=s
   [--verbose] [--quiet] [--ui]

Flags:
 --h   Print usage summary
 --v   Sortie du module en mode bava
 --q   Sortie du module en mode siler
 --ui Force launching GUI dialog



Parameters:
   map   Name of vector map
         Or data source for direct OGR access
   layer   Layer number where to add column(s)
         Vector features can have category values in different layers.
   columns   Name and type of the new column(s) ('name type [,name type, ...]')
         Data types depend on database backend, but all support VARCHAR(), INT,
         DOUBLE PRECISION and DATE

Final notes

- To run any of the above examples, GRASS GIS environment has to be set up
  - Launch startup script
  - Define environment variables manually
- Coding standards :

http://trac.osgeo.org/grass/wiki/Submitting

http://grass.osgeo.org

Find all the information from this presentation at
http://grasswiki.osgeo.org/wiki/GRASS_GIS_APIs

Coming soon:
GRASS GIS 7!

THANKS