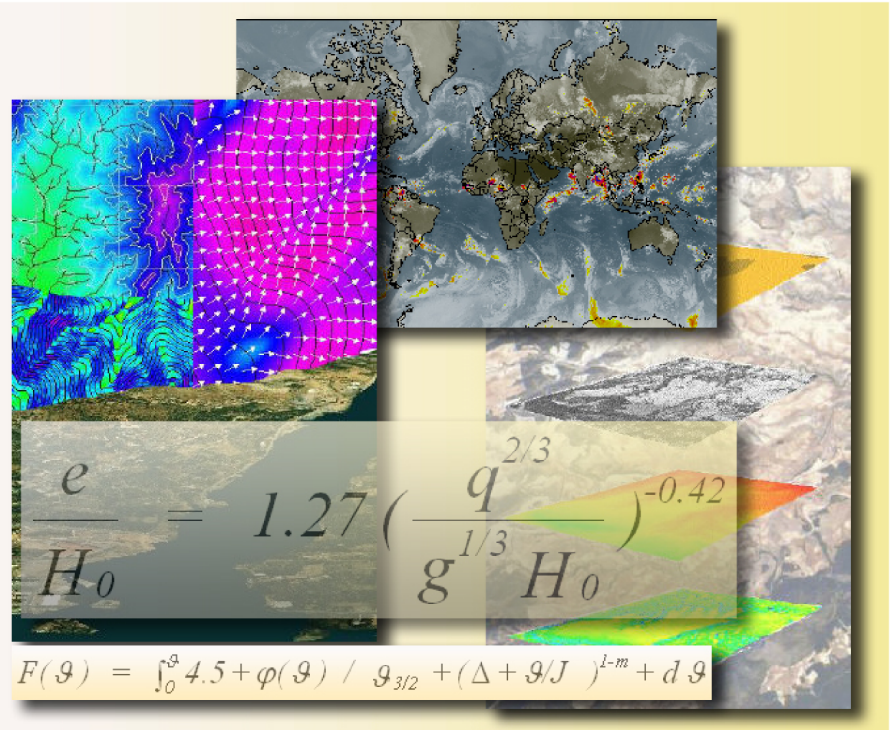
Universidad de Murcia

Software libre para SIG

Francisco Alonso Sarria
Jose Antonio Palazón Ferrando

Tecnologías de la Información Geográfica:

Territorio y Medio Ambiente



Murcia 20-23 septiembre' 04

Software Libre para Sistemas de Información Geográfica (versión 0.1) 20/09/04

F. Alonso Sarria (Dpto. Geografía, Física, Humana y Análisis Regional)
y
J.A. Palazón Ferrando (Dpto. Ecología e Hidrología)

Universidad de Murcia. España.

e-mail: `alonsarp@um.es` y `palazon@um.es`

Agradeceremos los comentarios, sugerencias, críticas constructivas, enlaces rotos, opiniones dispares, faltas y omisiones que puedan mejorar este documento, que se puede encontar en `caldum.um.es`.

Contenidos

1 Consideraciones generales acerca del software libre y los SIG **7**

1.1 ¿Qué es el software abierto/libre? . . . 7

1.2 ¿Por qué es interesante usar SL? . . . 8

1.3 ¿Por qué usar SL en SIG? . . . 11

1.4 *FreeGIS* . . . 12

2 GRASS **12**

2.1 GRASS (historia) . . . 14

2.1.1 Los primeros tiempos. GRASS en el USA/CERL (1982-1996) . . . 14

2.1.2 Transición. GRASS en la Universidad de Baylor (1997-1998) . . . 16

2.1.3 Consolidación y despegue. GPL GRASS (1999-2004) . . . 16

2.1.4 GRASS 5.3.0 . . . 17

2.1.5 GRASS 5.7.0 . . . 18

2.1.6 Modificaciones de cara al futuro . . . 19

2.2 Estructura de datos y directorios en GRASS . . . 20

2.2.1 El sentido de las LOCATION y los MAPSET . . . 20

2.2.2 Tipos de capas espaciales y formatos de almacenamiento . . . 21

2.2.3 La Región de trabajo . . . 21

2.2.4 Modelo raster . . . 22

2.2.5 Modelo vectorial (formato GRASS 5.0 y 5.3) . . . 23

2.2.6 Mapas de sites . . . 23

2.2.7 Modelo vectorial (formato GRASS 5.7) . . . 24

2.3 Módulos de GRASS . . . 24

2.3.1 Tipología de módulos de GRASS . . . 25

2.3.2 Interfaz de usuario de los módulos de GRASS . . . 26

3 Como hacerlo en GRASS **30**

3.1 ¿Como empezar? . . . 30

3.1.1 Instalar el programa . . . 30

3.1.2 Instalar una base de datos . . . 31

3.1.3 Inicio del programa . . . 31

3.2 ¿Como construir una LOCATION? . . . 31

3.2.1 Mapas en papel . . . 32

3.2.2 Digitalización de información raster . . . 32

3.2.3 Importación de otros formatos de SIG y CAD . . . 34

3.3 ¿Como compilar GRASS? . . . 34

3.3.1 configure y make . . . 34

3.4 ¿Como manipular la región de trabajo? . . . 37

4 Programación con GRASS **39**

4.1 Código binario . . . 39

4.1.1 Inicialización y Compilación. El primer programa de GRASS . . . 40

4.1.2 Como recopilar información acerca de la location y la región de trabajo . . . 40

4.1.3 Estructuras de Información Geográfica . . . 42

4.1.4 Transformaciones entre filas/columnas y coordenadas . . . 42

4.1.5 Gestión de la linea de comandos . . . 43

4.1.6 Manejo de ficheros raster . . . 44

4.2 BASH Scripts . . . 44

4.2.1 Como hacer un script con BASH . . . 44

4.2.2 Introducción de variables, parámetros y arrays . . . 45
4.2.3 Sentencias de control . . . 45
4.2.4 Filtros . . . 47
4.3 Interfaces gráficas de usuario (IGU) . . . 47
4.3.1 Personalización de TclTkgrass . . . 49

5 Integración de GRASS con otros programas 53
5.1 El programa de análisis de datos R . . . 53
5.1.1 Primeros pasos con R . . . 53
5.1.2 Expresiones en R . . . 54
5.1.3 Operadores en R . . . 54
5.1.4 Variables en R . . . 54
5.1.5 Funciones en R . . . 55
5.1.6 R y GRASS . . . 57
5.2 El programa de gestión de bases de datos PostgreSQL . . . 58
5.2.1 GRASS 5.0/5.3 y PostgreSQL como Base de Batos GeoRelacional . . . 60
5.2.2 GRASS 5.7 y PostgreSQL . . . 62
5.2.3 GRASS y PostGIS . . . 65
5.3 gstat . . . 68
5.3.1 Ejemplos de ficheros de instrucciones de gstat . . . 69
5.3.2 El paquete gstat de R . . . 69
5.4 Librerías, visores de mapas y servidores de mapas WEB . . . 71
5.4.1 CD-lives . . . 75

6 Entornos de trabajo (INUAMA) y docencia (microaulas) 76
6.1 Ventajas de un sistema multiusuario . . . 76
6.2 Curso de SIG . . . 76
6.3 Microaulas de terminales . . . 76
6.4 El laboratorio de SIG y teledetección del Instituto Universitario del Agua y del Medio Ambiente (INUAMA) . . . 77
6.5 Otras estrategias . . . 77

7 Bibliografía 78

Figuras

1 Personalización del escritorio de fvwm . . . 10
2 Ejemplo de aplicaciones WinXXX funcionado con wine . . . 11
3 Los 10 mayores proyectos de software libre . . . 14
4 Evolución del proyecto GRASS . . . 15
5 Evolución y desarrollo de las diferentes ramas de GRASS . . . 17
6 Arquitectura vectorial en GRASS 5.7 . . . 19
7 Pantallas de introducción a GRASS 5.0.3 y GRASS5.7.0 . . . 20
8 Estructura de directorios de las capas raster . . . 23
9 Estructura de directorios del formato vectorial . . . 23
10 Estructura de directorios para el almacenamiento de información vectorial integrando los formatos de GRASS 5.0 y 5.3 con los de GRASS 5.7 . . . 25
11 Mensaje de ayuda del módulo **d.rast** . . . 27
12 Página de manual de d.rast en español . . . 28
13 IGU del módulo i.points . . . 29
14 Pantalla de v.digit . . . 33
15 Ejecución en modo interactivo del módulo **g.region** . . . 38
16 Ejemplo de IGU con TclTk . . . 48
17 Estructura de directorios de tcltkgrass . . . 49
18 IGU de d.rast . . . 50
19 IGU de d.rast traducida al español . . . 51
20 Interfaz Gráfica de Usuario d.m en GRAS 5.7 . . . 52
21 Modelo georrelacional de bases de datos . . . 58
22 Consulta espacial en dos pasos en una base de datos georrelacional (ver tabla . . . 59
23 Visualización del mapa de polígonos de la Región de Murcia en función de los resultados de una consulta SQL . . . 63
24 Visualización del mapa de polígonos de la Región de Murcia en función de los resultados de una consulta SQL . . . 63
25 Resultados de una consulta con **d.what.vect** en GRASS 5.7 . . . 64
26 Observatorios meteorológicos situados en el municipio de Murcia . . . 66
27 Buffer a los observatorios meteorológicos de la Región de Murcia . . . 67
28 Buffer a los límites de los acuíferos de la Región de Murcia . . . 67
29 Árbol de decición de gstat . . . 68
30 Resultado del primer fichero de instrucciones para gstat . . . 70
31 Resultado del segundo fichero de instrucciones para gstat . . . 70
32 Resultados del script de la tabla 33) . . . 72
33 Pantalla de JUMP . . . 73
34 Pantalla de QGIS . . . 74
35 Pantalla de Thuban . . . 74
36 Pantalla de Mapserver . . . 75

Tablas

1 Funciones de GRASS . . . 13
2 Salida de **g.list rast** . . . 22
3 Estructura de un fichero de sites . . . 24
4 Algunos de los módulos de GRASS . . . 26
5 Resultados del comando **configure** de GRASS . . . 35
6 Error del comando **configure** de GRASS . . . 36
7 Salida de g.region -p (izquierda) y g.region -g (derecha) . . . 38
8 Un primer programa en C . . . 40
9 Consulta sobre las características de la LOCATION en un programa en C . . . 41
10 Consulta sobre las características de la LOCATION en un programa en C . . . 42
11 Estructura Cell_head . . . 42
12 Estructuras Option y Flag . . . 43
13 Conjunto de órdenes que forman el programa *guion* . . . 45
14 Programa *guion* incorporando variables . . . 46
15 Script de BASH incorporando un bucle for . . . 46
16 Algunas de las expresiones condicionales disponibles en BASH . . . 46
17 Expresiones aritméticas disponibles en BASH . . . 47
18 Ejemplo de programa en AWK . . . 47
19 Flujo de entrada al programa en AWK . . . 47
20 Flujo de salida del programa en AWK) . . . 48
21 Programa TclTk para la creación de una Intrfaz Gráfica de Usuario . . . 48
22 Sección de tcltkgrass que define el menú de ayuda . . . 49
23 Código TclTk para la IGU de d.rast . . . 50
24 Código TclTk para la IGU de d.rast en español . . . 51
25 Ejemplo de consulta espacial con una base de datos Geo-Relacional . . . 59
26 Ejemplo de consulta espacial con una base de datos Objeto-Relacional (PostGIS) . . . 59
27 Listado de tablas que aparecen en una base de datos . . . 60
28 Listado de campos que aparecen en una tabla . . . 61
29 Programa para pintar polígonos en función de los resultados de consultas SQL . . . 62
30 Consultas SQL incrporando PostGIS . . . 66
31 Ejemplo de fichero de instrucciones de gstat (1) . . . 69
32 Ejemplo de fichero de instrucciones de gstat (2) . . . 69
33 Ejemplo de script de R utilizando funciones de gstat . . . 71

1 Consideraciones generales acerca del software libre y los SIG

1.1 ¿Qué es el software abierto/libre?

Para recurrir a las tecnologías de la información (en adelante: TI) es preciso utilizar programas informáticos, estos conforman desde el sistema operativo hasta las aplicaciones más sencillas. En la mayor parte de los casos su origen es ajeno, han sido desarrollados por otras personas, y por lo tanto debe utilizarse de acuerdo a un conjunto de condiciones, que determinan los derechos y obligaciones del programador y del usuario, a las que llamamos **licencias**.

Las licencias se basan en los criterios que mueven a los autores a desarrollar los programas, así tenemos dos situaciones generales:

Software propietario: también llamado comercial o privativo, que se ha desarrollado con fines comerciales y su uso conlleva el pago de licencias de uso de distinta naturaleza (por puestos de trabajo, por volumen de uso, ...). Se trata de programas de fuente cerradas que ha de manejarse como una caja negra: sabemos que información requieren y que resultado deben proporcionar.

Software libre: Del inglés *fee software* se confunde en ocasiones con software gratuito pero no es necesaria esta condición para serlo, por contra, si debe garantizar una serie derechos o libertades al usuario, tal y como propone Richard Stalman uno de los pioneros del SL y fundador de la *Free Software Fundation* y del proyecto GNU[1] las libertades son:

- La libertad de usar el programa, con cualquier propósito (libertad 0).
- La libertad de estudiar cómo funciona el programa, y adaptarlo a tus necesidades (libertad 1).
- La libertad de distribuir copias, con lo que puedes ayudar a tu vecino (libertad 2).
- La libertad de mejorar el programa y hacer públicas las mejoras a los demás, de modo que toda la comunidad se beneficie. (libertad 3).

Para garantizar estas libertades es **imprescindible** disponer del código fuente, de todo el código.

Las licencias consagran estos derechos y pueden variar en algunas obligaciones así existen varios tipos entre los que cabe destacar:

GPL: Se trata de la *GNU Public License*[2] que adoptan la mayor parte de los programas de la FSF, y de la que existen diversas traducciones al castellano[3]. Su objetivo principal es salvaguardar a la comunidad.

Creative Common[4]: Se trata de una licencia de cesión de la obra (sea del tipo que fuere) al dominio público.

Otras: FreeBSD, Mozilla, X, etc.

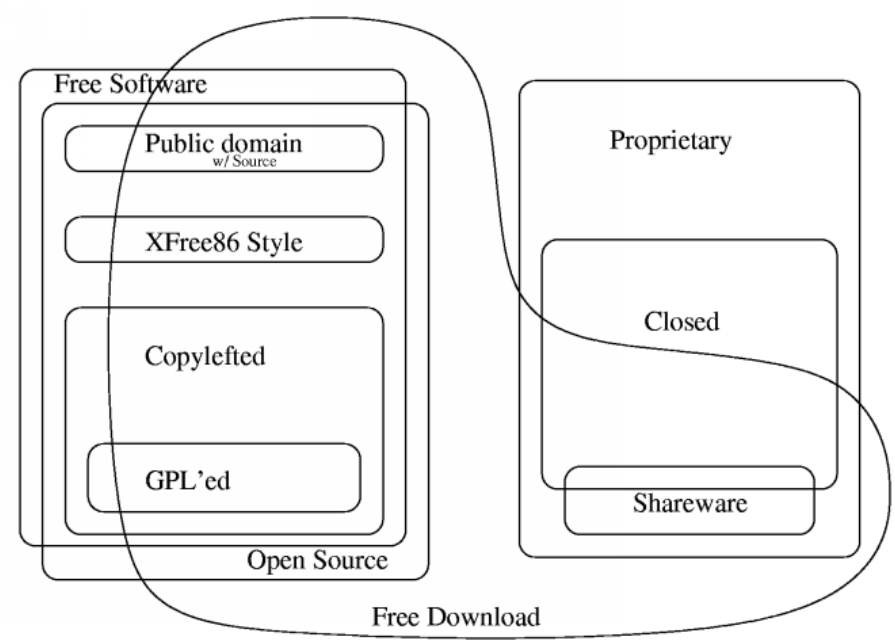
El siguiente esquema representa los conceptos relacionados con el tipo de software, para más información puede consultarse la página de la FSF.

En resumen la idea de las licencias comerciales radica en la posibilidad de la patente de ideas: si se patenta una idea cualquiera que la piense debe pagar al propietario de la patente. Esta situación puede llevar a un monopolio de ideas.

[1] en cuya página web podemos encontrar más información sobre diversos aspectos relacionados con el SL: `http://www.gnu.org`. El nombre del proyecto es un acrónimo que significa *GNU is not Unix*

[2] http://www.fsf.org/licenses/licenses.html

[3] http://gugs.sindominio.net/gnu-gpl



1.2 ¿Por qué es interesante usar SL?

El SL es a menudo un gran desconocido, es más, el mayor volumen de información que recibe el usuario medio es gracias a las practicas de FUD[5], este término se corresponde con las siglas inglesas de *Fear, Uncertainty, Doubt* (Miedo, Desconcierto y Duda). Esta información recibida por los ciudadanos, en la mayor parte de los casos es maliciosamente errónea y en otros falsa por torpe ignorancia, merced a ello se ha creado en torno al SL una serie de mitos:

1. **El SL está desarrollado por un puñado de voluntarios que pueden abandonar en cualquier momento.**

   El número de desarrolladores de SL es muy elevado y no solo actúan de forma voluntaria, además, muchas empresas (por propia iniciativa o por contratos con administraciones públicas), también lo están desarrollando. Fundamentalmente estas empresas dirigen su negocio a un modelo de servicio y no de explotación de un monopolio. Además, cualquiera puede continuar con un proyecto abandonado o iniciar una nueva rama de desarrollo (Eric S. Raymond: *La Catedral y el Bazar*[6]).

2. **El software que cuesta dinero es mejor que el que puede conseguirse gratis.**

   El precio no es necesariamente una medida de calidad, por el contrario, la accesibilidad del código, su posible estudio y depuración mejoran notablemente el resultado. También sucede que en SL no es necesario "justificar" que un error debe ser corregido. Muy al contrario, rápidamente se comunica el error para poner a salvo a la comunidad e iniciar los procesos de depuración. En otros softwares las correcciones, que llegan tarde o no llegan, se les llaman "servicios" o peor aún: torpeza del usuario (por ejemplo: pantallazo azul). Actualmente los grandes sistemas de cálculo (*cluster* o grupos de muchos ordenadores trabajando juntos) se montan con sistemas Linux: NASA, Pixar, ...

3. **WinXXX no me cuesta nada, viene con el ordenador.**

   Esta es una forma encubierta de venta en la que se ha de pagar un producto y además se hace responsable al vendedor del ordenador que no tiene capacidad alguna para resolver los problemas que surjan con el producto vendido.

   Cuanto cuesta realmente la licencia de los programas incluidos en el equipo. ¿Sería necesario un equipo de tamañas (y habitualmente caras) prestaciones para usarlo con otro sistema operativo? ¿Qué restricciones de uso impone la licencia de su sistema operativo? ¿Le impide usar cierto tipo de software? ¿Le impide acceder a cierto tipo de información?

[5] http://wiki.hispalinux.es/moin/FUD
[6] http://www.sindominio.net/biblioweb/telematica/catedral.html

4. **Si el SL no funciona no tengo a quien reclamar.**

   ¿Ha leído la licencia del WinXP? ¿Conoce al maestro armero?

   Por otro lado, cada vez más empresas (de todos los calibres) dan soporte al SL, una relación puede encontrarse en *El Libro Blanco del Software Libre*[7]; se potencian, así, las empresas de servicios, que tienen la ventaja económica de dejar el dinero en la región que lo genera. Pero sin duda la mejor baza del SL es la existencia de una comunidad, que se comunica a través de Internet y mediante foros de discusión, páginas de documentación, listas de correo, chats, etc. Existen proyectos de documentación de Linux (*The Linux Documentation Proyect*: TLDP[8]) y la correspondiente traducción de la documentación (mucha está en castellano[9]). Pueden consultarse numerosas revistas electrónicas de acceso libre (*Linux Journal*[10], *LinuxFocus*[11](con traducción a numerosos idiomas, incluido el nuestro), *Linux Gazette*[12] y su traducción al castellano *La Gaceta de Linux*[13]. Montones de FAQ[14], *tips and tricks*[15], *Howto's*[16], grupos de noticias (*news*)[17], ...

5. **No tengo/encuentro personal formado para trabajar con SL**

   Cada día aumenta el número de personas que utilizan programas de SL. Cualquier persona que utilice realmente (¿nivel de usuario?) un procesador de textos propietario puede usar uno libre (o mejor pasarse a Latex[18]). Además en muchos casos el software utilizado es especifico y el usuario debe aprender a manejarlo. Extremadura[19] (y le siguen otras como Andalucía, Valencia, Aragón, Madrid, etc.) ha iniciado un proceso de alfabetización tecnológica basado en SL.

6. **Costes de migración**

   ¿Cuanto cuesta migrar a una versión de sistema operativo superior? No sólo ha de calcularse el precio de la "actualización" de la licencia, seguramente el ordenador estará obsoleto. Además el software libre puede ejecutarse para cualquier plataforma e incluso compartir plataforma para más información ver el apartado 6, en la página 76.

7. **El SL es muy feo**

   Separemos dos aspectos: eficiencia y belleza. Cuando se desea ser eficiente se utilizan sistemas eficientes, habitualmente basados en lenguajes con sintaxis (ventaja, analogía entre lenguajes y modos de operar en el caso de SL). Cuando resolvemos tareas simples o muy de vez en cuando puede ser agradable un entorno gráfico. SL es fundamentalmente libertad, una de esas libertades es la de elegir (por gusto o por necesidad). Como ejemplos:

   - Para saber quien esta conectado a mi máquina basta con pulsar w y retorno de carro. (estando en una terminal de texto).
   - ¿Acaso esto es tan feo la imagen mostrada por la figura 1?

8. **Hay pocos programas**

   La distribución Woody de Debian incorpora cerca de 9000 programas ..., para encontrar lo que se necesita puede consultar Freshmeat[20] o Sourceforge[21]. También puede encontrase la equivalencia entre programas propietarios y libres en *La Tabla de equivalencias / reemplazos / de software análogo a Windows en Linux*.[22]

[7] http://www.libroblanco.com
[8] http://www.tldp.org
[9] http://es.tldp.org
[10] http://www.linuxjournal.com
[11] http://www.tldp.org/linuxfocus
[12] http://linuxgazette.net
[13] http://www.gacetadelinux.com
[14] *Frecuenly Asked Questions* (Respuestas a las preguntas más frecuentes)
[15] consejos y trucos
[16] Como hacer ...
[17] es.comp.os.linux
[18] http://es.tldp.org/CervanTeX/CervanTeX
[19] http://www.linex.org
[20] http://freshmeat.net
[21] http://sourceforge.net/
[22] http://linuxshop.ru/linuxbegin/win-lin-soft-spanish/index.shtml

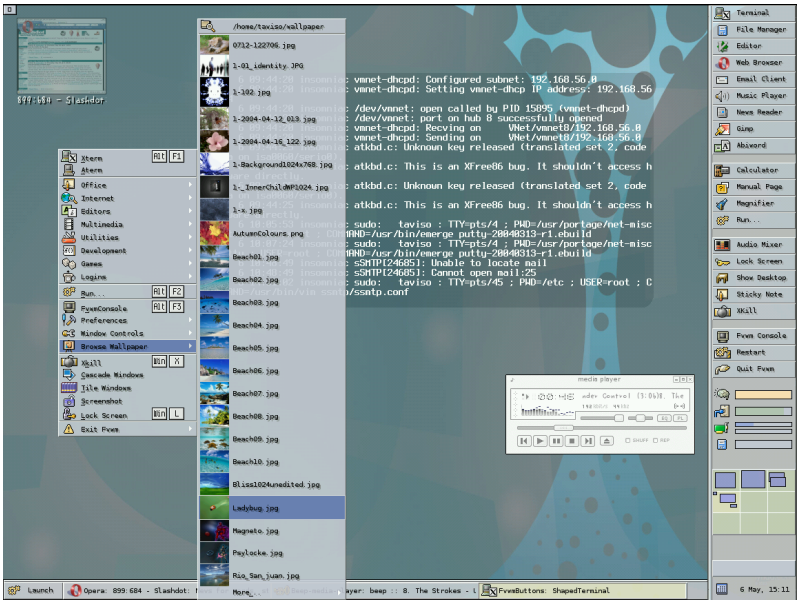


Figura 1: Personalización del escritorio de fvwm

9. **Tengo que tirar la aplicaciones desarrolladas en WinXXX**

   Puede utilizarse un API libre para estos programas como puede ser wine[23] con el que se ejecutan gran número de programas (incluidos juegos), ver figura 2.

10. **No es seguro**

    Es MUY seguro y numerosos sistemas de seguridad están construidos con SL.

11. **Trabaja con formatos de información extraños**

    Nada más lejos de la realidad, se utilizan formatos estándar (pdf, word, exel, wp, ...) y existe un numeroso grupo de programas destinado al cambio de información entre formatos. Sin duda lo más relevante en SL es la utilización de formatos abiertos que permiten disponer de la información de una forma libre.

12. **Es difícil de instalar**

    Algunas distribuciones modernas de Linux no preguntan nada al usuario.

    Podemos utilizar software libre sin instalarlo: CD-live (Knoppix, freeduc, caldum, etc.)

    Podemos utilizar vía red el ordenador (cpu, memoria, disco, programas, ...) de un amigo.

13. **El SL tiene problemas con el hardware**

    En general esto no es cierto, de hecho puede arrancarse un sistema completo en un ordenador con un CD-live sin añadir ni un sólo *driver* y con todo funcionando —de hecho, esta es la mejor forma de probar un ordenador antes de comprarlo: arrancando una CD-live y comprobando que todo funciona—. Para evitar problemas o adquirir el hardware más conveniente puede consultarse la página de *Linux on line!*[24].

14. **Los usuarios de SL son unos pedantes repelentes y pendencieros**

    ¡Mentira! Todo es paz y armonía.

[23] http://www.winehq.com
[24] http://www.linux.org/hardware

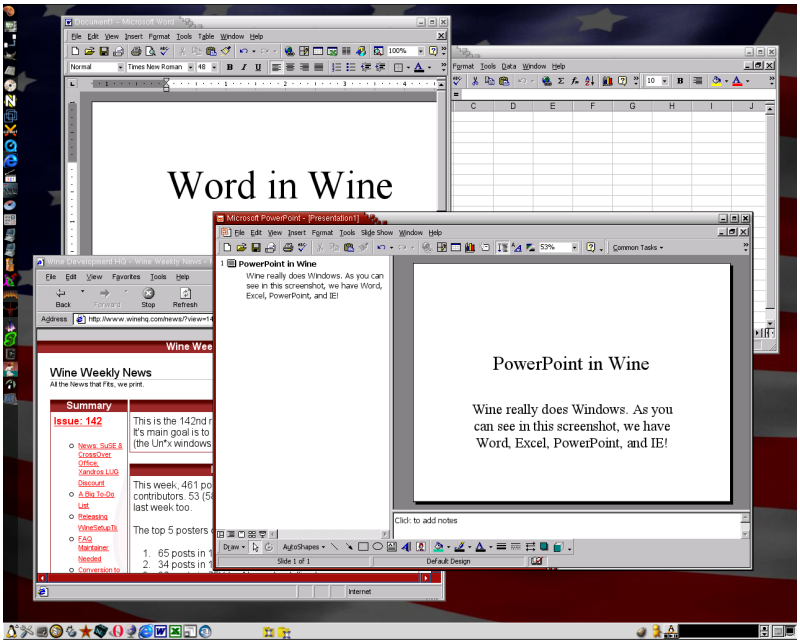


Figura 2: Ejemplo de aplicaciones WinXXX funcionado con wine

1.3 ¿Por qué usar SL en SIG?

Supongamos un caso habitual: se inicia un proyecto donde la componente SIG es relevante y se dispone de un grupo joven y con poca experiencia. Los problemas que surgiran en el desarrollo del proyecto son de diversa naturaleza:

- Incorporación de nuevos programas.
- Formación: Es necesaria la formación especializada de algunos de los miembros del grupo.
- Incorporación de la información: Los datos, la cartografía, bibliografía, protocolos, etc. deben ser mecanizados y puestos a disposición de los miembros del equipo.
- Adquisición de hardware: Los equipos obsoletos deben ser reemplazados para evitar pérdidas de tiempo.

Esencialmente el trabajo a desarrollar en las primeras etapas es el que corresponde a la mecanización de la información. Si utilizamos SL elegiremos la herramienta más adecuada para esta tarea, con independencia de trabajos posteriores. La elección de un software propietario condiciona todo el proceso. Además, desde que se inicia el proyecto hasta que se concluye la mecanización de los datos suele transcurrir un largo periodo de tiempo. La elección de un software propietario nos obliga a pagar la licencia completa antes

de empezar, cuando realmente estamos en disposición de utilizar la herramienta ya deberemos pagar “una actualización” y siempre por adelantado.

La adquisición de software propietario exigirá seguir cursos de pago desarrollados por la propia empresa, la elección de software libre permite la utilización de tutoriales y manuales disponibles en la red; además de los foros y listas de distribución propias de cada herramienta.

El contrato del software propietario limita el acceso al programa a un número dado en relación con el precio de la licencia. El SL puede ser utilizado libremente por todos lo miembros del equipo.

La utilización de software libre permite la “convivencia” en un sistema multiusuario, multiproceso que evita la duplicación de información e instalación de programas.

Si la herramienta utiliza el lenguaje AveHendida y la actualización (que por fin hace lo que usted realmente esperaba de ella) usa el lenguaje VeoLaBase y sus desarrollo no son trasladables: no importa puede empezar de nuevo el desarrollo. En SL los lenguajes son los propios del sistema (habitualmente independientes de los programas), las versiones nuevas pueden afectar de manera muy localizada.

También puede ser peor: no aparece la siguiente versión del programa puesto que la empresa desarrolladora ha sido adquirida por la competencia y se pone término al desarrollo.

La información que alojamos en un SIG no sólo tiene que ver con los datos, espaciales o no, también es importante la adecuación y el desarrollo de protocolos y la experiencia en el manejo de la herramienta.

Desde un punto de vista menos pragmático pero más importante en el plano teórico deben considerarse otros aspectos relacionados con la propia esencia del análisis SIG. Los algoritmos utilizados en un SIG son complejos y pueden influir de manera importante en los resultados. Los SIG se utilizan para procesar datos de entrada en función de algún tipo de modelo y tomar decisiones a partir de los resultados, por tanto si el algoritmo no es el adecuado los resultados pueden ser erróneos.

Es importante, por tanto, poder trabajar con un sistema que nos permita estudiar, entender e incluso modificar la implementación de una determinada función.

1.4 *FreeGIS*

La página web del proyecto FreeGIS[25] alberga diversas alternativas y herramientas complementarias para las personas que trabajan con SIG y optan por el SL. Buena parte de ellas se han desarrollado de manera que pueden integrarse con GRASS que sigue siendo el más importante Sistema de Información Geográfica entre los programas de código libre (figura 3).

2 GRASS

GRASS (Geographic Resources Analysis Support System) es el mayor proyecto de SIG desarrollado como software libre y, por el tamaño del código, uno de los 10 mayores proyectos [26] tal como se ve en la figura 3.

La adopción en 1999 de la licencia GPL protege los derechos de los autores del código al mismo tiempo que garantiza su libre distribución y da a los uasuarios la posibilidad de estudiar y modificar el código sin restricciones.

Como SIG multipropósito incluye un gran número de herramientas que le permiten llevar a cabo la mayor parte de las funcionalidades de un SIG. Sin embargo desde el principio GRASS se concibió como un programa dirigido a la gestión del terreno y la modelización y no como un programa de cartografía automática o *Desktop mapping*, por ello su interfaz gráfica se concibio como un medio (austero robusto y totalmente portable) y no como un fin en si mismo. La tabla 1 resume las principales funciones de GRASS (Neteler y Mitasova, 2002).

[25] http://freegis.org

[26] www.codecatalog.com (esta página ya no está activa)

Grupo de funciones	Funciones
Integración de datos geoespaciales	Importación-Exportación Transformación de coordenadas Transformación de formatos (raster, vectorial, sites) Interpolación espacial
Procesamiento de datos raster	Algebra de mapas Análisis de MDE Análisis estadístico Análisis de coste, *buffers*, cuencas visuales Medidas de ecología del paosaje Sistemas expertos
Procesamiento de datos vectoriales	Digitalización Superposición Análisis espacial
Procesamiento de *sites*	Modelización multitemporal-multiatributo *buffers* Interpolación Análisis estadístico
Procesamiento de imágenes de satélite	Análisis de datos multiespectrales Georreferenciación Análisis de Componentes Principales y Componentes Canónicos Corrección radiométrica Reclasificación y detección de bordes
Visualización	2D multicapa 3D multisuperficie Animaciones
Modelización y simulación	Hidrología, erosión, contaminación, etc.
Datos temporales	Inclusión y manejo de fechas en las capas de información
Procesamiento 3D	Algebra de mapas en 3D Interpolación y análisis 3D Visualización de isosuperficies
Integración con otras aplicaciones	R, gstat, PostgreSQL, MapServer, Vis5D, GDAL

Tabla 1: Funciones de GRASS

Top Ten Largest Packages by Total Number of Lines

Name	Version	NumFiles	TotalNumLines
mozilla	M15	14,079	2,344,285
linux	2.2.14	5,187	2,017,245
grass	5.0beta7	18,470	1,714,943
gcc	2.95.2	2,771	1,137,849
gdb	4.18	2,699	1,019,500
glibc	2.1.3	6,101	562,210
amaya	2.4	1,893	556,847
netbeans-jaga	060500	5,288	521,310
enhydra	3.0.1	4,540	506,472
Wine	20000227	1,634	506,177

Figura 3: Los 10 mayores proyectos de software libre

2.1 GRASS (historia)

2.1.1 Los primeros tiempos. GRASS en el USA/CERL (1982-1996)

A finales de la década de los setenta, el Acta Nacional de Política Medioambiental noteamericana implica la necesidad, por parte del ejército americano, de llevar a cabo una adecuada gestión ambiental en los espacios naturales a su cargo. Al no encontrar nigún programa comercial que cubriera estas necesidades, el Laboratorio de Investigación en Ingeniería de la Construcción del Cuerpo de Ingenieros del Ejército americano (USA/CERL) contrató un grupo de programadores para desarrollar un SIG con capacidad para manejar información raster y vectorial en un entorno Unix. De este modo se convirtió en uno de los primeros equipos trabajando en el desarrollo de un SIG sobre este sistema operativo, se procuró escribir el código para el Unix estandar. El estilo de programación de GRASS se caracterizó desde el principio por:

- Uso de librerias UNIX en la medida de lo posible, y creación de librerias específicas para las acciones más habituales.
- Manejo de tipos de datos tanto raster como vectoriales.
- Mayor atención al formato raster ya que el objetivo de GRASS era más el análisis de datos y la modelización que la cartografía automática.
- Capacidad de manejar mallas raster de diversas extensiones y resoluciones.
- Utilización de codificación *run-length* en los ficheros raster de manera transparente para el usuario.
- Los ficheros reclasificados se almacenan como *Look Up Tables* para ahorrar espacio.
- Aceptación de los estandares *de facto* del momento en lugar de tratar de crear nuevos formatos.
- Organización del código fuente en una estructura de directorios que facilita la inclusión de contribuciones y la gestión de código en diferentes fases de prueba.

Los principales hitos en esta primera etapa (figura 4) fueron:

- 1982 Diversos programas de demostración en una máquina con procesador Z80.

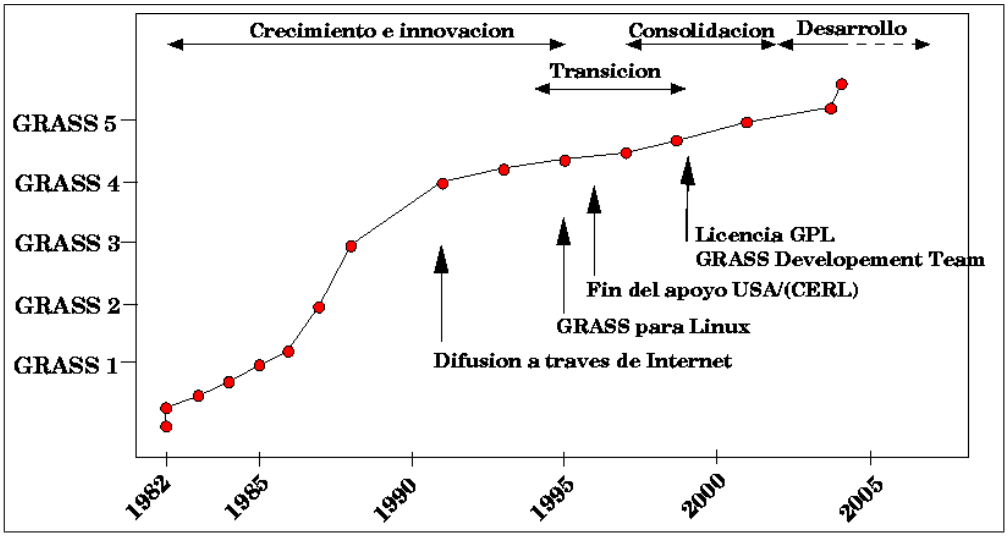


Figura 4: Evolución del proyecto GRASS

- 1982 Fort Hood Information System (FHIS) sobre un Vax 11/780[27].
- 1983 Installation Geographic Information System (IGIS) ya sobre un microordenador SUN1.
- 1984 Adopción del nombre GRASS.
- 1985 GRASS 1.0, primeros esfuerzos para coordinar un grupo de usuarios: GRASSnet[28]
- 1986 GRASS 1.1; recibe el premio "Exemplary Systems in Government Award" de URISA
- 1987 GRASS 2.0; Revista GRASSClippings para usuarios de GRASS[29]
- 1988 GRASS 3.0; premio I+D del ejército americano para el equipo de desarrollo formado por Webster, Goran, Shapiro y Westervelt.
- 1991 GRASS 4.0; acceso público a través de Internet, aumento de popularidad entre las universidades, empresas y agencias gubernamentales.
- 1993 GRASS 4.1; Premio del Federal Lab Consortium por excelencia en la transferencia de tecnología.
- 1995 GRASS 4.1.5; Versión para linux desarrollada por Yung-Tsung Kang de la Michigan State University.
- 1996 Fin del apoyo del USA/CERL a GRASS[30]

GRASS nació como programa de dominio público, es decir como el producto de una agencia estatal que se ponía a disposición de cualquier ciudadano. En consecuencia, se creó un amplio grupo de *beta-testers*, estos y los usuarios institucionales (National Park Service, Soil Conservation Service,..) contribuyeron al desarrollo de GRASS. La motivación de los usuarios institucionales era asegurar el mantenimiento de un proyecto que les ofrecía una excelente herramienta gratuita y en el que, además, podían influir para que cumpliese sus expectativas. Por otro lado, varias universidades adoptaron GRASS para investigación y docencia (Central Washington University, The University of Arkansas, Texas A & M University, The University of California at Berkeley, Rutgers University).

GRASS recibe entre 1988 y 1993 varios premios, pero al mismo tiempo comienzan las críticas al hecho de que un programa tan potente fuera de dominio público. Por otro lado los desarrolladores de SIG

[27] http://grass.itc.it/fhgis83rep.html
[28] http://grass.itc.it/mailing_lists/grasshopper_1995/GRASSList y http://op.gfzpotsdam.de/GRASSList/
[29] http://skagit.meas.ncsu.edu/ helena/grasswork/grassclip/
[30] http://web.archive.org/web/19990210183258/www.cecer.army.mil/grass/transition.html

comercial, aprovechando ese carácter de dominio público, empezaron a utilizar partes de código de GRASS e incluirlo en sus propios productos de pago.

A finales de los 80 y principios de los 90 el CERL empezó a intentar descargarse del coste asociado al mantenimiento de GRASS. Se crea la *Open GRASS Foundation* en cooperación con grupos de usuarios encargada de la transferencia tecnológica garantizando que GRASS continuaba abierto. La fundación se reoriento hacia temas de interoperabilidad convirtiendose en la *Open GIS Foundation* convirtiendose finalmente en un Consorcio (*OpenGIS Consortium*).

En 1996 USA/CERL, tras lanzar la versión beta de GRASS 5.0, anuncia el fin del apoyo al programa, aunque se llega a varios acuerdos con empresas comerciales de SIG para potenciar la comercialización de GRASS (GRASSLANDS). Muchos usuarios de GRASS migran hacia otros sistemas debido a la falta de continuidad en el desarrollo. Sin embargo muchos usuarios continuaron utilizando GRASS ya que seguía siendo uno de los programas más avanzados para la gestión de Sistema de Información Geográfica.

2.1.2 Transición. GRASS en la Universidad de Baylor (1997-1998)

A finales de 1997, un grupo de la Universidad de Baylor comienza a desarrollar un nuevo sitio web para GRASS que contendrá el código fuente de la versión 4.1 del programa, los binarios para Solaris y la documentación. En noviembre aparece la versión 4.2 y empiza a considerarse la incorporación de binarios para Linux y Windows NT. GRASS 4.2 incorpora mejoras procedentes del CERL y de la misma universidad de Baylor, se mejora la documentación y se busca cooperación de otros grupos para el futuro desarrollo del proyecto.

En 1998, Markus Neteler de la Universidad de Hannover comienza a trabajar con el grupo de Baylor. Añade más de 50 módulos, limpia el código, elimina errores y coordina la creación de una nueva IGU (TclTkgrass). Se lanza la versión 4.2.1

2.1.3 Consolidación y despegue. GPL GRASS (1999-2004)

Entre 1998 y 2000 hay un período de reorganización a partir del código fuente no oficial que el USA/CERL envió al nuevo equipo de desarrollo de GRASS formado en 1999 entorno a Markus Neteler y Bruce Byars. Este nuevo equipo se formo desde cero, sin ningún contacto con el del USA/CERL lo que significó la necesidad de revisar, comprender y depurar todo el código de un programa hecho por otros.

En 1999 GRASS adopta la licencia GPL y la filosofía del software libre, de esta manera se garantiza que compañías privadas no podrán apropiarse de futuros desarrollos del programa. GRASS había utilizado todas las posibilidades de Internet desde el lanzamiento de la versión 4.0 en 1991, en 1999 se da un paso más al integrar el código en un servidor CVS. De este modo el equipo de desarrollo, disperso por todo el globo, podía trabajar de forma conjunta. Se lanza la versión 4.3 como versión mejorada de la última estable creada en el USA CERL y se comienza a trabajar con los prototipos existentes de la versión 5.0. Esta versión supone el mayor cambio en años:

- Se añade la capacidad de trabajar en mapas raster con números reales.
- Herramientas de exportación-importación a varios formatos (shape, e00, mapinfo, etc.) utilizando la librería GDAL[31] (Geoespatial Data Abstraction Type). Se incluye también una librería para que otros programas puedan leer datos de GRASS.
- Se incrementa la portabilidad (MacOS X, CygWin)
- Reorganización de las funciones numéricas y utilización de las librerias LAPACK[32] (Linear Algebra Packages) y BLAS[33] (Basic Linear Algebra Subprograms) para optimizar la ejecución de cálculos matriciales.
- Apoyo en PROJ4[34] para permitir el uso de diferentes proyecciones y cambios entre ellas.

[31] http://www.remotesensing.org/gdal/
[32] http://www.netlib.org/lapack/
[33] http://www.netlib.org/blas/
[34] http://www.remotesensing.org/proj

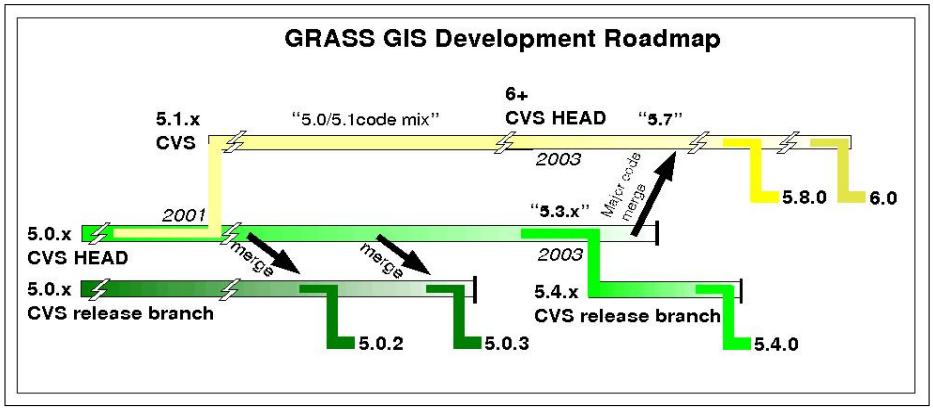


Figura 5: Evolución y desarrollo de las diferentes ramas de GRASS

- Mejora en la visualización de mapas
- Interfaces para otros prográmas de código abierto como PostgreSQL[35] (un gestor de bases de datos) R[36] (un programa de análisis estadístico) y gstat[37] (un programa de análisis geoestadístico).

A partir de diciembre del año 2000 se lanzan diveras versiones previas de GRASS 5.0 hasta que en septiembre de 2002 llega al primera versión estable. Mientras tanto en 2001, el centro de desarrollo de GRASS se había trasladado de Hannover a Trento (Italia).

En noviembre de 2003 se alcanza la versión 5.0.3, por otra parte se mantienen otras dos ramas de desarrollo. En primer lugar **GRASS 5.3.0** cuya versión de prueba fue publicada en mayo de 2004 y en segundo lugar **GRASS 5.7.0** cuya versión de desarrollo se publicó en junio de 2004.

La existencia de varias versiones funcionales es típica de los ciclos y métodos de de desarrollo del software libre. En este caso se trata de tener una versión estable y probada (GRASS 5.0.3); una versión en pruebas para que los usuarios detecten fallos (GRASS 5.3.0) que cuando se considere estable pasará a denominarse GRASS 5.4.0 y una versión en desarrollo avanzado y fase de pruebas al mismo tiempo (GRASS 5.7.0 que culminará en GRASS 6.0 (figura 5).

2.1.4 GRASS 5.3.0

Se trata de una rama de desarrollo en preparación de GRASS 5.4 que incluye numerosos parches a problemas detectados en la rama 5.0 añadiendo nuevas funciones respecto a GRASS 5.0.3:

- Utilización de la librería PROJ4 que permite el cambio de sistema de proyección o de datum.
- Compilación de parte del código común como librerías compartidas para reducir el tamaño de los módulos.
- Mejoras a la IGU tcltkgrass.
- Mejoras en el módulo NVIZ de visualización 3D.
- Herramientas experimentales de manejo de imágenes LIDAR.
- Modelos experimentales para simulación hidrológica
- Módulos para manejar capas verdaderamente 3D (aplicaciones en Geología, Oceanografía, Meteorología, etc.).

[35] http://www.postgresql.org/
[36] http://cran.r-project.org/
[37] http://www.gstat.org/

- Importación-Exportación a MATLAB.
- Análisis estadísitico de series temporales de capas raster.
- Importación y exportación al formato de Vis5D[38]. Este permite visualizar datos 5dimensionales (3 coordenadas espaciales, tiempo y variable).
- Colores de 24bits
- Mejora en la importación de datos raster utilizando las librerías GDAL
- Internacionalización mediante la incorpotración de la librería FreeType[39], que permite la incorporación de acentos y eñes, y un mayor esfuerzo en la traducción de documentación con la colaboración de grupos locales de usuarios.

2.1.5 GRASS 5.7.0

La rama de desarrollo 5.7 supone una modificación de mucho más calado, especialmente en lo que respecta a la gestión de datos vectoriales, al menos tan importante como la que supuso GRASS 5.0 respecto a los datos raster (6). De hecho el final de esta rama será la versión 6 del programa. Entre los cambios más relevantes destacan:

- **Gestión de la información geométrica** En GRASS 5.7 se ha modificado completamente el gestor de información vectorial para poder manejar información 2D y 3D multicapa incluyendo topología. El nuevo formato es intercambiable entre plataformas de 32 y 64 bits. Se ha incluido además un nuevo sistema de indexado espacial, basado en el algoritmo R-tree, que acelera los accesos. Otros formatos pueden importarse (permitiendo la limpieza de la topología de los mismos) e incluso utilizarse en su formato original con GRASS.
- **Multiformato** A partir de esta versión, las bases de datos de GRASS pueden incluir otros formatos de ficheros digitales (PostGIS, shape y otros) mediante la librería OGR.
- Una nueva librería de funciones *Directed Graph Library* permite hacer **análisis de redes**. Se ha implementado también operaciones de álgebra de mapas, intersección y extracción en vectorial.
- **Gestión de la información temática** El nuevo gestor de información vectorial permite además una completa y flexible integración de gestores de bases de datos para el manejo de atributos (PostgreSQL, mySQL, DBF y ODBC son las admitidas en este momento). Se utiliza SQL para hacer las consultas. También se ha implementado la posibilidad de edición gráfica de los atributos de los mapas vectoriales.
- **Interfaz de usuario** La ejecución en modo interactivo de cualquier módulo de GRASS, implica ahora la aparición de una Interfaz Gráfica de Usuario específica programada con TclTk. Además se dispone de un gestor de visualización. NVIZ se ha mejorado para incluir datos vectoriales tridimensionales. Nuevo entorno gráfico para visualización.
- **Multisesión** Ya es posible el uso concurrente de varias sesiones de un mismo usuario, también se permiten sesiones en paralelo de GRASS 5.3.x y 5.7.0 en la misma LOCATION.
- **Generación de nuevas LOCATIONs** Las nuevas LOCATION con la proyección establecida de forma automática utilizando los códigos EPSG (European Petroleum Survey GRoup). Esta opción aparece en la pantalla de inicio de GRASS 5.7 (figura 7)
- **Importación/Exportación** GRASS 5.7.0 se ha integrado con la librería GDAL/OGR para admitir un amplio rango de formatos raster y vectoriales.

[38] http://www.ssec.wisc.edu/ billh/vis5d.html
[39] http://www.freetype.org/

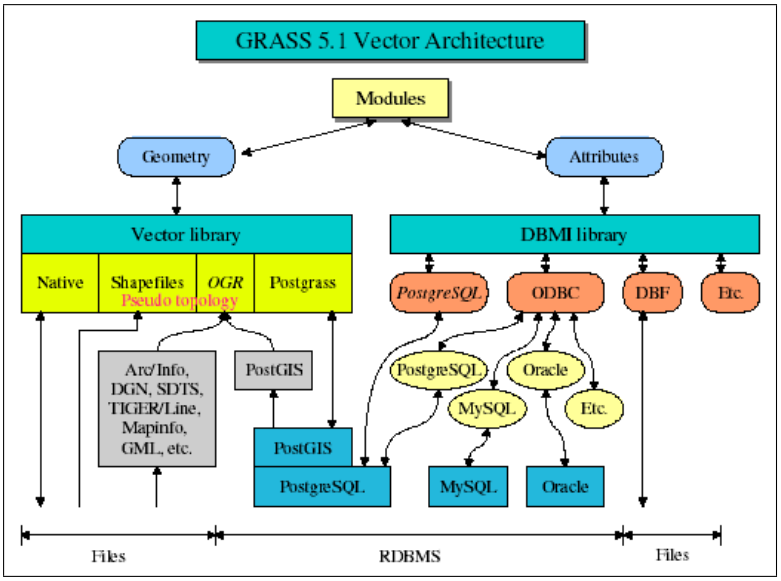


Figura 6: Arquitectura vectorial en GRASS 5.7

2.1.6 Modificaciones de cara al futuro

GRASS es un proyecto en continua evolución, por tanto existen varias lineas de desarrollo en marcha que siguen tanto las necesidades percibidas por el Equipo de Desarrollo como las apuntadas por los usuarios. Entre las ideas cuya implementación está en diferentes fases de desarrollo destacan:

- Desarrollo de *babyGRASS* para iPAQ (ordenadores de mano)
- Aplicaciones de cartografía web
- Nuevos módulos para incrementar la potencia de GRASS
- Nuevos campos de aplicación (geología, oceanografía, astronomía, meteorología o hidrología
- Sustitución de TclTkGRASS por interfaces de usuario más potentes
- Simplificación de los mecanismos de importación/exportación
- Nuevas herramientas de visualización y creación de cartografía
- Gestión mejorada de infomación multidimensional
- Gestión mejorada del tiempo en las bases de datos de GRASS. Adopción de la libreria *DateTime* de R
- Organización del código en paquetes (como en R)
- Mejorar las posibilidades de GRASS para trabajar en computación paralela o distribuida
- Inclusión de procedimientos para la gestión de metadatos
- Mejora de la documentación

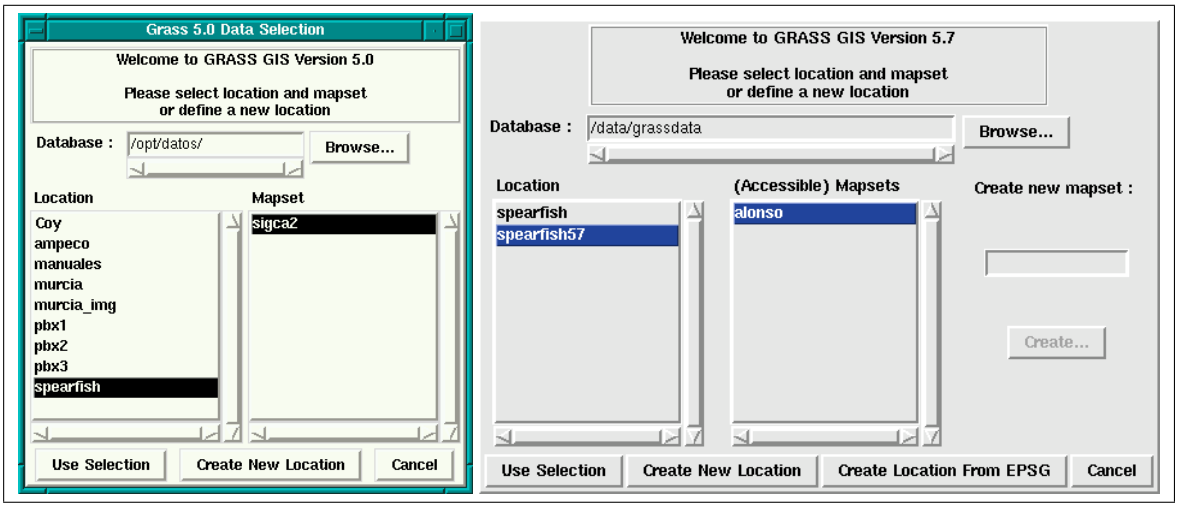


Figura 7: Pantallas de introducción a GRASS 5.0.3 y GRASS5.7.0

2.2 Estructura de datos y directorios en GRASS

2.2.1 El sentido de las LOCATION y los MAPSET

La primera dificultad con que se encuentra alguien que comienza a experimentar con GRASS es el sentido de estos conceptos ya que es la primera cuestión que el programa pregunta al usuario (figura 7). En realidad se trata de decirle al programa algo tan sencillo como en que directorios están los datos con los que queremos trabajar.

GRASS es un entorno multiusuario en el que se asume que varias personas van a trabajar en varios proyectos al mismo tiempo accediendo al mismo tiempo a un ordenador que actúa como **servidor** almacenando datos y programas. Los usuarios acceden a él a través de **terminales** conectadas a una red local o a Internet. La necesidad de evitar que varios usuarios manipulen a la vez un mismo mapa (con el consiguiente riesgo de pérdida de información), es la razón por la que los datos se almacenan en una estuctura de directorios que puede parecer a primera vista algo compleja.

Existe un directorio principal llamado **DATABASE** que, en principio, contiene todas las bases de datos de todos los proyectos gestionados por GRASS.

De este directorio cuelgan varios subdirectarios denominados **LOCATION**, cada uno de ellos corresponde a un proyecto de trabajo, utiliza un único sistema de proyección para sus mapas y, en principio, todos los mapas que contiene son, más o menos, de la misma zona.

Puesto que cada **LOCATION** corresponde a un proyecto en el que trabajan varias personas, cada una de ellas tiene un subdirectorio propio llamado **MAPSET** (conjunto o carpeta de mapas en inglés), que contiene todos los mapas que esa persona ha hecho y que, por tanto, puede modificar (figuras 8, 9 y 10).

Cada **LOCATION** tiene un **MAPSET** llamado **PERMANENT** que contiene todos aquellos mapas que se consideran acabados, que no van a sufrir posteriores modificaciones y que, por tanto, ningún usuario podrá modificar aunque todos los pueden leer. Por tanto un usuario puede leer mapas de su **MAPSET** y de **PERMANENT** pero solo puede escribirlos en su **MAPSET**. Si se conceden los permisos oportunos, el usuario podrá además leer, pero no modificar, mapas de todos los demás **MAPSETS**. En definitiva la filosofía de GRASS es que cualquier usuario puede consultar todos los mapas pero sólo puede alterar los suyos.

En realidad los **MAPSET** no tienen porque corresponder con usuarios físicos, también pueden existir mapsets por áreas temáticas (geología, hidrología, etc.) o por tareas (interpolación, modelos, etc.), siempre con la limitación de que cada **MAPSET** tiene un único propietario y que cada usuario del sistema sólo puede acceder a GRASS mediante un **MAPSET** del que sea propietario.

Por tanto, cuando se entra en GRASS se define cual será el **MAPSET** activo, el único en el que el usuario podrá crear nuevas capas de información. Aunque sólo pueda escribir en un **MAPSET**, podrá leer

datos de todos los **MAPSETs** de la **LOCATION** (siempre que los dueños de estos no jhayan establecido restricciones de acceso).

2.2.2 Tipos de capas espaciales y formatos de almacenamiento

Existen 3 tipos básicos de capas espaciales en GRASS:

rast Corresponden a capas de datos en formato raster. Es en el tratamiento de este tipo de capas donde GRASS se muestra más potente

vect Corresponden a capas de lineas o de polígonos en formato vectorial. El formato de almacenamiento y gestión de este tipo de datos se ha modificando profundamente en la versión 5.7 del programa para superar algunas de las limitaciones de las versiones anteriores.

sites Corresponden a capas de puntos, tienden a desaparecer en las últimas versiones de GRASS asimilados a capas vectoriales.

En una base de datos de GRASS existen más elementos que los tres tipos de capas anteriormente presentados

paint icon files Iconos para ser usados con módulos *paint* para crear mapas (poco utilizados).

paint label files Etiquetas para ser usados con módulos *paint* para crear mapas (poco utilizados).

region definition files Ficheros de definición de regiones. Las regiones son un elemento básico en el manejo de GRASS

imagery group files Definición de un *grupo de imagen*. Se trata de un mecanismo de GRASS para explicitar que capas raster corresponden a diferentes bandas de una imagen de satélite.

3D view parameters fichero de parámetros de visualización para el módulo **d.3d**

El módulo **g.list** presenta un listado de cualquiera de los 8 elementos de información antes mencionados. Para ello hay que ejecutar el módulo seguido de un solo parámetro, el nombre del tipo de elemento del que se quiere obtener un listado, estos nombres son: *rast, vect, sites, icon, labels, sites, region, group y 3dview*. Por ejemplo, para obtener un listado de los mapas vectoriales disponibles debes ejecutar:

GRASS: /path > **g.list** vect <ENTER>

Este comando nos presenta un listado con todas las capas del tipo elegido presentes en el conjunto de MAPSETS al que tenemos acceso. En principio estos son el **MAPSET PERMANENT** y el nuestro (el que hemos introducido al entrar al programa). Esta especificación por defecto puede modificarse con el módulo **g.mapsets**. Así la siguiente orden

GRASS: /path > **g.list** rast <ENTER>

dará el resultado que aparece en la tabla 2.

2.2.3 La Región de trabajo

Una capa raster está formada por una matriz de datos georreferenciada. Esta georreferenciación consiste en asignar coordenadas a los límites superior (Norte), inferior (Sur), derecho (Este) e izquierdo (Oeste) de la matriz. A partir de esos valores y del número de filas y columnas de la matriz, se deduce la resolución (tamaño de pixel) de la capa raster. En programas menos potentes, Idrisi por ejemplo, todas los capas raster que constituyen una base de datos deben tener exactamente la misma georreferenciación (filas, columnas, Este, Norte, Oeste, Sur y resolución). En GRASS las capas de una base de datos pueden ubicarse en el

raster files available in mapset raul:						
3_320	978c_5m	curvas36	h977_2b	mdt_ram1	r46	rm_us
889_320_3	cont36	curvas42	h977_2z	mdtrm1	r47	termu_rm
933r	cont42	curv_ram	m2	mdtrm100	r48	urb1
976c4_cm.pc	cuencas3	h25m	mdt_978c_25m	qk36	rm100	urba_mina1
978c_25m	curva1	h977_2a	mdt_978c_5m	qk42	rm1as	urba_rm
raster files available in mapset cuencaspc:						
ralgecirasr	rcarrizalejor	respunar	rmoror			
rarcasr	rcasarejosr	rlebolr	rnogalter			
raster files available in mapset PERMANENT:						
acc_cont36	basin42	curvas42	drain37	qk36	red6	
acc_cont42	basin5	curvas9	drain38	qk37	redcor32_f	
acc_cont9	cc	curvascm11	drain39	qk38	redord5	
acu_cont11	cont11	curvascm15	drain42	qk39	temp	
acu_cont37	cont36	curvascm1_b	drain9	qk42	tempo	
acu_cont38	cont37	curvascm23	lnacc_cont36	red11	tempo2	
acu_cont39	cont38	curvascm36	lnacc_cont42	red12	tempob	
basin11	cont39	curvascm37	lnacc_cont9	red13	tempoc	
basin12	cont42	curvascm38	lnacu_cont37	red1_b	tempod	

Tabla 2: Salida de **g.list rast**

mismo lugar, en lugares diferentes o ser parcialmente coincidentes; pueden tener además resoluciones diferentes.

De este modo las propiedades geométricas del área de trabajo son independientes de las que tienen las capas presentes en la base de datos. Debe, por tanto, ser establecida por el usuario y puede modificarse en todo momento (aunque estas modificaciones han de hacerse con precaución puesto que van a modificar cualquier tipo de análisis o visualización posterior).

En el **MAPSET PERMANENT** se guardan también las coordenadas de Región de trabajo por defecto en el fichero DEFAULT_WIND. Los ficheros PROJ_UNITS y PROJ_INFO contienen información acerca de la proyección utilizada en la base de datos. Finalmente en el **MAPSET** activo existe un fichero WIND que contiene la región de trabajo activa en cada momento.

2.2.4 Modelo raster

En GRASS se utilizan varios ficheros para almacenar una sola capa raster. Tienen todos el mismo nombre y, en lugar de distinguirse por la extensión, se distinguen por el directorio en el que se almacenan dentro de un determinado MAPSET. Estos directorios son:

/cell Contiene la matriz de datos, es decir los números que contienen las diferentes celdillas (formato binario).

/cellhd Contiene la información necesaria para leer y ubicar espacialmente los datos contenidos en el fichero anterior: Número de fiulas y columnas y coordenadas de los límites (formato ASCII).

/cats Guarda las etiquetas de texto que, en caso de que la variable sea cualitativa, se asocian a los identificadores numéricos almacenadso en **/cell** (formato ASCII).

/colr Guarda información acerca de la paleta de colores que se va a utilizar para representar el mapa (formato ASCII)

/fcell En caso de que la variable representada por la capa raster sea de tipo real, contiene información relativa a la codificación en coma flotante (formato binario).

/cell_misc Contiene estadísticos sobre la capa (formato ASCII) y la definición de las celdillas con valor nulo (formato binario).

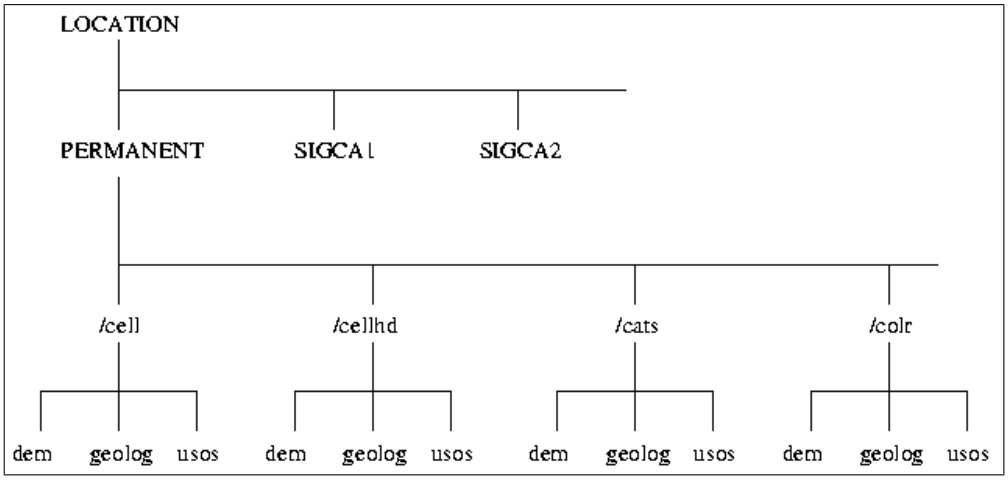


Figura 8: Estructura de directorios de las capas raster

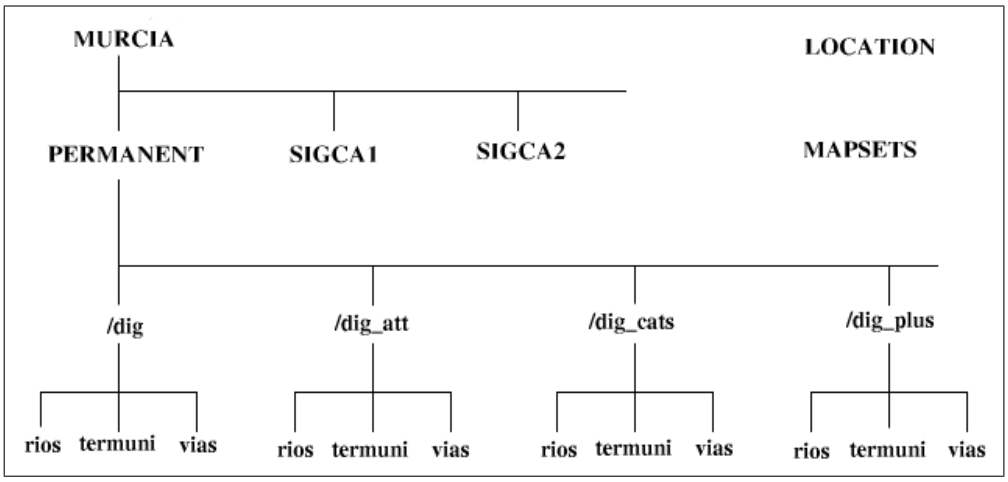


Figura 9: Estructura de directorios del formato vectorial

2.2.5 Modelo vectorial (formato GRASS 5.0 y 5.3)

/dig En este directorio se almacenan las coordenadas de los objetos junto con la indicación de si se trata de lineas o arcos que forman polígonos (formato binario).

/dig_att Almacena los ficheros que contienen los identificadores de los objetos definidos anteriormente (formato ASCII).

/dig_cats contiene la correspondencia entre categorías numéricas o alfanuméricas y los identificadores definidos anteriormente (formato ASCII). Tiende a sustituirse por enlaces a bases de datos

/dig_plus Se guardan ficheros que contienen la información topológica necesaria para cualquier tipo de consulta o análisis sobre los datos (formato binario).

/dig_ascii se almacenan copias en formato ASCII de los ficheros que se guardan en **dig**.

2.2.6 Mapas de sites

Este formato (ASCII) se utiliza para almacenar datos relativos a puntos concretos de la superficie terrestre. Los puntos se definen por 2 o 3 coordenadas y un conjunto de propiedades alfanuméricas. Se trata

```
name|diques
desc|Localización y características de los diques de contención de sedimentos
598680|420540|#1 %14 %4 %1 %1962 %0 @BOLOS @GAVION @Desconocido @Caliza,cong @Pino,dens,mat
598106|420635|#2 %15 %4 %2 %1962 %0 @BOLOS @GAVION @Desconocido @Caliza,cong @Pino,dens,mat
598035|4206577|#3 %16 %4 %16 %1962 %1996 @BOLOS @GAVION @BUENO @Arcillas @Pino,cultivo
598081|4207041|#4 %17 %4 %17 %1962 %1996 @BOLOS @GAVION @BUENO @Arcillas @Pino,cultivo
597958|4207501|#5 %18 %4 %18 %1996 %0 %0.96 @BOLOS @GAVION @BUENO @Arcillas @Pino,cultivo
579933|4207734|#6 %19 %4 %19 %1996 %0 %1.94 @BOLOS @GAVION @BUENO @Arcillas @Pino,matorr
597785|4208067|#7 %20 %4 %20 %1996 %0 %0.58 @BOLOS @GAVION @BUENO @Arcillas @Pino,matorr
597746|4208226|#8 %21 %4 %21 %1996 %0 %0.8 @BOLOS @GAVION @BUENO @Arcillas @Pino,matorr
```

Tabla 3: Estructura de un fichero de sites

de un formato bastante anticuado y que tienden a desaparecer. Los puntos se integrarían de esta manera en el nuevo formato vectorial (GRASS 5.7). En la tabla 3 aparece un ejemplo de fichero de sites. Las dos primeras lineas contienen el nombre del fichero y una descripción de los datos que contiene. A continuación hay una linea por site con los siguientes campos:

- Coordenadas separadas por "|"
- Identificador de cada site precedido de "#"
- Valores numéricos precedidos de "%"
- Valores alfanuméricos precedidos de "@"

2.2.7 Modelo vectorial (formato GRASS 5.7)

La estructura de directorios en esta nueva versión se complementa con un nuevo directorio llamado **/vector**, situado en el mismo nivel que los directorios **/cell**, **/cellhd**, etc. Este contiene un directorio por cada mapa vectorial, los cuales contienen a su vez tres ficheros:

head contiene metainformación e información de cabecera que antes se almacenaba en dig (formato ASCII)

coor almacena la información geométrica de los objetos (formato binario)

topo almacena información topológica

De este modo pueden convivir ambos formatos vectoriales en un mismo **MAPSET** ya que la estructura de directorios no se solapa (figura 10). GRASS 5.7.0 incorpora módulos para hacer la conversión entre ambos formatos.

2.3 Módulos de GRASS

Uno de los criterios que pueden emplearse para clasificar los diferentes programas para la gestión de información geográfica (lo que normalmente se denomina un SIG), es el predominio de los botones o de la linea de comandos. GRASS es un programa basado eminentemente en linea de comandos.

GRASS se maneja en una **consola de texto** en la que podemos incluir órdenes en formato texto. Salvo excepciones, no existen iconos, cuadros de diálogo, etc. Todas las órdenes se introducen por linea de comandos. Si bien al principios puede parecer un poco pesado, especialmente a usuarios acostumbrados a trabajar en Windows, tiene diversas ventajas como la posibilidad de combinar módulos de GRASS con las diversas herramientas de procesado de información de Unix, tal como se verá en el apartado de programación de scripts con BASH.

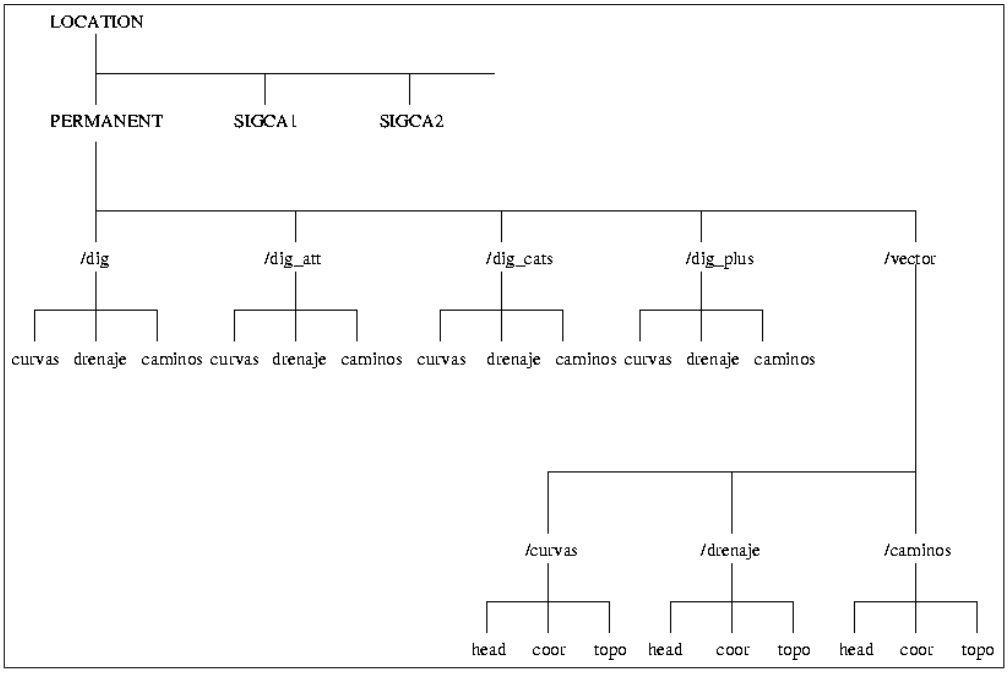


Figura 10: Estructura de directorios para el almacenamiento de información vectorial integrando los formatos de GRASS 5.0 y 5.3 con los de GRASS 5.7

Para la visualización de datos permite “abrir” monitores que pueden utilizarse para visualizar información gráfica a lo largo del desarrollo de una sesión con GRASS. Los módulos de consulta sobre el mapa o zoom requieren también interactuar con el mapa mostrado en el monitor.

El trabajar con comandos tecleados en lugar de menus o botones recuerda al viejo MS-DOS. Sin embargo Unix incorpora una linea de comandos mucho más potente que facilita el trabajo. En primer lugar se mantiene un histórico de las órdenes tecleadas al que puede accederse de diversos modos, por otro lado, si tras teclear algunas letras presionamos la tecla de tabulación, el sistema listará todos los comandos cuyo nombre empieza con esa particular combinación de caracteres. Esta característica será muy útil para localizar los diferentes módulos de GRASS.

2.3.1 Tipología de módulos de GRASS

Los nombres de los módulos de GRASS están constituidos por una letra minúscula que hace referencia al tipo del comando, un punto y el nombre del comando. Los tipos de comando, y sus letras identificadoras son:

- **g.*** -Módulos de gestión de ficheros y propósito general
- **d.*** -Módulos de sálida gráfica
- **r.*** -Módulos para análisis de capas raster
- **v.*** -Módulos para análisis de capas vectoriales
- **i.*** -Módulos para tratamiento de imágenes de satélite
- **s.*** -Módulos de procesamiento de sites
- **m.*** -Miscelanea de módulos con propósitos diversos
- **p.*** y **ps.*** Creación de cartografía y creación de cartografía postscript

r.in.gdal	Importación de capas raster de múltiples formatos mediante las librerías GDAL
v.in.shape	Importación de capas vectoriales en formato shape
s.in.ascii	Importación de capas de sites desde ficheros de texto
r.out.tiff	Exportación de capas raster a formato geotiff
v.out.dxf	Exportación de vectoriales a DXF
d.what.rast	Consulta interactiva sobre el monitor gráfico de los valores de las celdillas en una capa raster
d.what.vect	Consulta interactiva sobre el monitor gráfico de los identificadores de los objetos sobre una capa vectorial
d.what.sites	Consulta interactiva sobre el monitor gráfico de los valores de los sites en una capa de sites
r.surf.contour	Interpolación a partir de curvas de nivel rasterizadas
s.surf.idw	Interpolación por media ponderada por inverso de al distancia a partir de un mapa de sites
r.to.vect	Transformación de raster a vectorial
s.to.rast	Transformación de sites a raster

Tabla 4: Algunos de los módulos de GRASS

Esta particular notacion permite la clasificación de los módulos de GRASS, evita su confusión con otros comandos Unix y además permite echar un vistazo rápido al conjunto de módulos. Si tecleamos por ejemplo *r.* y seguidamente pulsamos la tecla de tabulación, obtendremos un listado de todos los módulos relacionados con el tratamiento de ficheros raster.

En algunos casos aparece una segunda palabra que nos da algo más de información sobre el propósito del programa, como por ejemplo:

- **in**, son módulos de importación de otros formatos a GRASS
- **out**, son módulos de exportación de GRASS a otros formatos,
- **surf**, interpola superficies a partir de mapas de puntos o lineas
- **what**, permite pinchar en un monitor gráfico y obtener las características de la celdilla, objeto o punto seleccionado
- **to**, realiza diversas operaciones de paso de formato entre raster, vectorial y sites

Existen diferentes combinaciones de estas palabras con las iniciales antes vistas. De este modo resulta bastante intuitivo tanto adquirir una idea general del propósito de un módulo a partir de su nombre como deducir cual puede ser el nombre de un módulo a partir de una idea general de cual es la función que buscamos. En la tabla 4 aparecen algunos de los módulos de GRASS junto con una breve descripción de su objetivo.

El orden en el que se han presentado anteriormente los diferentes tipos de comandos de GRASS implica también el orden en que se van a ir utilizando en el proceso de aprendizaje del programa así como lo habitualmente que se utilizan trabajando con el. De hecho los módulos **p.*** y **ps.*** apenas se utilizan ya que es más conveniente y sencillo utilizar el **driver PNG** para la generación de cartografía que va a ser impresa.

2.3.2 Interfaz de usuario de los módulos de GRASS

Los módulos de GRASS son programas independientes y, en muchos casos, programados por personas sin ningún de relación. Lo que da unidad al programa es que todos los módulos comparten:

- El tipo de estructuras de datos (formatos de los ficheros) a los que acceden y que crean
- La interfaz y las comunicaciones con el usuario tienen un diseño común lo que simplifica la interacción con el usuario
- Se utiliza un conjunto de librerias que incluyen funciones SIG en lenguaje C que simplifican la programación y aseguran que los módulos se comporten de forma consistente aunque hayan sido programados por equipos diferentes.

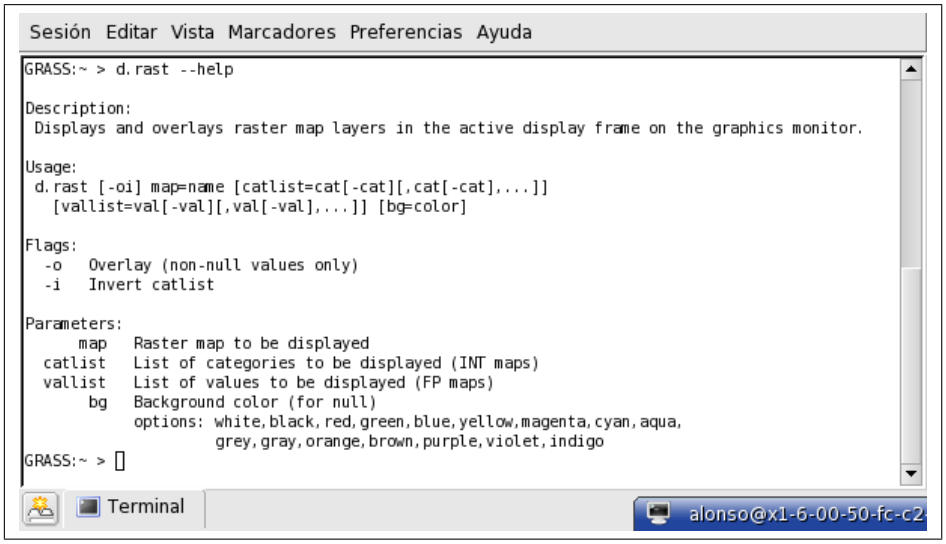


Figura 11: Mensaje de ayuda del módulo **d.rast**

Para un usuario novel, lo que más interesa es conocer algo más de como se van a comunicar con nosotros los diversos módulos de GRASS.

La mayor parte de los módulos transforman información de entrada (capas raster, vectoriales o de puntos) en información de salida (nuevas capas, representaciones gráficas, estadísticas o informes), esta transformación viene condicionada por un conjunto de **parámetros** y **opciones** que deben ser introducidos por el usuario. Los parámetros pueden ser **obligatorios**, es decir aquellos sin cuya presencia el módulo no puede funcionar, u **optativos** (los introducimos si queremos y si no queremos no), las opciones son como su propio nombre indica siempre optativas.

Una de las opciones más relevantes es **-help** que proporciona una descripción sucinta de las opciones y parámetros del módulo. Así la orden:

GRASS: /path > d.rast -help<ENTER>

proporciona la respuesta que aparece en la figura 11.

que incluye una breve descripción del objetivo del módulo e indicaciones de uso en linea de comandos. Al mismo tiempo proporciona un listado de las opciones (*flags*) y los parámetros. Los parámetros optativos aparecen entre corchetes en la descripción de la linea de comando. En este ejemplo, el módulo **d.rast** pinta una capa raster en el monitor gráfico que esté seleccionado, su parámetro obligatorio es, lógicamente, la capa que queremos pintar.

El módulo **g.manual** seguido del nombre de un módulo proporciona una página de manual con una descripción más completa de los propósitos, parámetros y opciones del comando. Tradicionalmente estas páginas de manual se codificaban con el formato de las páginas de manual de Unix y aparecían en la consola de texto. En las últimas versiones el usuario tiene la opción de visualizar las páginas de manual escritas en lenguaje HTML sobre un navegador. En la Universidad de Murcia se está desarrollando un proyecto de traducción de las páginas de manual de GRASS denominado *yerba*[40] (figura 12).

Existen dos modos de ejecutar los diferentes módulos de GRASS. En primer lugar, los módulos pueden ejecutarse en **modo interactivo**, simplemente se teclea el nombre del módulo y se espera a que el programa nos pregunte por los valores de las diferentes opciones y parámetros del mismo. En la versión 5.7, y en las anteriores si se utiliza **tcltkgrass**, esto se hace con una Interfaz Gráfica de Usuario en forma de cuadro de diálogo (figura 18). La otra posibilidad es ejecutarlo en **linea de comandos** pasando al módulo toda la información que necesita tal como se nos indica al utilizar la opción -help.

[40] www.um.es/geograf/sigmur/yerba/

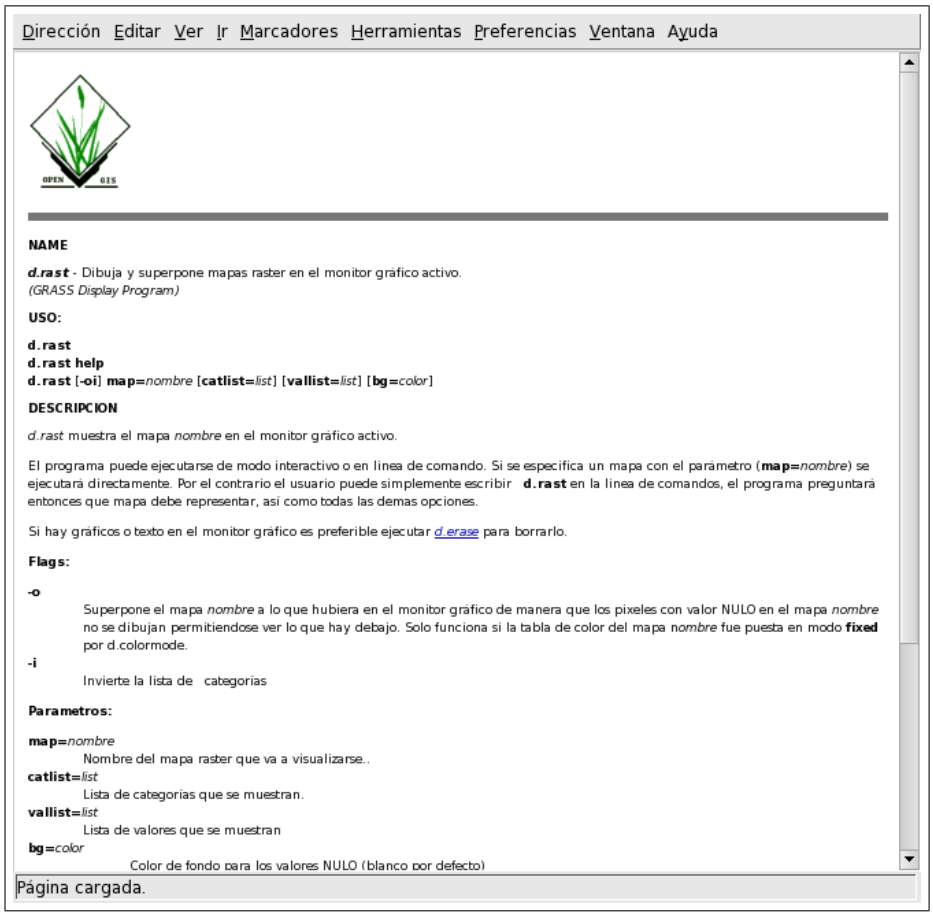


Figura 12: Página de manual de d.rast en español

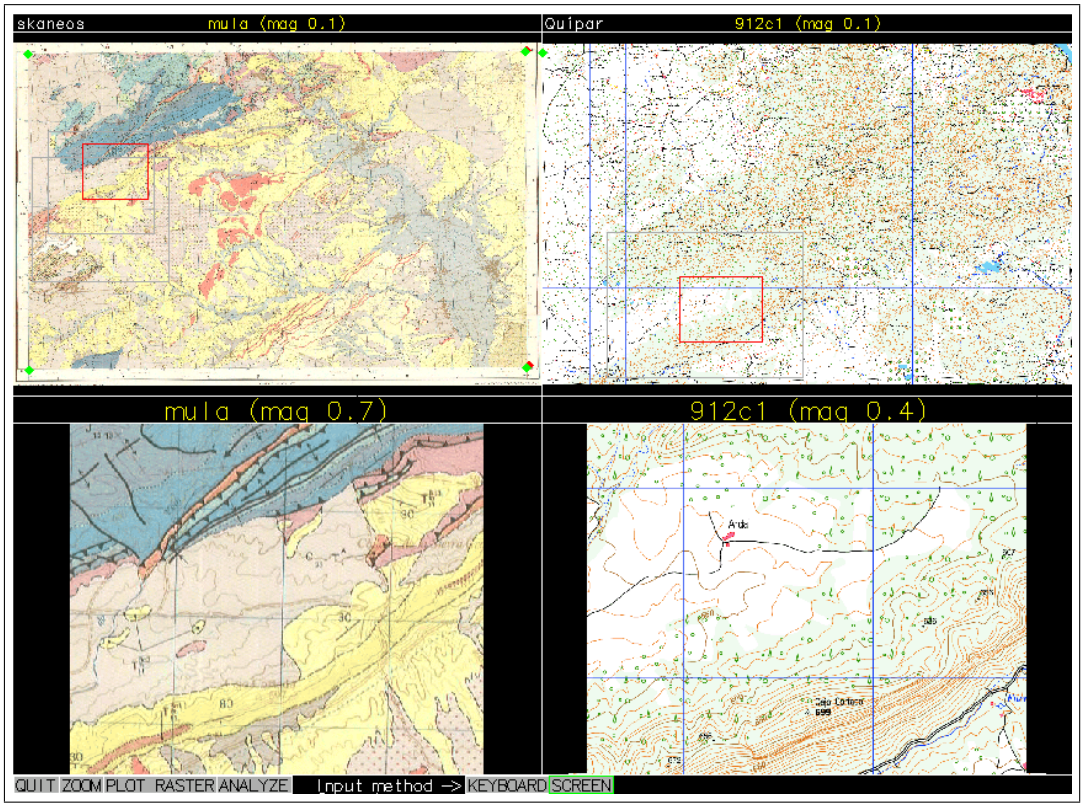


Figura 13: IGU del módulo i.points

El modo interactivo resulta adecuando cuando no se sabe muy bien que hace el programa y cuales son los parámetros y opciones que deben introducirse. Sin embargo algunos módulos permiten introducir múltiples opciones aunque estas apenas se usan. Un ejemplo es **d.rast** cuyo uso interactivo obligaría a contestar a 6 cuestiones mientras que en la mayoría de los casos lo único que queremos hacer es visualizar una capa raster:

GRASS: /path > d.rast *capa_raster*<ENTER>

en estos casos es preferible el modo linea de comandos.

La mayor parte de los módulos de GRASS son miniprogramas a los que se pasan unos datos (espaciales) de entrada y producen unos datos (espaciales) de salida. El trabajo con un SIG implica en muchas ocasiones la integración de varios de estos módulos generando un algoritmo que lleva a cabo una tarea compleja. Si estos módulos se ejecutan en linea de comandos, pueden escribirse en un fichero para luego ejecutarse todos juntos como un *script* tal como se verá posteriormente.

Un caso aparte son aquellos módulos cuya ejecución implica interacción compleja con los gráficos como puede ser la digitalización (**v.digit**, la extracción de perfiles (**d.profile**), la búsqueda y selección de puntos de control (**i.points**) o la combinación de bandas de color (**i.composite**). En la figura 13 aparece la pantalla de **i.points** y en la figura 14 la de **v.digit**.

3 Como hacerlo en GRASS

3.1 ¿Como empezar?

3.1.1 Instalar el programa

Desde las páginas del proyecto GRASS[41] pueden bajarse diversas versiones del programa. Las opciones que da esta página, para cada una de las versiones del programa, son:

- Bajar el código fuente, es la opción más apropiada para usuarios avanzados de sistemas Unix capaces de compilar sus propios programas ya que la instalación puede personalizarse y se optimiza para el tipo de máquina en que se compila
- Bajar el código binario, mucho más sencillo de instalar para un usuario novato de Unix
- Bajar los manuales de los módulos y documentación variada acerca del programa

Las versiones disponibles del programa son:

- GRASS 4.3, antigua
- GRASS 5.0, estable
- GRASS 5.3, en fase de depuración por parte de los usuarios
- GRASS 5.7, terminando la fase de desarrollo aunque lo suficientemente madura como para trabajar con ella

Para iniciarse en el manejo del programa las más aconsejables son la segunda y la tercera. En caso de que instalemos la versión estable debemos bajar los ficheros:

- grass5.0.3_i686-pc-linux-gnu_install.sh
- grass5.0.3_i686-pc-linux-gnu_bin.tar.gz

y ejecutar:

GRASS: /path > sh grass5.0.3_i686-pc-linux-gnu_install.sh grass5.0.3_i686-pc-linux-gnu_bin.tar.gz <ENTER>

Esta orden instalará GRASS en el directorio /usr/local del sistema de archivos y colocará un fichero ejecutable en el directorio /usr/local/bin que es uno de los directorios incluidos en el **PATH** del sistema. Si no se tiene permiso de escritura sobre el directorio /usr/local puede hacerse una instalación local en cualquier directorio de nuestro **HOME**. Para ver como hacerlo hay que ejecutar:

GRASS: /path > sh grass5.0.3_i686-pc-linux-gnu_install.sh -h<ENTER>

[41] http://grass.itc.it/download.html

3.1.2 Instalar una base de datos

Antes de empezar a trabajar con GRASS es necesario preparar el directorio donde se van a instalar las diferentes bases de datos. Ya se han mencionado los conceptos de **DATABASE**, **LOCATION** y **MAPSET**. En estos momentos es necesario crear el directorio **DATABASE** en un directorio en el que tengamos permiso de escritura, por ejemplo:

GRASS: /path > cd /opt/data<ENTER>

GRASS: /path > mkdir grassdata<ENTER>

Cuando se comienza a trabajar con GRASS, es conveniente instalar una base de datos de prueba. Una de las más utilizadas es *Spearfish* que puede descargarse de la dirección[42].

Para instalarla basta con copiar el fichero descargado (spearfish_grass50data.tar.gz) en un directorio que actuará como **DATABASE** (por ejemplo /opt/datos) y ejecutar:

GRASS: /path > tar xvfz spearfish_grass50data.tar.gz<ENTER>

este comando descomprime el fichero e instalará la **LOCATION** spearfish.

3.1.3 Inicio del programa

Dependiendo de la distribución de GRASS que estemos manejando y del modo en que esté instalado, el comando para ejecutar GRASS puede variar (grass4.3, grass5, grass53, grass57, etc). Para confirmar cual es la orden de entrada basta con escribir:

grass <TAB>

ya que Unix, si comenzamos a escribir una orden y pulsamos el tabulador, la completa o, en caso de varias opciones, nos las presenta en pantalla.

Una vez inicializado GRASS, en la primera pantalla que aparece (figura 7) debe indicarse la base de datos con la que se va a trabajar y los directorios de trabajo dentro de la misma (**LOCATION** y **MAPSET**).

En este caso los valores serían:

DATABASE: /data/grassdata
LOCATION: spearfish
MAPSET: *usuario*

donde *usuario* es el nombre del usuario que crea el **MAPSET** (sigca4 en el ejemplo de la figura 7). De esta manera se crea un **MAPSET** en el que se almacenarán todas las capas de información que se creen durante la sesión de trabajo.

3.2 ¿Como construir una LOCATION?

Una vez que se han hecho las primeras pruebas con una base de datos descargada de Internet para conseguir cierta destreza en el manejo del programa, el usuario tendrá la necesidad de crear su propia **LOCATION** para trabajar con sus datos.

Para ello basta con introducir como **LOCATION** el nombre de la nueva base de datos al iniciar la sesión. El programa preguntará si realmente se quiere construir un nueva **LOCATION**. Si se responde afirmativamente el usuario deberá entrar la siguiente información:

1. El sistema de coordenadas para la base de datos

[42] http://grass.itc.it/data.html

- X,Y (para datos espaciales no georreferenciados)
- Coordenadas geográficas (latitud-longitud)
- UTM
- Otras

2. Las características geométricas de la que será la región de trabajo por defecto (coordenadas de los límites del rectángulo que la circuncribe y número de filas y columnas de la malla)
3. Una descripción voluntaria de una linea

Dependiendo del tipo de sistema de coordenadas que se haya seleccionado, el sistema planteará otras opciones (por ejemplo la zona UTM o el elipsoide de referencia).

La incorporación de información a esta nueva LOCATION se puede hacer de diferentes modos en función de cuales sean las condiciones de la misma:

3.2.1 Mapas en papel

Deberán en primer lugar escanearse con una resolución adecuada y almacenarse como un fichero gráfico (formato tiff preferiblemente). Una vez dentro de GRASS deberán llevarse a cabo tres pasos:

- Importar el fichero a una **LOCATION** de tipo X,Y con **r.in.gdal**
- Buscar una muestra adecuada de puntos de control con **i.points**
- Hacer la rectificación con **i.rectify**. Este módulo aplicará a cada celdilla una ecuación de georreferenciación cuyos parámetros se obtienen de los puntos de control conseguidos. La capa resultante se almacena en la LOCATION seleccionada como destino

3.2.2 Digitalización de información raster

GRASS dispone de un módulo para la digitalización de información **v.digit**. Este permite digitalizar sobre una capa raster georreferenciada. Se trata de un módulo que tiene una interfaz de usuario propia consistente en un sistema de menús que se desarrolla en la consola y se activa por teclado y la transformación del monitor gráfico en una pantalla de digitalización (figura 14).

En el nivel principal del menú de **v.digit** encontramos las siguientes opciones:

Digitize Abre el menú de digitalización

Edit Permite editar (borrar, mover, cortar, *snap*, etc.) los diferentes objetos digitalizados

Label Permite etiquetar los objetos digializados

Customize Personaliza el aspecto de la pantalla de **v.digit** incluyendo cambios en el color de los objetos o poner capas raster o vectoriales como fondo

Toolbox Busqueda de polígonos abiertos, arcos repetidos, etc.

Window Selecciona los elementos que se visualizan en la pantalla de digitalización

Help Ayuda

Zoom Heeramientas de *zoom* y desplazamiento de pantalla. Son accesibles desde cualquiera de los submenús de **v.digit**

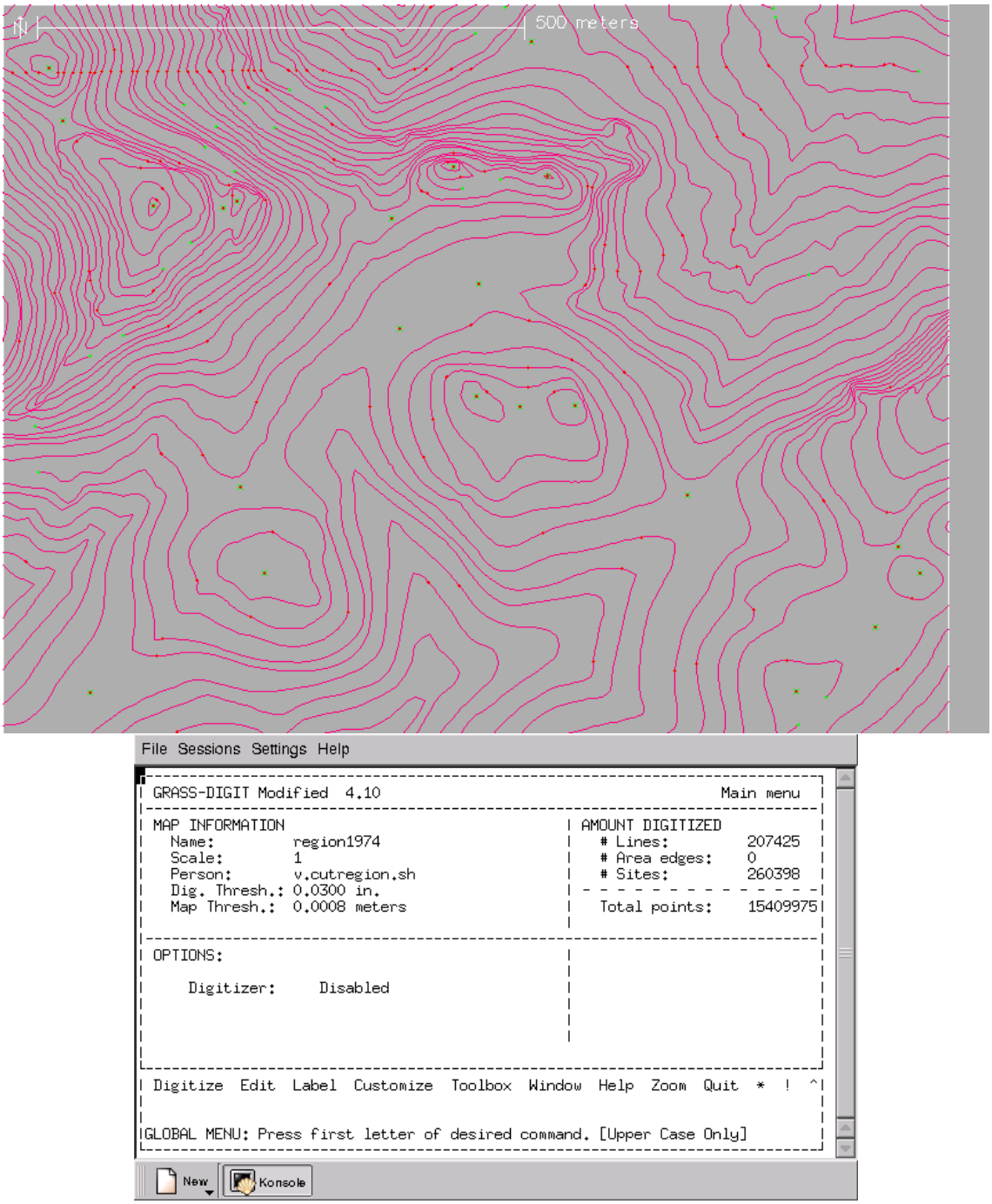


Figura 14: Pantalla de v.digit

3.2.3 Importación de otros formatos de SIG y CAD

En las antiguas versiones de GRASS, la importación o exportación de cada formato diferente implicaba un módulo diferente. Las nuevas versiones del programa, al utilizar las librerias GDAL y OGR para hacer las conversiones permiten que un sólo módulo se encarge de las mismas.

En el caso de la importación de ficheros raster este módulo es **r.in.gdal**. En la página web del proyecto[43] aparecen todos los formatos admitidos por la librería GDAL. El uso del módulo es tan sencillo como

GRASS: /path > r.in.gdal in=mapa.tif out=mapa<ENTER>

GRASS: /path > r.in.gdal in=dem.arc out=dem<ENTER>

El módulo **v.in.ogr** permite la importación de diversos tipos de datos vectoriales. Entre ellos los más utilizados son

- ESRI Shapefile
- Mapinfo File
- DGN
- GML
- PostgreSQL

3.3 ¿Como compilar GRASS?

Una vez que el usuario se siente suficientemente seguro con GRASS puede querer compilarlo el mismo, en general un programa compilado en nuestra propia máquina y siguiendo nuestras propias opciones será más eficiente. En el caso de GRASS, la distribución de código fuente incluye además el conjunto de herramientas necesarios para compilar módulos independientes programados por el usuario.

Una buena idea sería bajar el fichero el fichero de fuentes, por ejemplo **grass-5.0.3_src.tar.gz**, del sitio de GRASS y guardarlo en el directorio **/usr/local**. Una vez descomprimido y desempaquetado creará un directorio **grass-5.0.3_src** que contiene la distribución de fuentes completa. Entre los ficheros que contiene este directorio se encuentra el programa **configure**

3.3.1 configure y make

Son las dos herramientas básicas para compilar un programa en Linux. El programa **configure**, se utiliza en las distribuciones de código fuente de Unix y sirve para determinar las características de la máquina y preconfigurar, en función de estas, la compilación; permite además la introducción de diversas opciones por parte del usuario.

Si **configure** determina que puede compilarse el programa, de acuerdo con las especificaciones del usuario y las características de la máquina, escribirá un fichero **makefile** que será leido por el comando **make**; en casos contrario aborta con un mensaje de error.

Por su parte **make** es un programa cuyo propósito es facilitar la compilación de programas complejos. Para ello se apoya en el fichero **makefile** que describe las relaciones entre los ficheros del código fuente, las librerías que hay que enlazar y las ordenes necesarias para crear los ejecutables.

Cuando se ejecuta el programa **configure** de GRASS:

GRASS: /path > ./configure<ENTER>

GRASS is now configured for i686-pc-linux-gnu	
Source directory:	/usr/local/grass-5.0.3
Build directory:	/usr/local/grass-5.0.3
Installation directory:	/usr/local/grass5
C compiler:	gcc -g -O2
FORTRAN compiler:	g77
NVIZ:	yes
X11 support:	yes
DBM support:	no
JPEG support:	yes
TIFF support:	yes
PNG support:	yes
GD support:	yes
Tcl/Tk support:	yes
PostgreSQL support:	yes
OpenGL(R) support:	yes
ODBC support:	yes
FFTW support:	yes
BLAS support:	yes
LAPACK support:	yes
Motif support:	no
FreeType support:	yes
GLw support:	no
NLS support:	no
Readline support:	no

Tabla 5: Resultados del comando **configure** de GRASS

```
...
...
...
checking for Tk_MainWindow in -ltk... yes
checking whether to use PostgreSQL... yes
checking for location of PostgreSQL includes...
checking for libpq-fe.h... no
configure: error: *** Unable to locate PostgreSQL includes.
```

Tabla 6: Error del comando **configure** de GRASS

este chequea la existencia de diversos ficheros y librerías para terminar dando un mensaje de éxito similar a:

En la que nos dice cual es el directorio de los fuentes, el de compilación y en el que aparecerá finalmente instalado el programa; cuales son los compiladores de C y fortran que se van a utilizar y finalmente todas las opciones con las que se ha compilado y las que no.

También puede ser que **configure** se detenga a mitad del proceso de comprobación con un mensaje de error similar al de la tabla 6

es decir, en uno de los pasos de comprobación determina que alguno de los componentes necesarios no se encuentra, por tanto deberiamos proceder a instalarlo. Otra posibilidad es que el componente esté pero no donde **configure** lo busca, en tal caso deberíamos decirle donde está. Existe una tercera posibilidad que sería indicar a **configure** que no instale dicho componente.

Para ver en cada caso como se pueden enviar mensajes a configure, teclea:

GRASS: /path > ./configure –help<ENTER>

En primer lugar cabe destacar las opciones para indicar si la compilación debe dar soporte o no a determinadas funcionalidades:

```
--with-PACKAGE
--without-PACKAGE
```

por ejemplo

```
--with-tiff
```

da soporte a las librerias **tiff** y permitirá compilar los módulos **r.in.tiff** y **r.out.tiff**. En realidad –with-tiff es una opción por defecto por lo que no habrá que decir nada. Sin embargo hay otros casos en los que la opción por defecto es la contraria, por ejemplo si no decimos nada **./configure** se ejecuta con la opción

```
--without-freetype
```

pero freetype es una librería que nos permitiría compilar el módulo **d.text.freetype** que sirve para escribir en el monitor gráfico textos incluyendo eñes y acentos, por tanto como buenos hispanohablantes deberíamos seleccionar la opción contraria:

```
--with-freetype
```

En el caso de que **configure** deba instalar el soporte a una libreria concreta, deberá encontrar los ficheros que corresponden a la libreria y a los *includes* de la libreria (ficheros que contienen las especificaciones de las funciones y tipos de datos de una libreria). Si **configure** nos dice que no encuentra alguno de ellos y sabemos que están podemos necesitar las opciones de tipo:

[43] http://www.remotesensing.org/gdal/formats_list.html

```
--with-freetype-includes=DIRS
--with-freetype-libs=DIRS
```

donde DIRS hace referencia al directorio donde se hallan los archivos requeridos. En caso de que no tengamos instalados estos ficheros en nuestro sistema habrá que buscarlos e instalarlos o pasarle a **configure** la opción de no instalación.

Hay que tener en cuenta que cada distribución de Linux organiza los ficheros y directorios a su modo, por tanto las opciones de **configure** pueden variar de unas a otras. Una vez que hayas conseguido una ejecución con éxito de **configure**, es necesario compilar e instalar el programa:

GRASS: /path > make<ENTER>

GRASS: /path > make install<ENTER>

si no hay ningún problema el sistema habrá quedado listo para trabajar.

3.4 ¿Como manipular la región de trabajo?

En GRASS el área de trabajo se denomina **region** y los comandos para modificarla son básicamente **d.zoom** y **g.region**. Como ya se ha mencionado, la configuración de la región actual se almacena en el **MAPSET** de trabajo en el fichero **WIND**, mientras que la región por defecto se almacena en el fichero **DEFAULT_WIND** del **MAPSET PERMANENT**.

El comando **g.region** es bastante complejo y dispone de múltiples opciones y parámetros. Pueden distinguirse entre aquellos parámetros que modifican directamente las propiedades básicas de la región:

e= modifica la coordenada X del límite este de la región

w= modifica la coordenada X del límite oeste de la región

n= modifica la coordenada Y del límite norte de la región

s= modifica la coordenada Y del límite sur de la región

res= modifica la resolución o tamaño del pixel

nsres= modifica la resolución vertical o tamaño del pixel en sentido Norte-Sur

ewres= modifica la resolución horizontal o tamaño del pixel en sentido Este-Oeste

y aquellos parámetros que toman como parámetros de la región de trabajo los equivalentes de una capa espacial

rast= copia los parámetros de una capa raster

vect= copia los parámetros de una capa vectorial

sites= copia los parámetros de una capa de sites

puede también guardarse la región activa con un nombre (parámetro **[save=]**), cargarse una región previamente guardada (parámetro **[region=]**) o cargar la región por defecto con la opción **-d**. Puede obtenerse un listado de las regiones previamente guardadas con la orden:

GRASS: /path > **g.list region** <ENTER>

Las opciones **-p** y **-g** devuelve las características de la región activa con dos formatos diferentes (tabla 7). El primero es más apropiado para informar al usuario y el segundo para utilizarlo dentro de un *script*.

Finalmente la ejecución interactiva de **g.region** presenta una pantalla con la información acerca de la region (figura15) desde la que se puede modificar.

Otros comandos que modifican la región de trabajo son **d.zoom** y **d.pan**. Este último desplaza la región de trabajo por la pantalla.

projection: 1 (UTM) zone: 13 datum: nad27 ellipsoid: clark66 north: 4928010 south: 4913700 west: 589980 east: 609000 nsres: 30 ewres: 30 rows: 477 cols: 634	n=4928010 s=4913700 w=589980 e=609000 nsres=30 ewres=30

Tabla 7: Salida de g.region -p (izquierda) y g.region -g (derecha)

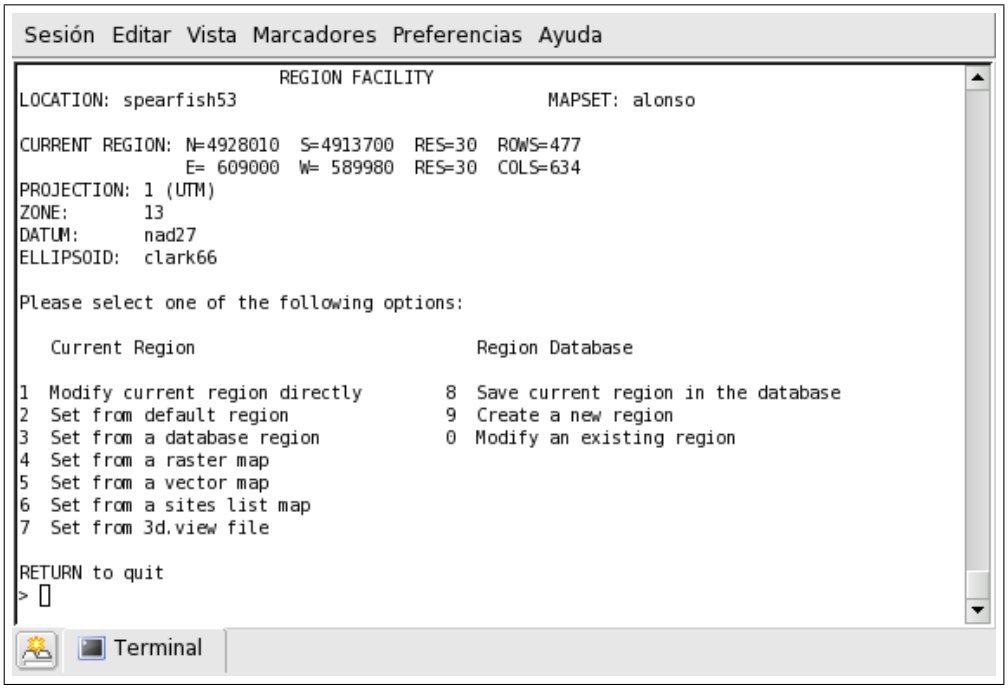


Figura 15: Ejecución en modo interactivo del módulo **g.region**

4 Programación con GRASS

Hoy en día se considera prácticamente imprescindible, en el trabajo con un SIG, disponer de un entorno de programación adecuado para ampliar las capacidades del sistema. En el caso de GRASS, el carácter modular del programa incrementa el potencial para programar, pudiendose emplear diversos modos de programación. Por otro lado la licencia GPL permite mantener el código libre mientras que protege los derechos de los autores al mismo tiempo que se potencia la transferencia de conocimiento.

La mayor parte de los módulos de GRASS se han programado en ANSI-C en entornos Unix utilizando el conjunto de librerías que proporciona el programa. Los scripts se programan utilizando la BASH shell de Unix. Como excepciones a esta regla general existen algunos módulos escritos en Perl y binarios desarrollados en Fortran77, estos últimos tienden a reescribirse en C para garantizar la portabilidad. Por otro lado se ha desarrollado una Interfaz Gráfica de usuario con lenguaje TclTk[44] (TclTkGRASS).

En GRASS se establecen, de este modo, cuato niveles de programación:

Funciones de librería el desarrollo a este nivel se lleva a cabo de manera centralizada por el equipo de desarrollo de GRASS. Se pretende poner a disposición de los programadores un amplio conjunto de funciones SIG de bajo nivel. Entre las más críticas destacan las funciones de lectura y escritura de datos que permiten mantener un entorno de programación consistente a pesar de los cambios en las estructuras de datos.

Programación de módulos binarios se estimula el desarrollo de módulos por parte de los usuarios siguiendo las siguientes directrices:

- Programación en ANSI-C
- Utilización de las librerías disponibles
- Creación de interfaces de usuario coherentes con el resto de los módulos

Los módulos así generados pueden enviarse para su aprobación por parte del Equipo de Desarrollo y su inclusión en posteriores versiones del programa o mantenerse como módulos aparte descargables por Internet

Scripts se trata de módulos generados utilizando cualquiera de las shells programables de que dispone Unix, aunque se recomienda utilizar la BASH (Bourne Again Shell). Puede tratarse de simples guiones para el desarrollo de una tarea rutinaria concreta o elaborados programas de propósito general debido a la gran potencia de BASH y al carácter modular de GRASS. El destino de los módulos resultantes puede ser el mismo que el de los binarios.

Desarrollo de interfaces para usuario en estos momentos sólo existe una linea de desarrollo madura. Se trata de TclTkGrass, una IGU programada con TclTk modular y fácimente personalizable.

A continuación se comentará brevemente las técnicas básicas de programación de módulos binarios, scripts e interfaces de usuario, obviando, por su complejidad, la programación de funciones. En los tres casos se tratará de una introducción necesariamente muy breve, dándose enlaces y referencias a sitios donde encontrar más información.

4.1 Código binario

El equipo de desarrollo de GRASS ha marcado unas lineas generales de como deben programarse nuevos módulos para GRASS:

- Utilizar el lenguaje C
- No acceder a los datos directamente sino a través de las funciones de librería correspondientes. De este modo se evitan problemas por posibles cambios en las estructuras de los ficheros

[44] http://www.tcl.tk/

```
#include “gis.h”
int main(int argc, char **argv){
      G_gisinit(argv[0]);
      printf(“‘Hola mundo de GRASS\n”);
}
```

Tabla 8: Un primer programa en C

- Utilizar los procedimientos de compilación propios de GRASS
- Utilizar las librerías de GRASS ya que permiten acelerar considerablemente el proceso de programación y mantienen la consistencia entre los módulos

Entre las librerías disponibles, las más relevantes y utilizadas son

- **gislib**. Es la más importante y la primera que se desarrollo. Contiene todas las rutinas necesarias para leer y escribir capas raster y sus diferentes ficheros de soporte; rutinas de acceso a la base de datos; métodos estandarizados para crear una interfaz de usuario; algunas de las funciones más básicas de gráficos y para el manejo de vectores y sites y otras funciones de propósito general como reserva de memoria, análisis de cadenas de caracteres, etc.
- **vectorlib**. Contiene rutinas para la gestión de datos vectoriales
- **displaylib**. Contiene funciones para realizar diversas operaciones gráficas y de conversión de coordenadas
- **rasterlib**. Contiene las primitivas de las operaciones gráficas de GRASS

Las directivas de preprocesador que deben incluirse en cualquier programa de GRASS que vaya a utilizar estas librerías son:

```
#include "gis.h"
#include "Vect.h"
#include "display.h"
#include "raster.h"
```

4.1.1 Inicialización y Compilación. El primer programa de GRASS

El sistema de programación de GRASS necesita inicializarse antes de empezar a llamar a sus funciones, para ello se utiliza la función

int G_gisinit (char *programa);

Esta función lee las variables de entorno de GRASS e inicializa una serie de variables ocultas utilizadas por otras funciones. Utiliza como único parámetro el nombre del módulo que se le debe pasar como **argv[0]**. Un primer programa en GRASS podría ser similar al que aparece en la tabla 8.

La compilación de estos programas resulta algo compleja para explicarla aquí. Puede consultarse el manual del programador de GRASS[45].

4.1.2 Como recopilar información acerca de la location y la región de trabajo

Existe un conjunto de funciones para consultar las variables de entorno relativas a la location y al conjunto de directorios relacionados con GRASS. En general los nombres de las funciones son bastante explicativos:

char *G_location();

[45] http://grass.itc.it/grassdevel.html#prog

```
#include "gis.h"

void main(int argc, char **argv){
          char location[30],mapset[30],gisbase[30],gisdbase[30];

          G_gisinit(argv[0]);
          printf("Hola mundo de GRASS\n");

          location=G_location();
          mapset=G_mapset();
          gisbase=G_gisbase();
          gisdbase=G_gisdbase();

          printf("Location : %s\n",location);
          printf("Mapset : %s\n",mapset);
          printf("Gisbase :%s\n",gisbase);
          printf("Gisdbase : %s\n",gisdbase);
          printf("Location path : %s\n",G_location_path());
}
```

Tabla 9: Consulta sobre las características de la LOCATION en un programa en C

char *G_mapset();

char *G_myname();

char *G_gisbase();

char *G_gisdbase();

char *G_location_path();

Todas ellas devuelven un puntero a una cadena de caracteres, recuerda que para dar a una variable el valor devuelto por estas funciones habrás tenido previamente que declarar esta variable y reservar en memoria el espacio suficiente para almacenar todos los caracteres que va a recibir. La tabla 9 muestra un ejemplo

Otras funciones que suministran información relevante son:

int G_projection(); devuelve un entero con el tipo de proyección.

char *G_database_projection_name(int proj); recoge este entero y devuelve una cadena de caracteres con el nombre de la proyección:

- **0** Filas y columnas
- **1** UTM
- **2** State Plane
- **3** Latitud-longitud
- **4** Otros

int G_zone(); en el caso de estar trabajando con la proyección UTM devuelve la zona en la que se encuentra el área de trabajo.

char *G_database_unit_name(int plural); que devuelve las unidades en que se mide el tamaño de celdilla. Si plural=1 las devuelve en plural si no en singular (eso si, las devuelve en inglés)

char *G_database_units_to_meters_factor(); que devuelve el factor de conversión de las unidades de la rejilla a metros.

Por tanto una manera de comunicar al usuario el tipo de proyección de la location en la que está trabajando sería con el programa que aparece en la tabla 10.

```c
#include "gis.h"

void main(int argc, char **argv){
          int zona;

          G_gisinit(argv[0]);
          zona=G_zone();
          printf("Bienvenido al mundo de GRASS. Proyección: %s \n"
                    , G_database_projection_name(G_projection()));
          if (G_projection()==1)printf("Zona %d UTM \n",zona);
}
```

Tabla 10: Consulta sobre las características de la LOCATION en un programa en C

```c
struct Cell_head{
     int format;      /* Máximo número de bytes por celdilla menos 1 */
     int compressed;  /* 0 = no comprimido 1 = comprimido, */
     int rows;        /* Número de filas */
     int cols;        /* Número de columna */
     int proj;        /* Proyección */
     int zone;        /* Zona UTM */
     double ew_res;   /* Resolución horizontal */
     double ns_res;   /* Resolución vertical */
     double north;    /* Límites de la capa */
     double south;
     double east;
     double west;
};
```

Tabla 11: Estructura Cell_head

4.1.3 Estructuras de Información Geográfica

Una de las características más potentes de C es el uso de estructuras, formadas por variables de diferente tipo, para almacenar objetos complejos, como pueden ser los objetos espaciales. Las librerías de GRASS incluyen diversas estructuras como *Cell_head* que almacena una región de trabajo (tabla 11) o bien la información de cabecera de un mapa raster; o *Map_info* que almacena mapas vectoriales completos e incluye otras estructuras que representan objetos geométricos (puntos, lineas, polígonos), así como sus relaciones topológicas.

Si queremos obtener las características de la región de trabajo actual o la región de trabajo por defecto deberíamos utilizar las funciones:

int G_get_window(struct Cell_head *region); lee la región de trabajo almacenada en el fichero WIND del MAPSET del usuario y la almacena en **region**.

int G_get_default_window(struct Cell_head *region); lee la región del fichero DEFAULT_WIND del mapset PERMANENT.

int G_put_window(sruct Cell_head *region); escribe **region** en el fichero WIND del mapset del usuario.

4.1.4 Transformaciones entre filas/columnas y coordenadas

int G_window_rows(); devuelve el número de filas de la región de trabajo.

int G_window_cols(); devuelve el número de columnas de la región de trabajo.

```
struct Option{
          char *key;          // Nombre del parámetro
          int type;           // Tipo de parámetro
          int required;       // Obligatorio u opcional
          int multiple;       // Multiples entradas
          char *options;      // Rango de valores válidos
          char *description;  // Descripción
          char *answer;       // Valor pasado por el usuario
          char **answers;     // Valores del usuario si multiple=YES)

          int count;
};

struct Flag{
          char key;           // Nombre de la opción
          char answer;        // Guarda el estado de la opción: 0/1
          char *description;  // Descripción
};
```

Tabla 12: Estructuras Option y Flag

double G_col_to_easting(double col, struct Cell_head *region); convierte el número de columna de una celdilla en la coordenada X correspondiente

double G_row_to_northing(double row, struct Cell_head *region); convierte el número de fila de una celdilla en la coordenada Y correspondiente

double G_easting_to_col(double easting, struct Cell_head *region); convierte la coordenada X de una celdilla en el número de columna correspondiente

double G_northing_to_col(double northing, struct Cell_head *region); convierte la coordenada Y de una celdilla en el número de fila correspondiente

4.1.5 Gestión de la linea de comandos

Los parámetros y opciones que el usuario pasa a un módulo a través de la linea de comandos se controlan en GRASS mediante dos estructuras, **Option** para los parámetros y **Flag** para las opciones, que permiten gestionar tanto la forma de pasar los parámetros como su evaluación.

struct Option *G_define_option(); para definir un parámetro

struct Option *G_define_flag(); para definir una opción

Una vez definido un parámetro u opción, es necesario definir los diferentes miembros de sus estructuras. Los miembros más relevantes de las estructuras **Option** y **Flag** aparecen en la tabla 12. La función encargada de procesar la linea de comandos y almacenar la información extraida en estas estructuras es:

int G_parser(int argc, char *argv[ );]

que toma los parámetros de la linea de comandos que se pasan a la función **main** en C. A continuación los diferentes parámetros pueden consultarse, lógicamente el más importante es opt->answer. Es una cadena de caracteres, por tanto es necesario pasarlo a entero o real si es de tipo numérico.

4.1.6 Manejo de ficheros raster

int G_open_cell_old(char *name, char *mapset); Abre un fichero raster preexistente a partir de su nombre y el **MAPSET** en que se encuentra, devuelve un descriptor del fichero.

int G_open_cell_new(char *name); Crea un nuevo fichero raster, devuelve un descriptor del fichero

CELL *G_allocate_cell_buf(); reserva memoria suficiente como para almacenar una fila de datos raster basandose en el número de columnas de la región.

int G_zero_cell_buf(CELL *buffer); rellena de ceros una fila de datos raster cuya memoria ha sido previamente reservada con **CELL *G_allocate_cell_buf();**

int G_get_map_row(int fd, CELL *cell, int nfila); Lee la fina número *nfila* del fichero cuyo descriptor es *fd* y la guarda en el puntero *cell* previamente reservada con **CELL *G_allocate_cell_buf();**

int G_close_cell(int fd); cierra un fichero raster abierto

int G_get_cellhd(char *name, char *mapset, struct Cell_head *cellhd); guarda en **cellhd** los datos de cabecera del fichero **name**. La librería gislib proporciona un tipo específico de estructura para el almacenamiento de los datos de cabecera de un fichero raster denominada **Cell_head**.

Puede encontrarse más información en el manual del programador de GRASS[46] y para la versión 5.7 las referencias de la API[47].

4.2 BASH Scripts

Para la programación de scripts no hay ningún tipo de limitación ya que cualquier lenguaje de scripts válido en Unix puede utilizarse (awk, PHP, TclTk, Python, etc.), sin embargo la mayor parte de los scripts se programan en el lenguaje de BASH, una de las shells de Unix. Una shell en Unix es un simple intérprete de comandos que dispone de un lenguaje propio, se trata por tanto de shells programables, una de las más utilizadas es la BASH (Bourne Again Shell) que incorpora las características de otros shells.

El lenguaje de la BASH (Mike G., 2000; FSF, 2002) permite combinar variables, comandos del sistema, bucles y estructuras condicionales, expresiones aritméticas y lógicas para llevar a cabo tareas específicas de forma automatizada y guardarlos en forma de programa ejecutable. La gran ventaja es que permite utilizar la gran cantidad de herramientas presentes en un sistema Unix como por ejemplo **sed**, **awk** o **cut** (Kernighan y Pike, 1987)

Bash reconoce dos tipos de comandos: los internos y cualquier programa ejecutable externo de que disponga el sistema, si se está ejecutando GRASS, los diferentes módulos del mismo son para BASH programas externos. Un módulo de GRASS programado como script de BASH hará, lógicamente, uso intensivo de los módulos del sistema. Una buena ayuda en la creaión de scripts para GRASS es la consulta de los scripts presentes en la distribución del programa en el directorio $GISBASE/scripts. Donde GISBASE es una variable del sistema en la que se almacena el nombre del directorio base de GRASS (por ejemplo /usr/local/grass53).

4.2.1 Como hacer un script con BASH

En el trabajo con un SIG modular es frecuente tener que encadenar diversas órdenes consecutivas que, en algún momento, será necesario repetir. Una buena idea para facilitar esta tarea, así como para mantener un histórico de las operaciones ejecutadas, es abrir un fichero de texto en el que se escriben las diferentes órdenes. Por ejemplo el conjunto de órdenes de la tabla 13.

adapta la región de trabajo a la capa raster **mde**, borra la pantalla gráfica, pinta el mapa **mde** y despues pinta el vectorial **rios** de color azul.

Ejecutar despues estas órdenes es tan simple como cortar y pegar de la ventana del editor a la ventana de trabajo. Sin embargo, pueden también ejecutarse directamente guardando el programa como un fichero y escribiendo en la consola de usuario:

[46] http://grass.itc.it/grassdevel.html#prog
[47] http://grass.itc.it/grass57/manuals/

```
g.region rast=mde
d.erase
d.rast mde
d.vect rios color=blue
```

Tabla 13: Conjunto de órdenes que forman el programa *guion*

GRASS: /path > **bash** *guion*<ENTER>

donde *guion* es el nombre que se ha dado al fichero. Sin embargo puede resultar más útil convertir este fichero en un ejecutable añadiendo al principio del mismo una linea que indique al sistema que el contenido del fichero es un guión y que lenguaje debe utilizarse para interpretarlo:

```
#!/bin/bash
```

Esto es válido para cualquier lenguaje de guiones (awk, perl, python, tcl, etc.). Posteriormente hay que convertirlo en ejecutable mediante la orden:

GRASS: /path > **chmod a+x** *guion*<ENTER>

4.2.2 Introducción de variables, parámetros y arrays

Definir una variable es tan sencillo como escribir una sentencia de asignación:

```
k=blue
```

Para recuperar el valor de la variable esta debe escribirse precedida por el símbolo $:

```
d.vect rios color=$k
```

Pueden utilizarse arrays, en primer lugar hay que declararlos y darles valor:

```
declare -a color
color=(red blue green black white)
```

y posteriormente utilizarlas:

GRASS: /path > d.vect rios color=${color[1]}<ENTER>

Un guión en BASH puede recibir también parámetros en linea de comandos sin más que escribir estos detras del nombre del programa, sepearados por espacios. Para hacer referencia a estos parámetros dentro del programa se utilizan las variables predefinidas $1, $2, ... que devuelven respectivamente los parámetos primero, segundo, etc.

Añadiendo todo esto al guión anterior quedaría algo así:

4.2.3 Sentencias de control

En BASH se dispone de un gran número de ordenes para construir condicionales y bucles:

- if [condición]; then sentencias elif [condición] else sentencias; fi
- case variable in patrón) sentencias;; patrón) sentencias;; ... esac

```
#!/bin/bash
declare -a color
color=(red blue green black white)
g.region rast=mde
d.erase
d.rast mde
d.vect rios color=$color[$1]
```

Tabla 14: Programa *guion* incorporando variables

```
#!/bin/bash
declare -a color
color=(red blue green)
declare -a mapa
mapa=(litologia Suelosb quipar)

d.erase white
for i in 0 1 2; do
     d.vect ${mapa[$i]} color=${color[$i]}
done
```

Tabla 15: Script de BASH incorporando un bucle for

- while [condición]; do sentencias; done
- until [condición]; do sentencias; done
- for variable in valores; do sentencias; done
- for ((expr; expr; expr)); do sentencias; done

El lenguaje de BASH incorpora un gran número de operadores lógicos, algunos de ellos comprueban la existencia y características de los ficheros, otros comparan cadenas de caracteres y finalmente otros comparan números. En la tabla 16 aparecen algunas de las expresiones lógicas que pueden utilizarse para crear condiciones. A continuación aparece un ejemplo de script que incorpora un bucle for.

El lenguaje de la shell dispone de otras muchas instrucciones propias de un lenguaje de alto nivel como puede ser la creación de menús (orden **select**) o de funciones (orden **function**).

operador	objetivo
-a fichero	Devuelve 1 si fichero existe
fic1 -nt fic2	Devuelve 1 si fic1 es posterior a fic2
-z cadena	Devuelve 1 si la longitud de cadena es 0
-n cadena	Devuelve 1 si la longitud de cadena es mayor que 0
cad1==cad2	Devuelve 1 si ambas cadenas de caracteres son iguales
cad1!=cad2	Devuelve 1 si las cadenas no son iguales
cad1 < cad2	Devuelve 1 si cad2 es anterior a cad1 en orden alfabético
cad1 > cad2	Devuelve 1 si cad2 es posterior a cad1 en orden alfabético
n1 -eq n2	Devuelve 1 si $n1 = n2$
n1 -lt n2	Devuelve 1 si $n1 < n2$
n1 -gt n2	Devuelve 1 si $n1 > n2$
n1 le n2	Devuelve 1 si $n1 <= n2$
n1 ge n2	Devuelve 1 si $n1 >= n2$

Tabla 16: Algunas de las expresiones condicionales disponibles en BASH

operador	objetivo
- +	Suma y resta
/ %	Producto división y módulo
**	Potencia
&& ||	y/o lógico

Tabla 17: Expresiones aritméticas disponibles en BASH

```
awk '   $1="Alicante"{print $1}
        $3+0>0 {print $0}
        {suma=suma+$2}
        END{print "suma=",suma}' fichero.txt
```

Tabla 18: Ejemplo de programa en AWK

4.2.4 Filtros

Unix dispone de diversas herramientas útiles para filtrado de flujos de texto. Estas son de gran utilidad cuando, por ejemplo se necesita utilizar la salida de un programa modificada como entrada para otro programa. Entre las más utilizadas destacan sed y awk (Kernighan y Pike, 1987).

Awk (Close *et al.*, 1995) es un lenguaje de programación orientado a la lectura de un flujo de datos (entrada estandar, fichero, tubería) que transforma y convierte en un flujo de salida (salida estandar, fichero, tubería). Este flujo está medianamente estructurado en registros y campos.

Puesto que todo programa de awk "sabe" que debe leer una serie de lineas de texto no es necesario programar nada para decirselo lo que simplifica notablemente el desarrollo de programas. Al leer el fichero de entrada linea por linea, compara estas con una serie de patrones, si encaja con estos ejecutará las acciones asociadas.

El ejemplo de la tabla 18 contiene cuatro patrones, (uno por linea):

$1="Alicante" se cumple si el primer campo es igual a "Alicante" (se cumple en la linea 4 de la tabla 19)

$3+0>3 se cumple si el tercer campo es numérico y mayor que 3 (se cumple en las lineas 2 y 4)

patrón vacio se cumple siempre

END se cumple tras leer la última linea del fichero de entrada (lógicamente también hay un patrón "BEGIN")

El resultado del programa de la tabla 18 sobre la entrada de la tabla 19) aparece en la tabla 20)

Algunos de los programas de GRASS generan una salida de texto que puede ser manipulada mediante este tipo de filtros. Por ejemplo **r.stats** o **r.report** que se utilizan para obtener estadísticos crudos de los valores que aparecen en una capa raster.

4.3 Interfaces gráficas de usuario (IGU)

Una de las críticas que tradicionalmente se han hecho a GRASS es el no disponer de una Interfaz Gráfica de Usuario adecuada. La principal preocupación durante la creación y evolución del programa fue más crear

Murcia	3	2	3	4
Albacete	3	4	5	3
Almería	3	3	2	
Alicante	4	5	2	2

Tabla 19: Flujo de entrada al programa en AWK

```
Albacete 3 4 5 3
Alicante
Alicante 4 5 2 2
suma 13
```

Tabla 20: Flujo de salida del programa en AWK)

```
#!/usr/bin/wish8.3
button .hola -text "Hola" -command puts "Hola Mundo!!!"
button .salir -text "Salir" -command exit
pack .hola -padx 20 -pady 10 -side left
pack .salir -padx 20 -pady 10
```

Tabla 21: Programa TclTk para la creación de una Intrfaz Gráfica de Usuario

un conjunto programable de herramientas para la gestión de espacios que crear un programa de cartografía automática, por ello el resultado es un conjunto de módulos sin una IGU que les de unidad.

La popularización de las herramientas SIG entre usuarios más preocupados por disponer de herramientas sencillas de visualización, consulta y cartografía automática; así como la multiplicación de módulos en GRASS llevó al desarrollo de una herramienta de menús que diera unidad al programa al mismo tiempo que un entorno más amigable. Para ello se utilizó el lenguaje TclTk.

Tcl (tool command language) es un lenguaje de scripts para controlar el manejo de aplicaciones introduciendo diversas herramientas sencillas de programaci'on. Tk es una extensión de Tcl que permite acceder al sistema Xwindows para construir interfaces gráficas de usuario mediante la inclusión de *widgets*, objetos gráficos (botones, menús, entradas de texto, etc.), en una ventana.

Juntos aportan al desarrollador:

- Un potente lenguaje de scripts para cualquier aplicación
- Programación de alto nivel y desarrollo rápido
- Excelente lenguaje para “unir” aplicaciones
- Facilidad para leer y personalizar otras aplicaciones escritas con TclTk

El ejemplo de la tabla 21 crea la ventana que aparece en la figura 16. Se trata simplemente de crear los componentes con la orden adecuada (button para un boton, label para una etiqueta, etc.) y pasarles los valores de las variables que los definen mediante el nombre de la variable precedido de un guión seguido del valor de la variable (*-text “Hola”*). Si el valor es una cadena de caracters incluyendo espacios se debe cerrar entre llaves.

El comportamiento de este programa es muy simple, si se pulsa en [Hola] aparece un mensaje en la consola, si se pulsa [Salir] el programa termina. Como puede verse, la primera linea es similar a la de los programas que se han hecho para BASH pero ahora cambiando el prorama intérprete del lenguaje.

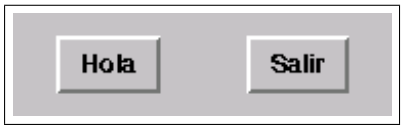


Figura 16: Ejemplo de IGU con TclTk

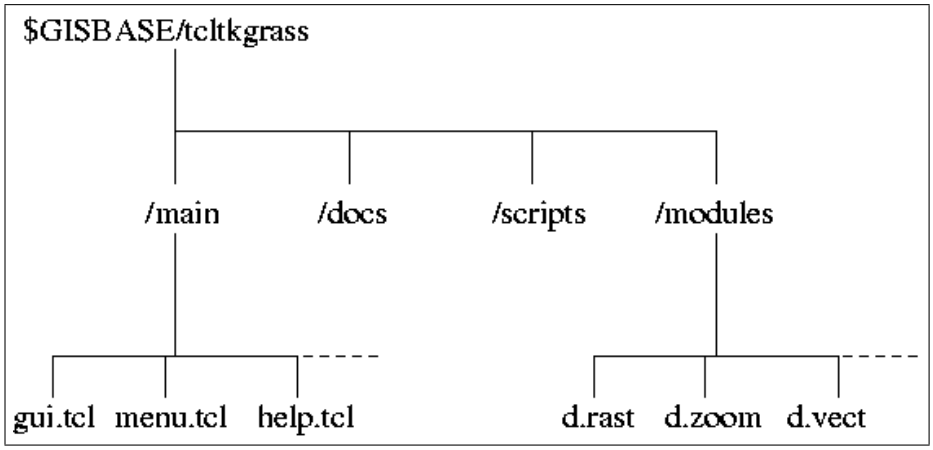


Figura 17: Estructura de directorios de tcltkgrass

```
        }
        -separator
        }
        }
        }
        -separator
        -separator
        }
        }
  }
```

Tabla 22: Sección de tcltkgrass que define el menú de ayuda

4.3.1 Personalización de TclTkgrass

La Interfaz Gráfica de Usuario TclTkgrass ha sido desarrollada siguiendo el mismo espíritu de modularidad que caracteriza a todo el programa, gracias a ello resulta fácimente personalizable. Todo el código de la interfaz se encuentra en un cnjunto de ficheros con extensión tcl en el directorio $GISBASE/TclTkgrass/main. El fichero $GISBASE/TclTkgrass/main/gui.tcl contiene los procedimientos para generar la IGU. El fichero $GISBASE/TclTkgrass/main/menu.tcl contiene la definición del menú de la ventana principal de TclTkgrass. Los códigos para crear las ventanas de diálogo de los diferentes módulos están en $GISBASE/TclTkgrass/modules (figura 17).

En un experimento de personalización sencilla de la interfaz, bastaría con modificar el fichero menu.tcl y añadir o modificar los ficheros en $GISBASE/TclTkgrass/modules.

Para añadir un nuevo módulo basta con editar el fichero TclTkgrass/main/menu.tcl.Este fichero contiene la definición de los diferentes módulos que forman TclTkgrass (figura). A continuación aparece el código que define la entrada *Help*.

Una simple traducción al español de la entrada permitirá la personalización (figura 19). Como se puede ver en el código mostrado, cada entrada de menú hace referencia a un fichero tcl que contiene la interfaz gráfica de los módulos concretos y que se activa cuando el usuario selecciona la opción correspondiente.

En la tabla 23 se muestra el fichero d.rast.tcl que contiene el código para la construcción de la ventana de diálogo.

Se trata de una llamada al procedimiento de TclTkgrass *interface_build* que crea un cuadro de diálogo con las propiedades que se le pasan como parámetros. Estos cuadros de diálogo permiten introducir las opciones (*checkbox*) y parámetros (*entry*) que se le pasan al correspondiente módulo de GRASS.

Cada módulo de GRASS tiene su correspondiente fichero tcl en el directorio $TCLTKGRASSBASE/module/,

```
interface_build {
    {d.rast} 0
    {Displays or overlays raster map layers.}
    {entry map {Raster map to be displayed:} 0 raster}
    {entry bg {Background color for null:} 0 color}
    {entry catlist {List of categories to be displayed (integer maps):} 0 ""}
    {entry vallist {List of values to be displayed (floating point maps):} 0 ""}
    {checkbox -o {Overlay (displays non-zero values only):} "" -o}
}
```

Tabla 23: Código TclTk para la IGU de d.rast

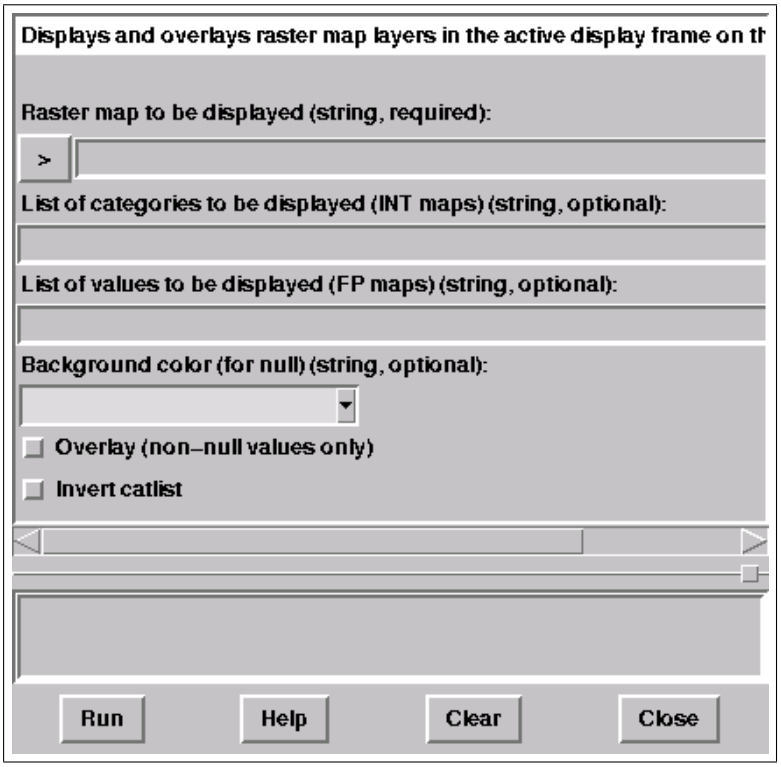


Figura 18: IGU de d.rast

```
interface_build {
    {d.rast} 0
    {Dibuja o superpone capas raster en pantalla.}
    {entry map {Capa raster que va a mostrarse:} 0 raster}
    {entry bg {Color del fondo:} 0 color}
    {entry catlist {Lista de categorias que se muestran (mapas enteros):} 0 ""}
     {entry vallist {Lista de valores que se muestran (mapas en coma floante):} 0 ""}
    {checkbox -o {Overlay (muestra sólo las celdillas no nulas:} "" -o}
}
```

Tabla 24: Código TclTk para la IGU de d.rast en español

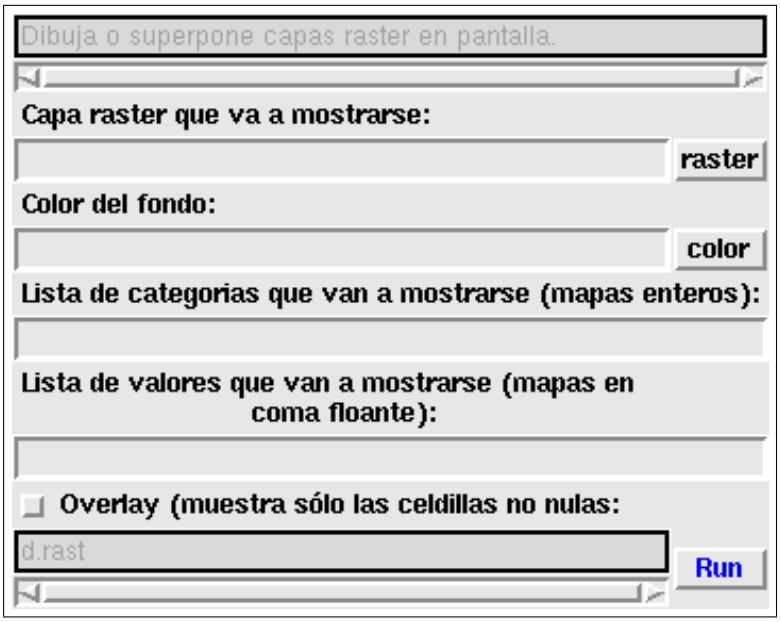


Figura 19: IGU de d.rast traducida al español

por lo que resulta sencilla su personalización, tanto por lo que se refiere al idioma (tabla 4.3.1 y figura 19) como por la posibilidad de programar interfaces más sencillas para usuarios noveles o para gestores que deben tomar decisiones en función de los resultados de un análisis SIG pero que no trabajan con el.

Actualmente se está planteando una modificación desde la raiz de esta intefaz de usuario. La posibilidad de crear una aplicación tipo *Desktop mapping* como ArcView dentro de/compatible con GRASS ha conducido a diversos debates en las listas de correo. Las principales cuestiones son:

- ¿Empezar desde cero con un programa independiente o modificar GRASS para que actue como *Desktop mapping*? Por el momento se adoptan ambas soluciones, GRASS 5.7 incluye el módulo **d.m** (display manager) que pretende ser un sencillo programa de *Desktop Mapping* escrito con TclTk (figura 20. Por otro lado se ha comenzado a utilizar el programa QGIS dentro de GRASS5.7.
- ¿Qué lenguaje utilizar? TclTkgrass produce interfaces de usuario bastante espartanas, aunque las extensiones han mejorado bastante el aspecto inicial. Las opciones más defendidas son Python con wxPython (como Thuban) y C++ con Gtk. Esta última opción, significaría dejar de utilizar lenguajes de script.

A pesar de las críticas sobre la pobreza de las interfaces de usuario en TclTk este lenguaje sigue desarrollandose y mejorando desde hace 16 años. Una de sus extensiones: Bwidget[48] permite introducir nuevos objetos en las Interfaces Gráficas de Usuario. Esta extensión se ha utilizado para programar el módulo d.dm de GRASS 5.7

[48] http://tcltk.free.fr/Bwidget/

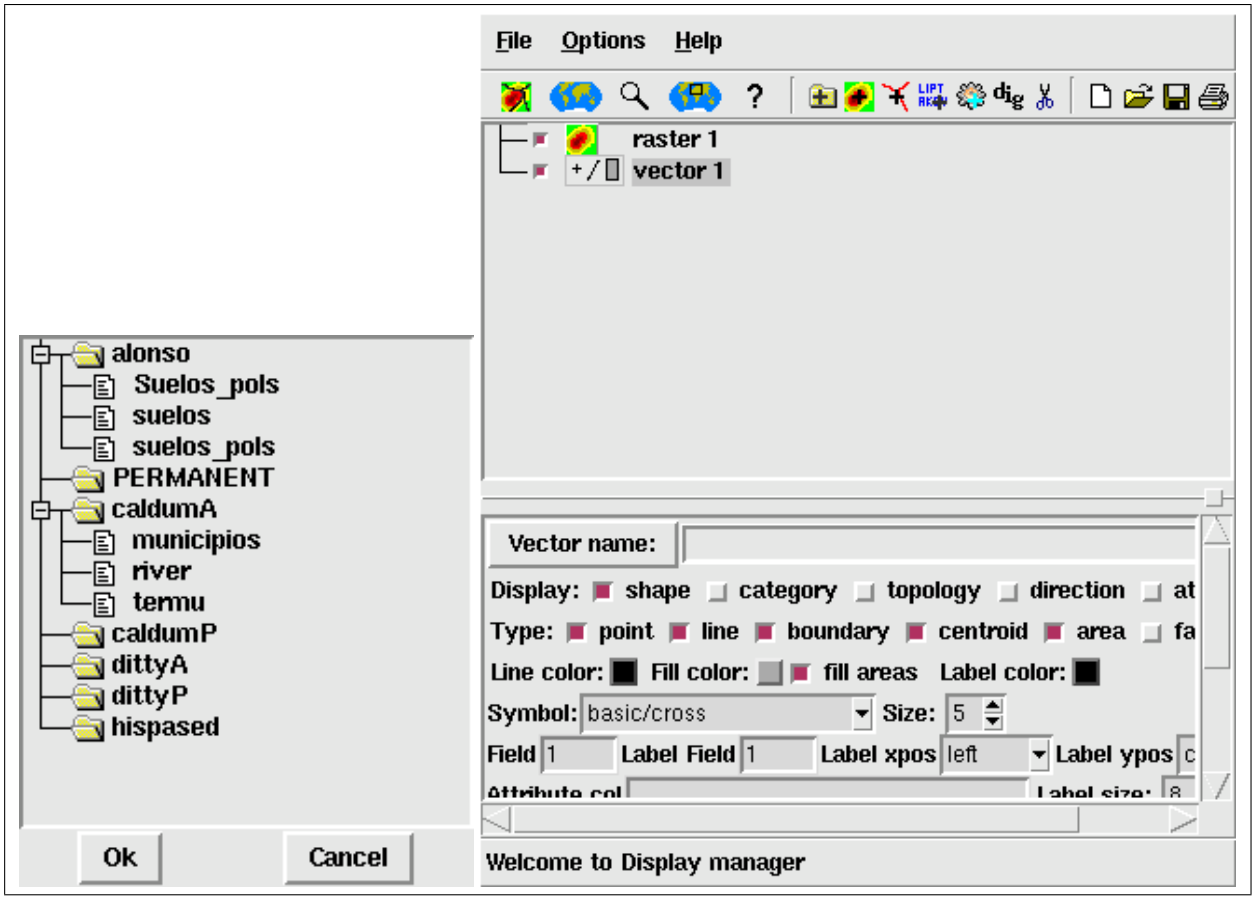


Figura 20: Interfaz Gráfica de Usuario d.m en GRAS 5.7

5 Integración de GRASS con otros programas

Una de las grandes ventajas de que GRASS sea un sistema abierto (es decir que su código sea público) es la facilidad con que puede interrelacionarse con otros sistemas abiertos. El conocimiento total que los diferentes desarrolladores pueden obtener del código de otros, así como el carácter público de las librerías de funciones facilita considerablemente la programación.

5.1 El programa de análisis de datos R

El lenguaje S, desarrollado en los laboratorios de la compañía AT&T por Rick Becker, John Chambers y Allan Wilks, pretende simplificar tanto el almacenamiento de datos como su tratamiento estadístico. De este lenguaje se han realizado varias implementaciones, algunas de ellas comerciales[49]. R es un programa de distribución libre, y puede utilizarse en distintos sistemas operativos. Si se suma a ello la facilidad de manejo derivada del propio lenguaje S, y la sencillez con la que pueden combinarse los distintos análisis estadísticos, tenemos en R el programa ideal para la utilización en el trabajo cotidiano.

Sintetizando, puede describirse a R como un entorno integrado para trabajar con el lenguaje S, que proporciona:

- Un conjunto coherente y extensivo de instrumentos para el análisis y el tratamiento estadístico de datos.
- Un lenguaje para expresar modelos estadísticos y herramientas para manejar modelos lineales y no lineales.
- Utilidades gráficas para el análisis de datos y la visualización en cualquier estación gráfica o impresora.
- Un eficiente lenguaje de programación orientado a objetos, que crece fácilmente merced a la comunidad de usuarios.

Una descripción detallada de R puede conseguirse en Venables *et al* (1999)[50], este texto así como información adicional, y el propio programa está disponible en la página oficial del proyecto R [51], para distintas plataformas (Unix, Linux, Window, Mac).

Para utilizar R bastará con invocarlo por su nombre: R, si esta ejecución se lanza desde una sesión de GRASS podremos conectar ambas herramientas.

5.1.1 Primeros pasos con R

Una vez iniciado el programa ofrecerá información básica sobre él y quedará a la espera de recibir una orden[52]:

```
R : Copyright 1997, Robert Gentleman and Ross Ihaka
Version 0.61.1 Alpha (December 21, 1997)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type "license()" for details.

Type "demo()" for some demos, "help()" for on-line help, or
"help.start()" for a HTML browser interface to help.
>_
```

Para abandonar el programa bastará con escribir `q()` y el resultado será:

[49] La aplicación comercial más conocida es S–plus.
[50] Notes on R: A programming environment for data analysis and graphics, recientemente traducido al castellano.
[51] `http://cran.r-project.org/`
[52] esta presentación puede variar ligeramente de una versión a otra del programa.

```
> q()
Save workspace image? [y/n/c]:
```

que nos permite guardar o abandonar los datos y las ordenes utilizadas durante la sesión de trabajo, o bien, cancelar la orden de abandonar el programa.

Para obtener ayuda del programa puede utilizarse la orden `help()` o `help.start()`. Para ver una sesión de demostración puede utilizarse la orden `demo()`. Las funciones de ayuda en R son:

- `help(), ?`: Proporciona ayuda sobre una palabra clave
- `help.start()`: Inicia consulta de la ayuda desde un navegador.
- `example()`: Muestra los resultados propuestos en el ejemplo.
- `demo()`: Muestra la relación de "demos" disponibles.
- `library()`: Muestra la relación de bibliotecas de funciones disponibles.
- `data()`: Muestra la relación de datos de ejemplo disponibles.
- `apropos()`: Muestra la relación de objetos disponibles con una cadena dada.

5.1.2 Expresiones en R

R propone una sintaxis muy sencilla en la que la unidad está constituida por la *expresión*. La expresión más sencilla es un simple número entero. Las más complejas combinan variables, valores numéricos y de texto, asignaciones y funciones.

```
> sqrt(3^2 + 5^2)
```

5.1.3 Operadores en R

En R tenemos los siguientes operadores que puede utilizarse:

`+, -, *, /`	suma, resta, producto, cociente
`%%, %/%, ^`	módulo, cociente entero, potencia
`==, !=, !`	igual, distinto, no
`>, >=, <, <=`	mayor que, mayor o igual que, menor que, menor o igual que
`|, ||, &, &&`	y, y, o, o
`<-, ->`	asignar a la izquierda, asignar a la derecha
`:`	generar una serie

Para obtener información de los distintos operadores puede solicitarse la ayuda mediante `?"+"`.

5.1.4 Variables en R

Los nombres de las variables deben comenzar obligatoriamente por una letra, distinguiéndose entre mayúsculas y minúsculas, y a continuación, opcionalmente, una combinación de letras y números, a ellos puede incorporarse el punto "."; así son nombre válidos: `a, A, A.1, altura, densidad,` ... Los nombre de las variables pueden coincidir con los nombre de funciones, aunque resulta poco aconsejable.

Las variables pueden contener valores sencillos, atendiendo a su naturaleza tenemos:

Lógicos:	TRUE,FALSE,T,F
Enteros:	-10, 1, 1000, ...
Precisión doble:	-10.1, 6.02310e24, ..., -Inf, Inf, NaN
Complejos:	1+3i, 1+0i, 9i, ...
Carácter:	"Hola", "Enero", "sin(x)", "pino", ...
Perdidos:	Na

también cabe la posibilidad de tener más de un valor en una variable, siendo entonces, según la estructura:

Vector	conjunto ordenado de datos del mismo tipo básico.
Array	vector con atributo de dimensión, es válido cualquier número de dimensiones.
Matriz	es un array con dos dimensiones
Factor	tipo especial de vector en el que se codifican las clases.
Lista	conjunto de elementos que pueden ser de distintos tipos.
Estructura de datos	mezcla de matriz y lista.

5.1.5 Funciones en R

R dispone de un gran número de funciones que se incorporan mediante bibliotecas de funciones denominadas `library`[53], si bien están agrupadas en base a bibliotecas que deben ser cargadas especificamente (mediante la función `library()`), las más habituales están en una biblioteca (`base`) que esta disponibe al arrancar el programa.

Funciones básicas	`c()`	Concatenar los elementos que se indican, separados por comas.
	`seq()`	Generar una secuencia numérica.
	`rep()`	Generar un conjunto de valores repetidos.
	`t()`	Transponer una matriz.
	`sqrt()`	Raíz cuadrada.
	`abs()`	Valor absoluto.
	`sin(), cos(), ...`	Funciones trigonométricas.
	`log(), exp()`	Logaritmo y exponencial.
	`round()`	Redondeo de valores numéricos.
	`ls()`	Relación de objetos disponibles.
	`rm()`	Elimina uno o varios objetos.
	`for(), while()`	Evalúa una o un conjunto de expresiones repetitivamente.
	`if(), ifelse()`	Evalúa una expresión condicionalmente.

Funciones gráficas básicas	`plot()`	Función genérica para epresentar en el plano xy puntos, lineas, etc.
	`lines()`	Añade líneas a un gráfico.
	`points()`	Añade puntos a un gráfico.
	`segments()`	Añade segmentos a un gráfico.
	`arrows()`	Añade flechas a un gráfico.
	`polygons()`	Añade polígonos a un gráfico.
	`rect()`	Añade rectángulos un gráfico.
	`abline()`	Añade una recta de pendiente e intersección dada.
	`curve()`	Representa una función dada.

Funciones gráficas básicas avanzadas	`piechart()`	Diagramas de sectores.
	`boxplot()`	Diagramas de *box-and-whisker*.
	`stripplot()`	Similares a `boxplot() con puntos`.
	`hist()`	Histogramas.
	`barplot()`	Diagramas de barras.
	`sunflowerplot()`	Representación en el plano xy de diagramas de girasol.
	`qqnor()`	Diagramas de cuantil a cuantil frente a la distribución normal.
	`qqplot()`	Diagramas de cuantil a cuantil de dos muestras.
	`qqline()`	Representa la línea que pasa por el primer y el tercer cuartil.

[53] estas pueden ser desarrolladas por los propios usuarios

Funciones estadísticas (1)

`mean()`	Media.
`sd()`	Desviación típica.
`var()`	Varianza.
`mad()`	Desviación absoluta media.
`max()`	Valor máximo.
`min()`	Valor mínimo.
`length()`	Número de elementos.
`range()`	Rango.
`sum()`	Suma de los elementos.
`prod()`	Producto de los elementos.
`IQR()`	Recorrido intercuartílico.

Funciones estadísticas (2)

`cumsum()`	Suma acumulada
`cumprod()`	Producto acumulado
`cummax()`	Máximo acumulado
`cummin()`	Mínimo acumulado
`sort()`	Ordena el vector.
`rev()`	Invierte el orden del vector.
`order()`	Indica el índice de orden del elemento.
`rank()`	Convierte el valor a rango.
`cov()`	Matriz de covarianzas.
`cor()`	Matriz de correlaciones.
`density()`	Determina la densidad en la distribución.
`table()`	Calcula la tabla de contingencia.

Leyes estadísticas y funciones (1)

`dnorm()`	Valor de la función de densidad de la normal.
`pnorm()`	Probabilidad encerrada por la curva normal.
`qnorm()`	Valor de z para una probabilidad dada.
`rnorm()`	Valor normal aleatorio.

Leyes estadísticas y funciones (2)

`rbeta()`	Gauss(normal)
`rbinon()`	Binomial
`rcauchy()`	Cauchy
`rchisq()`	χ^2 o Pearson
`rexp()`	Exponencial
`rf()`	F-Snèdecor
`rgamma()`	Γ
`rgeom()`	Geométrica
`rhyper()`	Hipergeométrica
`rlogis()`	Logística
`rlognorm()`	Logarítmico-normal
`rnbinom()`	Binomial negativa
`rpois()`	Poisson
`rt()`	T-Student
`runif()`	Uniforme
`rweibull()`	Weibull

Bibliotecas de R en CRAN[54] relacionadas con el análisis de datos espaciales y SIG

En CRAN pueden encontrarse más de 400 bibliotecas para R, las siguientes 22 (más del 5%) se han seleccionado por su relación con los SIG.

ade4	análisis multivariante
adehabitat	análisis de habitat
fields	herramientas para datos espaciales
geoR	análisis geoestadístico
geoRglm	modelos espaciales generalizados
GRASS	Interfaz con el programa GRASS v. 5.0
gstat	modelado, predicción y simulación geoestadísticos uni y multivariantes
lattice	gráficos de lattice
maps	representación de mapas geográficos
mapdata	datos adicionales para maps
mapproj	proyecciones cartográficas
maptools	herramientas para leer y manejar ficheros shape
PBSmapping	representación de mapas e implementación de otros procedimientos SIG
RArcInfo	funciones para la importación de datos en formato binario de Arc/Info V7.x
rgdal	comunicaciones con la biblioteca de abstracción de datos geoespaciales (gdal)
RODBC	acceso a bases de datos mediante ODBC
ROracle	interfaz para Oracle
sgeostat	infraestructura orientada a objetos para modelado geoestadistico con S+
shapefiles	lectura y escritura de ficheros shape de ESRI
spatstat	análisis, ajuste de modelos y simulación de patrones espaciales de puntos
splancs	análisis espacial y temporal de patrones de puntos
tripack	triangulación de datos espaciales irregulares

5.1.6 R y GRASS

Para conseguir que R pueda leer y escribir mapas de GRASS, debemos ejecutar R en un terminal de texto que este en GRASS.

1. Una vez abierto R ejecutamos `library(GRASS)` para cargar la libreria.
2. Para capturar la región de trabajo que se ha definido en GRASS utilizamos la orden `G<-gmeta()` que lee las características de la región y crea un objeto-región de R con ellas.
3. Puede obtenerse un resumen de las características de la región mediante `summary(G)` o representar la región mediante `plot(G)`.
4. Un listados de ficheros se obtiene mediante la función `get.mapsets()`, que devuelve los `mapsets` accesibles. Mediante `list.grass(type = "cell" || "dig_plus" || "site_lists")` se obtiene un listado de los mapas disponibles del tipo especificado (en este caso se han indicado varios tipos mediante una operación lógica).
5. Para realizar la lectura y escritura de ficheros de GRASS, ya que R siempre trabaja con una imagen de la información de la base de datos, se utilizan las ordenes: `rast.get()` y `rast.put()`, para traer la información de la ventana activa de un mapa raster a R, y viceversa (si no se alamcenan los resultados GRASS no podrá acceder a ellos); `sites.get()` y `sites.put()`, análogas a las anteriores con datos de puntos.

[54] http://cran.at.r-project.org

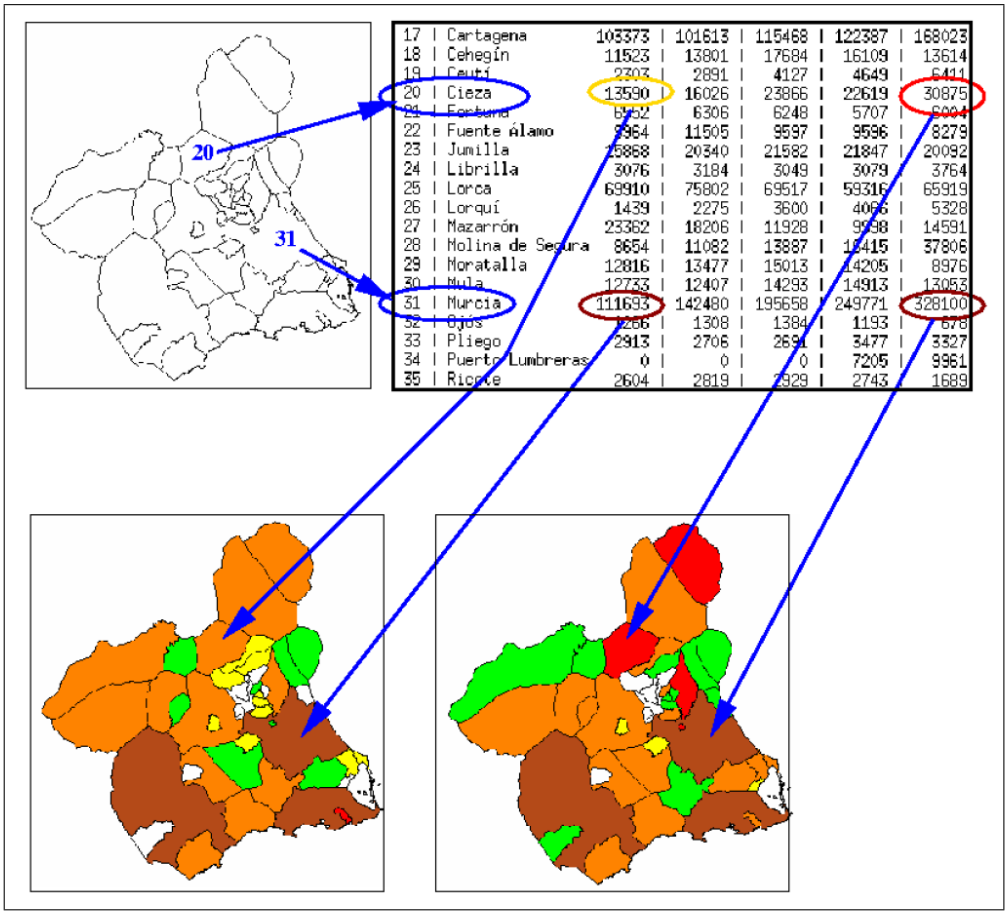


Figura 21: Modelo georrelacional de bases de datos

Los ficheros raster se almacenan como un objeto específico que en realidad es una ristra de valores numréricos. Por contra, los ficheros de sites como un data.frame.

5.2 El programa de gestión de bases de datos PostgreSQL

Gran parte de la información necesaria en el trabajo con un Sistema de Información Geográfica no es necesariamente información espacial sino temática. La forma óptima de almacenarla es mediante **Bases de Datos Relacionales (BDR)**. El lenguaje estandarizado para hacer consultas a una BDR es el **SQL (Standarized Query Language)**. El enlace de un mapa y una BDR se hace mediante un identificador común constituyendo el denominado **modelo de base de datos geo-relacional** (figura 21)

La gran limitación de SQL en relación con los SIG es su incapacidad para procesar consultas espaciales. Una solución parcial a este problema sería generar las consultas incluyendo condiciones espaciales (tabla 25 arriba) y procesar los resultados de la consulta mediante funciones espaciales en un SIG (tabla 25 abajo). En la figura 22 aparece el resultado de este ejemplo.

Otra alternativa desarrollada en los últimos años es la utilización de **bases de datos objeto-relacionales**. Se trata de un modelo en desarrollo en cuya definición trabaja, entre otros, el Open GIS Consortium. Este nuevo modelo admite tipos de datos abstractos que incluyen objetos geométricos, al mismo tiempo se define un SQL extendido lo que permite funciones y operadores espaciales.

La tabla 26 muestra un ejemplo incluyendo el operador espacial **within** que corresponde al SQL extendido de PostGIS. El resultado es el mismo que aparece en la figura 22.

Las ventajas básicas de este modelo son el permitir el acceso concurrente a la base de datos; la inclusión de geometría, topología y datos temáticos en una sola base de datos; y la disponibilidad de operadores y

```
SELECT ident,x,y
   FROM observatorios
   WHERE x>'$oeste' and x<'$este' and y>'$sur' and y<'$norte' ;
```

si ([x,y]=>caravaca) { pinta [x,y] }

Tabla 25: Ejemplo de consulta espacial con una base de datos Geo-Relacional

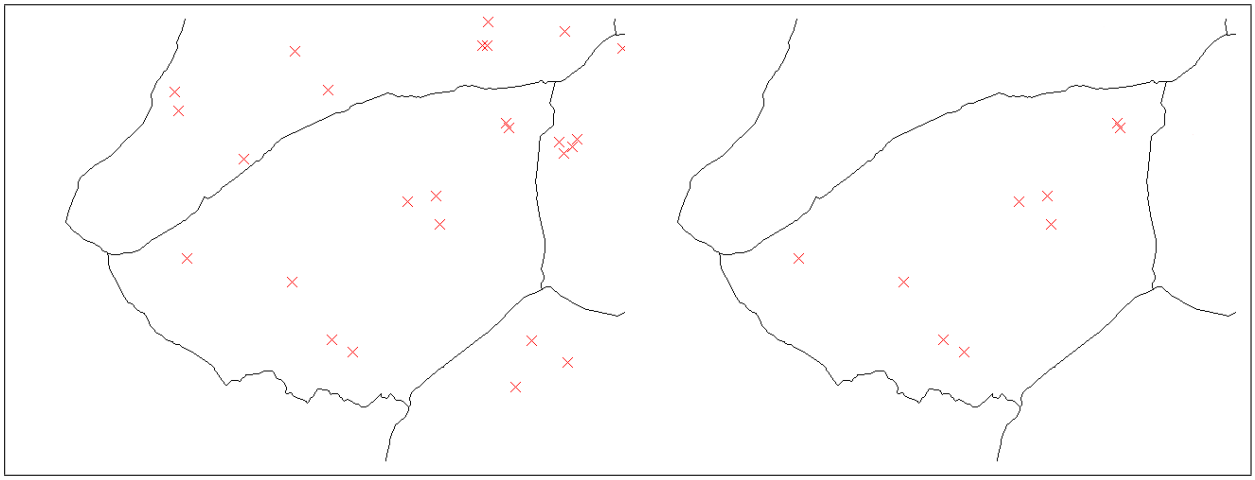


Figura 22: Consulta espacial en dos pasos en una base de datos georrelacional (ver tabla

```
SELECT o.ident,o.punto
   FROM observatorios o, municipios m
   WHERE within(o.punto,m.poligono) AND m.nombre='Caravaca';
```

Tabla 26: Ejemplo de consulta espacial con una base de datos Objeto-Relacional (PostGIS)

```
Welcome to psql 7.3.4, the PostgreSQL interactive terminal.

Type:  \copyright for distribution terms
       \h for help with SQL commands
       \? for help on internal slash commands
       \g or terminate with semicolon to execute query
       \q to quit

murcia=#
```

```
                 Listado de relaciones
 Schema |     Nombre      | Tipo  | Dueño
--------+-----------------+-------+--------
 public | edades          | tabla | alonso
 public | general         | tabla | alonso
 public | municipios      | tabla | alonso
 public | olivares        | tabla | alonso
 public | poblacion       | tabla | alonso
 public | puntos_prot_civ | tabla | alonso
 public | sectores        | tabla | alonso
 public | uso_suelo       | tabla | alonso
(8 filas)
```

Tabla 27: Listado de tablas que aparecen en una base de datos

funciones geométricos y topológicos

5.2.1 GRASS 5.0/5.3 y PostgreSQL como Base de Batos GeoRelacional

En GRASS existen diversos comandos para enlazar mapas vectoriales y bases de datos. Hay varios proyectos en marcha para la utilización de diferentes sistemas de gestión de base de datos *Open Source* con GRASS. Nos centraremos en PostgreSQL, uno de los SGBD de código abierto más populares, completos y flexibles pero a la vez sencillas de manejar como usuario.

La gestión de bases de datos se basa en la existencia de un **programa servidor**; que organiza los datos, recibe las consultas, las ejecuta y las devuelve; y un **programa cliente** que el usuario ejecuta y que lanza las consultas creadas por este al servidor, el programa cliente y el servidor no tienen siquiera que ejecutarse en el mismo ordenador. Por tanto para conectarnos a la base de datos lo primero que hay que hacer es ejecutar un programa cliente:

GRASS: /path > **psql murcia** <ENTER>

En este caso se ha invocado al programa cliente de bases de datos de PostgreSQL especificando la base de datos a la que queremos conectarnos. El mensaje de bienvenida de psql será algo parecido a:

Disponemos de una serie de comandos formados por una barra y una letra que realizan operaciones sencillas:

- \h para pedir ayuda sobre comandos SQL
- \¿ para pedir ayuda sobre los comandos de *barra y letra*
- \q para salir del programa
- \d para obtener un listado de las tablas que forman la base de datos o de als columnas que forman una tabla

```
                    Tabla "public.general"
          Columna          |  Tipo     | Modificadores
---------------------------+-----------+---------------
| ident                    | int4      |             4 |
| municipio                | varchar   |            60 |
| poblacion                | int4      |             4 |
| variacion9698            | float8    |             8 |
| matrimonios              | int4      |             4 |
    ...                      ...                  ...
| presupuestos             | int4      |             4 |
| hoteles                  | int4      |             4 |
| extrahotelera            | int4      |             4 |
```

Tabla 28: Listado de campos que aparecen en una tabla

La tabla *municipios* contiene los nombres de los términos municipales; la tabla *población* la de la evolución de la población entre 1900 y 1991; la tabla *uso_suelo* los usos del suelo por municipio; la tabla llamada *general* contiente, finalmente, un resumen de indicadores socioeconómicos de la Región de Murcia. Puede comprobarse cuales son las variables que contiene cada una de ellas con el comando **d** seguido del nombre de la tabla. Por ejemplo la orden **d general** produce el siguiente resultado:

Como puede verse, aparecen 3 tipos de variables. El tipo *int4* corresponde a números enteros, el tipo *float8* corresponde a numeros reales y *varchar* a cadenas de caracteres. De todos estos atributos el más importante es **ident** ya que asigna a cada municipio un identificador único que coincide con el identificador del polígono correspondiente a dicho municipio en el mapa vectorial.

A partir de aquí hay que continuar utilizando el lenguaje SQL que, evidentemente, no es el objeto de este curso. Puede consultarse la numerosa bibliografía al respecto (Cárdenas Luque, 2001; PGDP, 2002; Worsley y Drake, 2002).

En GRASS existen diversos comandos para enlazar capas de información espacial y bases de datos. Hay varios proyectos en marcha para la utilización de diferentes sistemas de gestión de base de datos *Open Source* con GRASS. PostgreSQL es un SGBDOR en la vanguardia del desarrollo de software abierto. Las versiones antiguas de GRASS (hasta la 5.7) incluyen un conjunto de módulos desarrollado en 2000 por Alex Shevlakov (Neteler, 2003) para integrar GRASS y PostgreSQL en modo geo-relacional:

- Conexión y exploración de la base de datos:

  g.select.pg conecta GRASS con una base de datos almacenada en PostgreSQL

  g.table.pg devuelve un listado de las tablas disponibles en la base de datos seleccionada

  g.column.pg devuelve un listado de las columnas disponibles en una tabla

  g.stats.pg calcula los estadísticos básicos de una columna

- Visualización:

  d.rast.pg Permite visualizar pixels cuyo identificador corresponda a un objeto (registro en la base de datos) que cumplen determinadas condiciones

  d.site.pg Permite visualizar puntos cuyo identificador corresponda a un objeto (registro en la base de datos) que cumplen determinadas condiciones

  d.vect.pg Permite visualizar puntos cuyo identificador corresponda a un objeto (registro en la base de datos) que cumplen determinadas condiciones

- Consulta espacial interactiva

  d.what.r.pg Permite pinchar en un mapa raster y obtener la entrada en la base de datos correspondiente al valor almacenado en la celdilla pinchada

  d.what.s.pg Permite pinchar en un mapa de sites y obtener la entrada en la base de datos correspondiente al identificador del site pinchado

```
declare -a r
r=(240 180 150 120 90 60 30 0)
declare -a g
g=(0 30 60 90 120 150 180 240)
b=0
declare -a pr
pr=(0 1000 2500 5000 10000 25000 50000 100000)
declare -a si
si=(1000 2500 5000 10000 25000 500000 5000000)

for i in $(seq 0 6);do
      d.vect termub fcolor=${r[$i]}:${g[$i]}:$b
             lcolor=black where="poblacion>=${pr[$i]} and poblacion<${si[$i]}"
done
```

Tabla 29: Programa para pintar polígonos en función de los resultados de consultas SQL

d.what.v.pg Permite pinchar en un mapa vectorial y obtener la entrada en la base de datos correspondiente al identificador del objeto pinchado

- Importación:

  v.in.shape.pg Importa ficheros shape guardando la base de datos en postgres

  v.in.arc.pg Importa ficheros arcinfo guardando la base de datos en postgres

  pg.in.dbf Importa bases de datos en formato DBF a PostgrSQL

- Módulos para crear mapas con categorías obtenidas de la base de datos:

  r.reclass.pg cuyos identificadores se obtienen de una columna o índice en una tabla de la base de datos.

  r.rescale.pg crea un mapa de polígonos cuyos identificadores se obtienen reescalando los valores de una columna o índice a un nuevo rango de valores.

Curiosamente no existen módulos para lanzar consultas en SQL directamente a la base de datos. Esto se puede hacer en linea de comandos, por ejemplo con la orden:

GRASS: /path > echo 'select * from arable where x<656000;'|psql suelos2<ENTER>

que envía al servidor de base de datos la consulta entrecomillada para base de datos suelos2. El resultado de la consulta se manda directamente a la consola de usuario donde se puede manejar con cualquiera de las utilidades de Unis para procesar flujos de texto, por ejemplo puede ser filtrada con un programa de **awk**.

5.2.2 GRASS 5.7 y PostgreSQL

La versión 5.7 de GRASS incluye nuevos módulos para llevar a cabo la conexión a una base de datos. Por otra parte se incorpora a los módulos de visualización (**d.rast**, **d.vect** y **d.sites**) la posibilidad de incluir consultas SQL para dibujar sólo aquellos objetos que cumplan determinadas condiciones. Así por ejemplo la orden:

GRASS: /path > **d.vect** termub where="poblacion>=25000 and poblacion<50000" fcolor=red <ENTER>

genera como resultado la figura 23.

Esta nueva versión de d.vect permite además modificar definir los colores en formato RGB, tal como se aprecia en la figura 24 que es el resultado de ejecutar un script de BASH con un bucle **for** que ejecuta la orden para diferentes rangos de valores de la variable **poblacion** y diferentes combinaciones de color. En el centro de dicho script esta la orden:

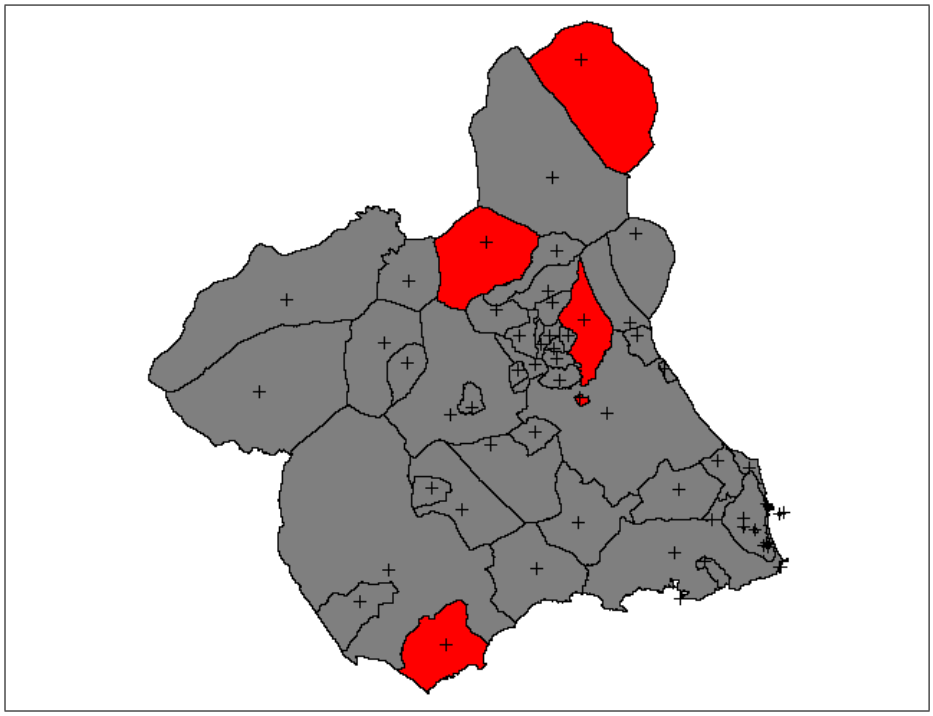


Figura 23: Visualización del mapa de polígonos de la Región de Murcia en función de los resultados de una consulta SQL

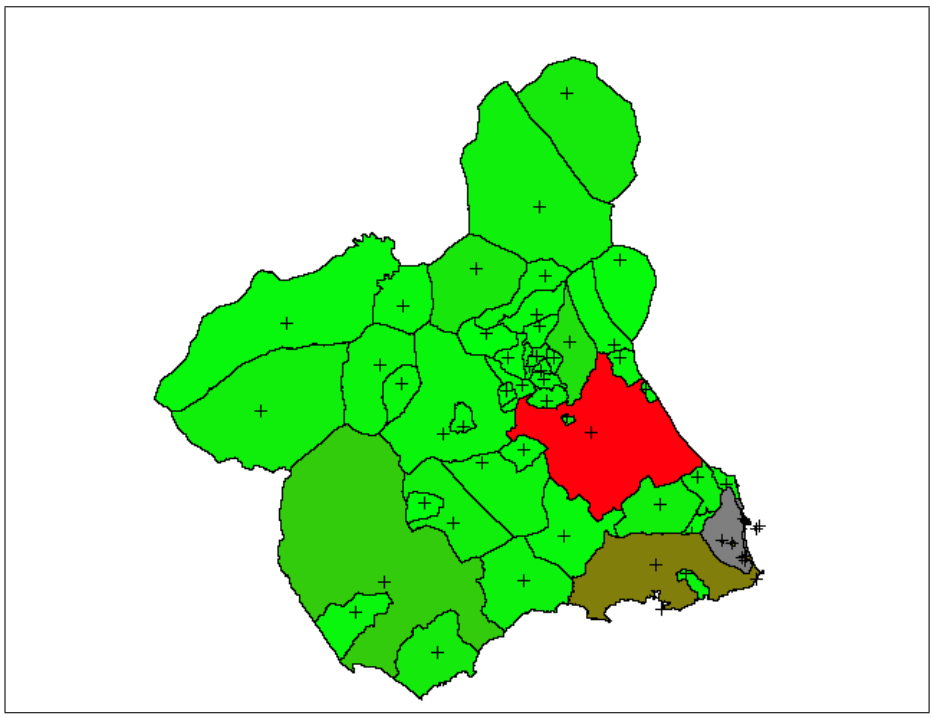


Figura 24: Visualización del mapa de polígonos de la Región de Murcia en función de los resultados de una consulta SQL

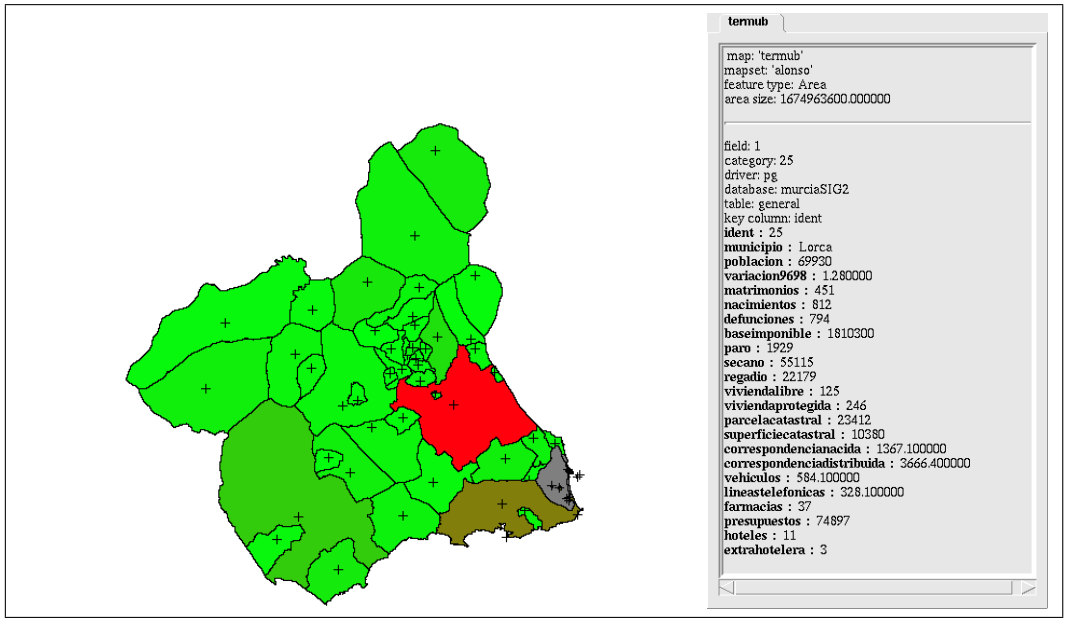


Figura 25: Resultados de una consulta con **d.what.vect** en GRASS 5.7

En el script de BASH que aparece en la tabla 29 los símbolos “$” indican que se trata de variables previamente definidas. De este modo puede programarse un script para pintar diferentes polígonos de diferentes colores en función de los valores que adopta una variable devuelta por una consulta SQL.

Los módulos de consulta (**d.what.rast**, **d.what.vect** y **d.what.sites** muestran también el resultado de una consulta a la base de datos en lugar de un simple identificador tal como se aprecia en la figura 25. En lugar de presentar los resultados en la consola de texto lo hacen en forma de tabla generada con TclTk.

db.connect permite al usuario establecer la conexión a una base de datos por defecto.

GRASS: /path > db.connect driver=dbf
database=’$GISDBASE/$LOCATION_NAME/$MAPSET/dbf/’<ENTER>

GRASS: /path > db.connect driver=pg database="host=myserver.itc.it,
dbname=mydb, user=name"<ENTER>

En el primer ejemplo la base de datos está en el mismo ordenador, en el mapset del usuario, en el segundo la base de datos se aloja en un servidor.

v.db.connect establece o devuelve la conexión a base de datos de un determinado mapa.

GRASS: /path > v.db.connect -p map=vectormap<ENTER>

GRASS: /path > v.db.connect -c map=vectormap<ENTER>

GRASS: /path > v.db.connect map=vectormap table=tabla field=1 key=ident driver=dbf
database=’$GISDBASE/$LOCATION_NAME/PERMANENT/dbf’<ENTER>

db.copy copia una base de datos en un formato a otro formato.

db.tables -p hace un listado de todas las tablas

db.columns hace un listado de todas las columnas de una determinada tabla

db.describe se consigue una descripción completa de cada una de las columnas de una tabla

db.select lanza una consulta SQL al servidor de bases de datos.

db.execute permite ejecutar cualquier orden SQL

5.2.3 GRASS y PostGIS

PostGIS es una extensión de PostgreSQL que permite el manejo de objetos geoespaciales. Permite utilizar todos los objetos de la especificación del Open GIS Consortium constituyendo una base de datos espacial como SDE de ESRI o Oracle Spatial. Existe un manual de PostGIS (Ramsey).

Para crear una base de datos capaz de manejar PostGIS hay que seguir los siguientes pasos:

1. Crear la base de datos y entrar en ella:

   GRASS: /path > createdb murcia;psql murcia<ENTER>

2. Crear el intérprete del lenguaje pl/pgsql

   psql=# create function plpgsql_call_handler () returns opaque as '/usr/local/pgsql/lib/plpgsql.so' language 'C';<ENTER>

   psql=# create language 'plpgsql' handler plpgsql_call_handler lancompiler 'PL/pgSQL';<ENTER>

3. Incorporar postgis y el conjunto de identificadores de sistemas de coordenadas del EPSG.

   psql=# psql -d murcia -f postgis.sql<ENTER>

   psql=# psql -d murcia -f spatial_Ref_sys.sql<ENTER>

Hasta la versión 5.7 de GRASS, para pasar información de GRASS a Postgis era necesario pasarla en primer lugar a formato shape y de este cargarla en la base de datos:

GRASS: /path > **v.out.shape** map=termu type=area mapset=PERMANENT<ENTER>

GRASS: /path > **shp2pgsql** termu.shp termu murcia|**psql** murcia<ENTER>

Para pasar de Postgis a GRASS era necesario llevar a cabo el procedimiento contrario:

GRASS: /path > **pgsql2shp** murcia mapa2<ENTER>

GRASS: /path > **v.in.shape** input=mapa2.shp output=mapa2<ENTER>

Los programas **shp2pgsql** y **pgsql2shp** se instalan con PostGIS.

GRASS 5.7 es capaz de leer y escribir ficheros PostGIS de forma transparente para el usuario, para ello basta con incluir un fichero head y otro frmt en el directorio del mapa vectorial en cuestión en lugar de los ficheros coor y topo, etc. El fichero frmt hace referencia a la base de datos y tabla que contiene el mapa.

PostGIS permite incorporar operadores topológicos y geométricos en SQL. Los ejemplos de las figuras 26, 27 y 28 han sido creados con las consultas que aparecen en la tabla 30.

En el primer caso se ha utilizado el operador within para identificar todos los observatorios incluidos en el polígono del municipio 31 (Murcia). En el segundo se ha utilizado el operador buffer para construir un círculo de 2 Km de diámetro alrededor de cada observatorio. Finalmente se ha utilizado este mismo poperador para construir buffers entorno a los polígonos que definen un conjunto de acuíferos.

```
SELECT o.x,o.y,o.ident
   INTO obs31
   FROM observatorios o, municipios t
   WHERE within(o.puntos,t.poligonos)
     and t.cat_id=31;

   SELECT   buffer(puntos,2000) AS buf
   INTO buff
   FROM observatorios;

   SELECT buffer(poligonos,2000) AS buf
   INTO buffacu
   FROM acuiferos;
```

Tabla 30: Consultas SQL incrporando PostGIS

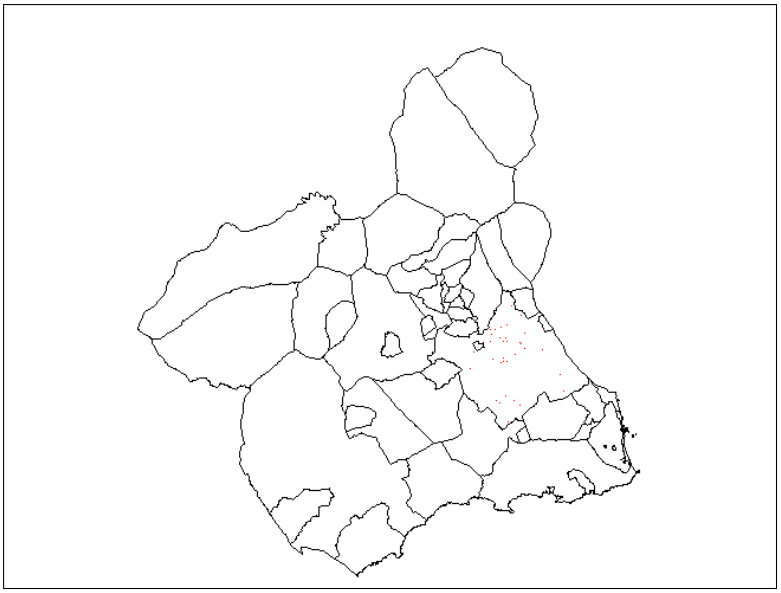


Figura 26: Observatorios meteorológicos situados en el municipio de Murcia

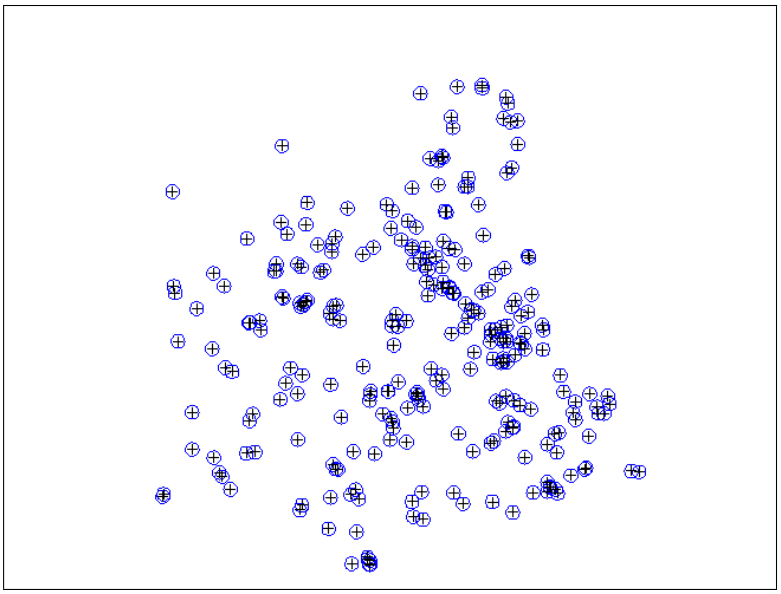


Figura 27: Buffer a los observatorios meteorológicos de la Región de Murcia

Figura 28: Buffer a los límites de los acuíferos de la Región de Murcia

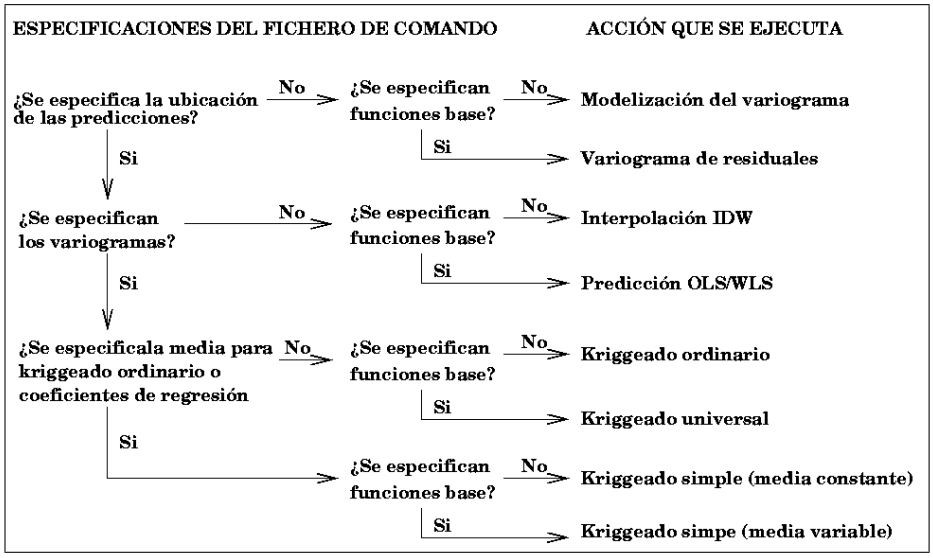


Figura 29: Árbol de decición de gstat

5.3 gstat

Gstat es un programa de análisis geoestadístico multivariante bajo licencia GPL creado en 1997 en el Departamento de Geografía de la Universidad de Utrecht[55]. Implementa diversos procedimientos geoestadísticos para modelizar la estructura de variación espacial de una variable (función semivariograma), interpolarla (kriggeado) y hacer simulación, en una, dos o tres dimensiones. Permite por tanto generar en primer lugar un modelo de variabilidad y, en función de este, interpolar unos datos o simular diversas realizaciones (utiliza el programa gnuplot para mostrar los semivariogramas). El modelo resultante es $Z = f(x, y) + v(z) + \epsilon$.

Una de las características de gstat es su capacidad para integrarse con otros programas con licencia GPL. Por ejemplo utiliza el programa gnuplot para mostrar los semivariogramas; puede leer ficheros de sites de GRASS y escribir datos raster condicionados a las especificaciones de región de trabajo de GRASS. Finalmente, puede accederse a sus funcionalidades desde R con el paquete gstat.

Para integrar gstat con GRASS se requiere

1. Compilar gstat para leer y escribir ficheros de sites o ficheros raster de GRASS condicionados a las especificaciones de región de trabajo de GRASS.

   GRASS: /path > ./configure —with-grass=path_a_GISBASE<ENTER>

   GRASS: /path > make;make install<ENTER>

   Si ya está compilado verificalo con la orden:

   GRASS: /path > gstat -v<ENTER>

   si en la respuesta aparece alguna referencia a GRASS significa que ha sido compilado correctamente

2. gstat debe ejecutarse dentro de una sesión de GRASS

3. El programa gstat debe invocarse con un fichero de instrucciones

Este fichero de instrucciones de gstat le dice al programa que acciones debe emprender y con que datos cuenta, siguiendo el árbol de decisión que aparece en la figura 29.

```
#
# Fichero de instrucciones para gstat 1
# Cálculo de semivariograma
#

data(mo): 'suelo', x=1, y=2, v=1;
data(co3ca): 'suelo', x=1, y=2, v=2;
data(gr0_2): 'suelo', x=1, y=2, v=3;

set cutoff = 20000;
set fit = 1;
set width = 2000;

variogram(co3ca): -312.801 Nug(0) + 609.775 Sph(2913.6);
```

Tabla 31: Ejemplo de fichero de instrucciones de gstat (1)

```
#
# Fichero de instrucciones para gstat 2
# Cálculo de semivariograma e interpolación
# mediante krigeado ordinario
#
data(mo): 'suelo', x=1, y=2, v=1;
data(co3ca): 'suelo', x=1, y=2, v=2;
data(gr0_2): 'suelo', x=1, y=2, v=3;
variogram(co3ca): -312.801 Nug(0) + 609.775 Sph(2913.6);
mask: 'salada';
predictions: 'salada_co3ca'x;
variances: 'co3ca_var';
```

Tabla 32: Ejemplo de fichero de instrucciones de gstat (2)

5.3.1 Ejemplos de ficheros de instrucciones de gstat

El fichero de instrucciones de la figura 31 define 3 conjuntos de datos obtenidos del mapa de sites de GRASS "suelo" especificando, en cada caso, que coordenadas corresponden a la X, la Y la variable a analizar. Se da valor a una serie de variables que determinan como se calculará el semivariograma y finalmente se da la orden de calcular este y ajustarle un modelo teórico. El resultado aparece en la figura 30

En el ejemplo se va un paso más allá y se hace el kriggeado de los datos produciendose un mapa raster de GRASS que puede verse en la figura 31

5.3.2 El paquete gstat de R

La dependencia de gstat de los ficheros de instrucciones complica la creación de scripts. Una posible solución es utilizar la librería de R para acceder a las funciones de gstat en combinación con la librería para acceder a los datos de GRASS. Entre las funciones más útiles de la librería gstat de R destacan:

library(gstat) carga la librería

data(meuse) carga los datos de ejemplo.

bubble crea un gráfico X,Y en el que el tamaño de los simbolos de los sites es proporcional al tamaño de la variable.

[55] www.gstat.org

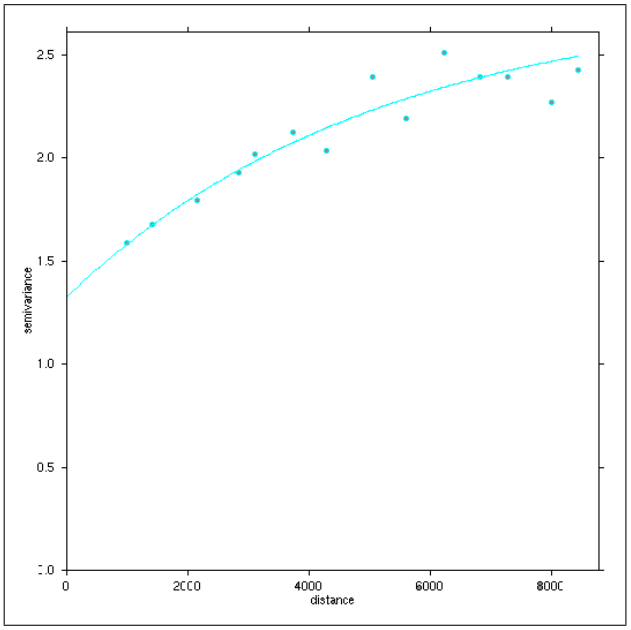


Figura 30: Resultado del primer fichero de instrucciones para gstat

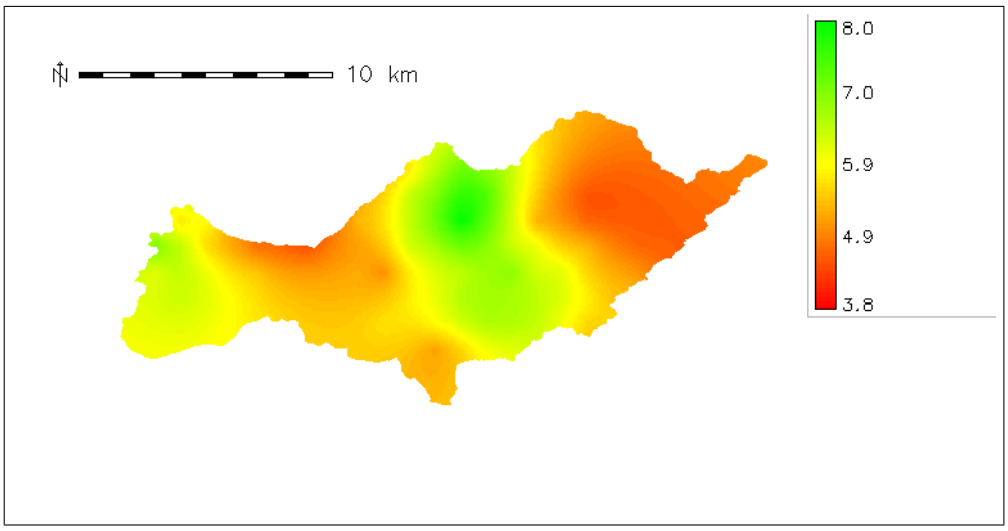


Figura 31: Resultado del segundo fichero de instrucciones para gstat

```
#
# Script de R para trabajar con las funciones de la librería gstat
# Cálculo de semivariograma y kriggeado
#
library(gstat);data(meuse);data(meuse.grid)
buble(meuse,max=2,main="Concentración de cadmio (ppm)",
     key.entries=c(.5,1,2,4,8,16),col=c(1,2))

v<-variogram(log(zinc) 1,loc= x+y,meuse)
plot(v)

fv2<-fit.variogram(v2,vgm(1,"Sph",300,1))
kv2<-krige(log(zinc)~x+y,meuse,model=fv2,meuse.grid)
image(kv2)
```

Tabla 33: Ejemplo de script de R utilizando funciones de gstat

variogram obtiene un objeto semivariograma en el que se almacena el semivariograma muestral a partir de un *data.frame* con los puntos de observación

fit.variogram ajusta un modelo teórico a un semivariograma experimental

krige hace el kriggeado

krige.cv hace validación cruzada

La tabla 33 muestra un script de R que utiliza la librería gstat y ejecuta un ejemplo que acompaña a la propia librería. Los resultados aparecen en la figura 32

5.4 Librerías, visores de mapas y servidores de mapas WEB

Ya se han mencionado las carencias de GRASS como programa de *Desktop mapping*. La aparición de una serie de librerías para el manejo de datos geoespaciales en el entorno del software abierto ha propiciado la aparición de programas de este tipo que aportan una Interfaz Gráfica de Usuario a las funciones y los datos. Estas librerías son:

GDAL/OGR . GDAL[56] es una librería con licencia abierta para la conversión de formatos raster. OGR, incluida en el sistema de directorios de GDAL, incluye similares capacidades para ficheros vectoriales. Entre los programas que utilizan estas librerías destacan GRASS, gstat, QGIS, Thuban o Mapserver

libgrass permite acceder a datos de GRASS a programas compilados fuera del entorno de programación de GRASS.

PROJ4 (Cartographic Projections Library)[57] esta librería incluye un conjunto de funciones que permiten modificar el sistema de coordenadas de la información geoespacial. Es utilizado entre otros programas por GRASS, MapServer, Postgis o Thuban.

JTS (Java Topology Suite)[58] es un API desarrollada en Java que incluye funciones geométricas y un modelo de objetos espaciales. Implementa el modelo geométrico definido por el OpenGIS Consortium en *Simple Features Specifications for SQL*. Proporciona las bases para construir aplicaciones espaciales como visores o procesadores de consultas espaciales.

GEOS (Geometry Engine - Open Source)[59] es una versión C++ de la Java Topology Suite (JTS) que trata de proporcionar toda funcionalidad de esta a los desarrolladores de C++.

[56] http://www.remotesensing.org/gdal/
[57] www.remotesensing.org/proj
[58] www.vividsolutions.com/jts/jtshome.htm
[59] geos.refractions.net/

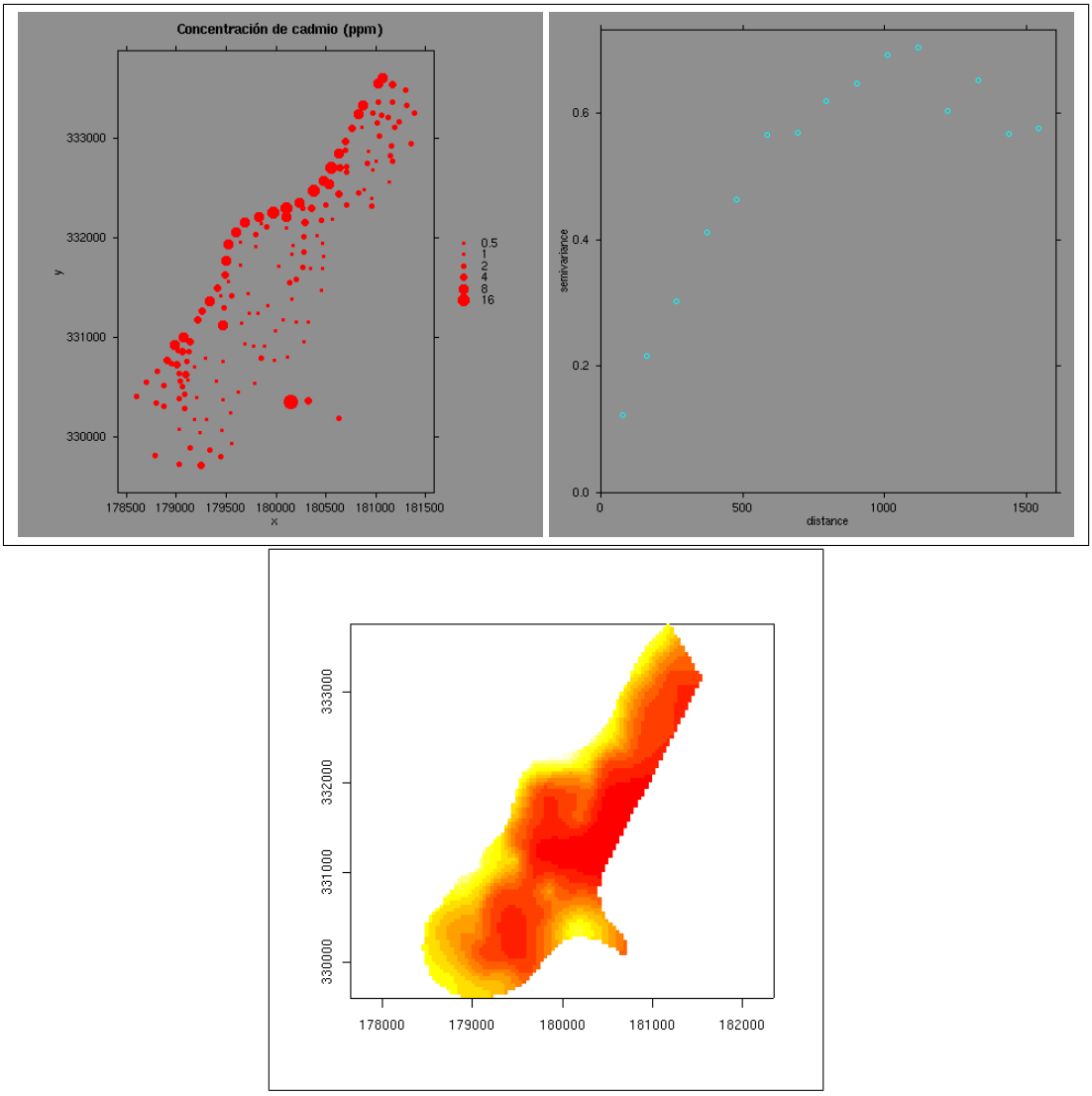


Figura 32: Resultados del script de la tabla 33)

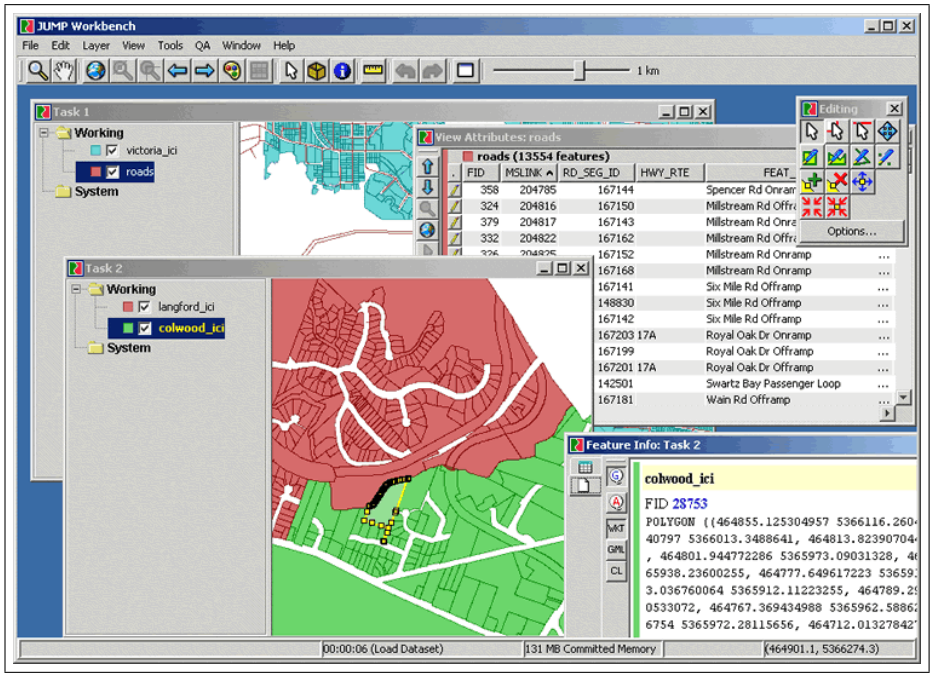


Figura 33: Pantalla de JUMP

En cua nto a los programas de *Destok Mapping* abiertos, cabe destacar:

JUMP (Unified Mapping Platform)[60] es una interfaz gráfica de usuario desarrollada en Java y orientada a la visualización y procesamiento de datos espaciales incluyendo las funciones SIG y espaciales usuales. Se ha diseñado también para ser una herramienta apropiada para desarrollar y ejecutar aplicaciones personalizadas de procesamiento de datos espaciales.

Quantum GIS (QGIS)[61] se define como un SIG construido para Linux/Unix, sin embargo es más un programa de *Desktop Mapping* que permite visualizar información raster y vectorial, así como hacer enlaces a bases de datos. Entre las potencialidades del programa se destacan:

1. Admite tablas postgis
2. Admite coberturas de ArcInfo, Mapinfo, shapefiles y otros formatos compatibles con las librerías OGR
3. Visualiza capas raster compatibles con la librería GDAL
4. Consultas para identificar y seleccionar objetos
5. Mostrar tablas de atributos
6. Exportación a Mapserver
7. Capacidad de modificar las leyendas de representación

Thuban [62] es un visor interacivo para datos geográficos. Fue desarrollado debido a la inexistencia de programas de este tipo en el entorno del software libre. Fue escrito esn Python y C++ y utiliza la librería wxWindos que le permite ser ejecutado en diversas plataformas, incluyendo GNU/Linux y windows.

Permite superponer diversas capas y manejar sus bases de datos asociadas.

Por otro lado se han desarrollado diferentes iniciativas para la creación de programas servidores de datos geoespaciales por Internet:

[60] www.vividsolutions.com/jump
[62] http://thuban.intevation.org/

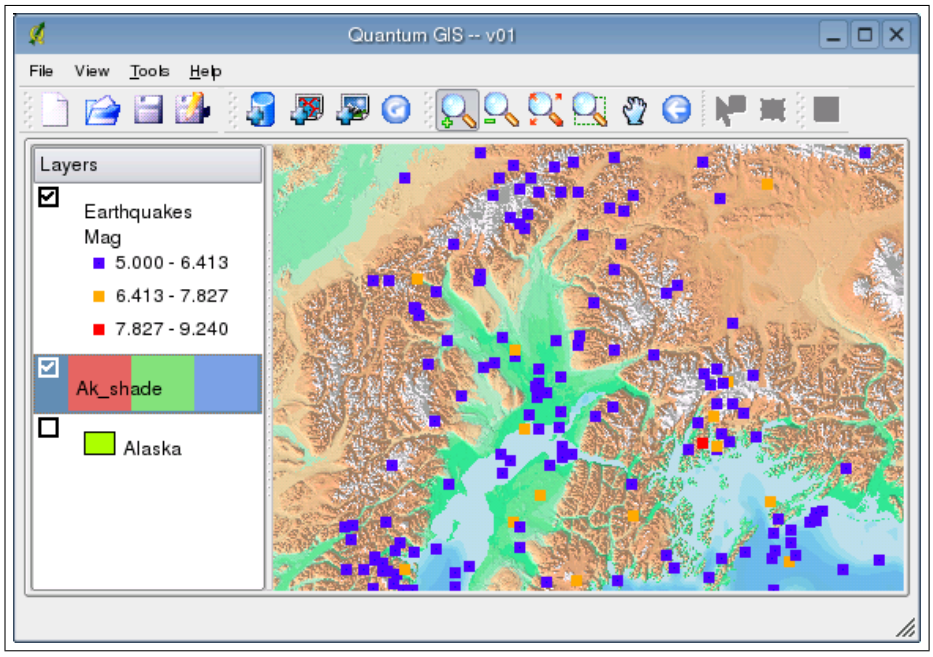


Figura 34: Pantalla de QGIS

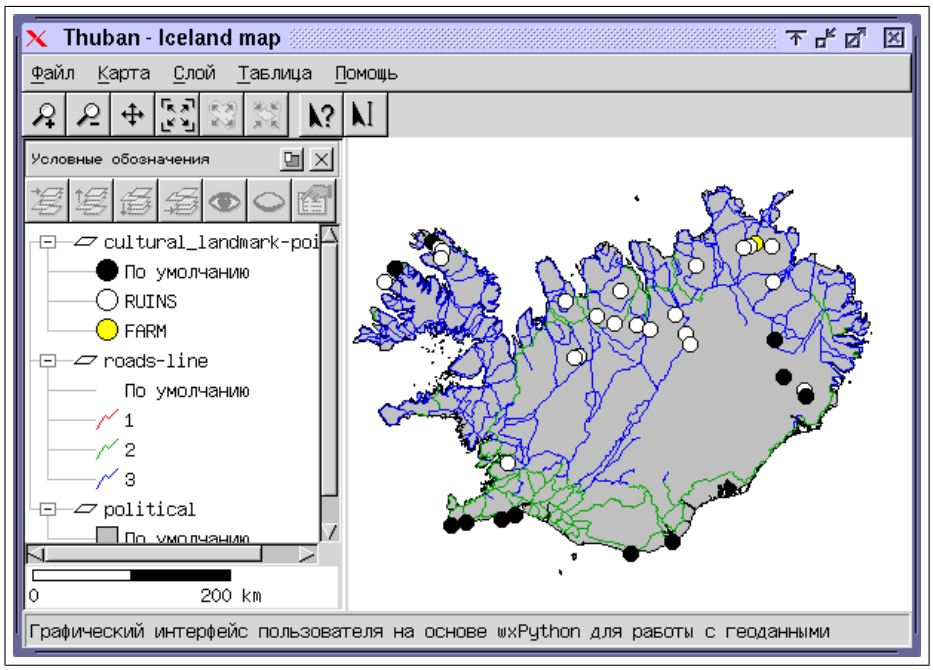


Figura 35: Pantalla de Thuban

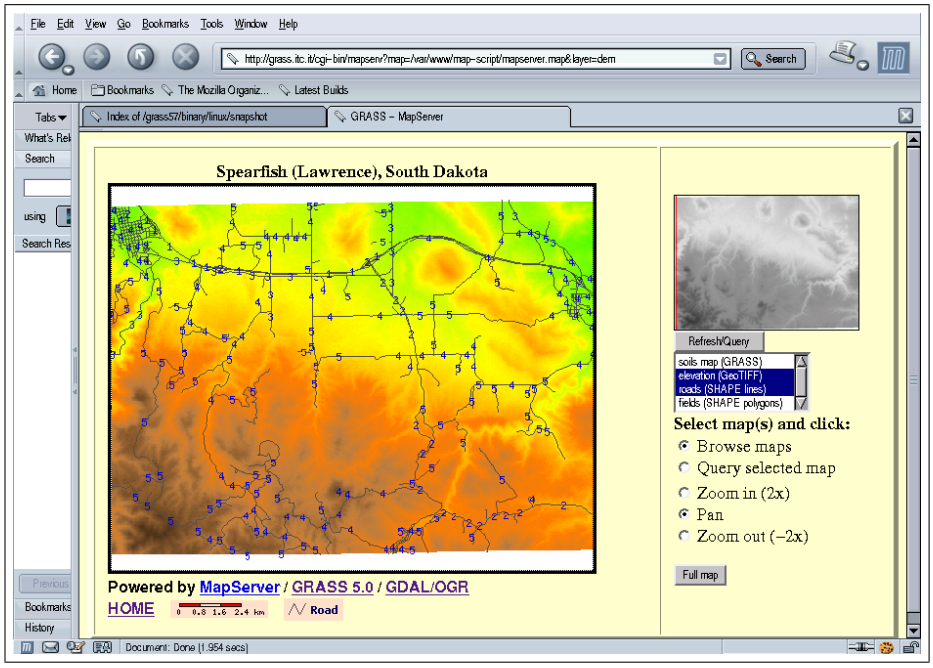


Figura 36: Pantalla de Mapserver

MapServer[63] es un entorno de desarrollo abierto para construir aplicaciones de Internet capaces de manejar datos espaciales. Emplea diversas librerías también abiertas o freeware (Shapelib, FreeType, Proj.4, GDAL/OGR). Puede compilarse en UNIX/Linux, Microsoft Windows y MacOS.

El sistema MapServer incluye MapScript que permite trabajar con diversos lenguajes de scripts (PHP, Perl, Python) e incluso con Java para acceder a la API de Mapserver escrita en C; así como una aplicación CGI ya construida que incluye gran número de funciones.

MapServer cumple con las especificaciones WMS y WFS del OpenGIS Consortium (OGC) web specifications.

MapServer no es un plenamente un SIG ni aspira a ello, simplemnente trata de proporcionar herramientas para programar diversos tipos de aplicaciones para la consulta de información geográfica vía web.

Fue desarrollado inicialmente por la Universidad de Minnesota en colaboración con la NASA y el Departamento de Recursos Naturales de Minnesota. Un creciente númer de voluntarios (en torno a 20) se encargan de mantener el código.

GeoServer[64] Se trata de una implementación en Java de las especificaciones WFS (Web Featura Server) del OpenGIS Consortium integrando un WMS (Web Map Service).

Openmap OpenMap[65] es un conjunto de herramientas de programación de código abierto basado en JavaBeans. Permite construir aplicaciones y applets para acceder y manipular datos geoespaciales. En Febrero de 2004 se publicó la versión 4.6

5.4.1 CD-lives

Una de las lineas de trabajo dentro del mundo del software libre es la creación de *CD-lives*. Se trata de CDs autoarrancables que instalan una distribución linux (de tamaño reducido por razones obvias) en un disco virtual creado en la memoria RAM del ordenador.

El objetivo es permitir un primer contacto con el sistema a nuevos usuarios sin necesidad de instalar nada en el ordenador, aunque siempre se da la posibilidad de instalar en disco duro la distribución.

[65] http://openmap.bbn.com/

Uno de los aspectos más interesantes de estos discos, es que permiten al usuario crear su propia distribución autoarrancable añadiendo o quitando software a la utilizada. Destaca en este sentido el proyecto **GIS-Knoppix**[66] cuya versión 1.0 (descargable desde al página web) incluye GRASS y Mapserver mientras que la versión 1.1 (de pago en la misma página) añade Thuban, Postgis, Jump, etc. Se basa en uno de los proyectos más conocidos dentro de los CD-live: knoppix[67].

En la Universidad de Murcia el grupo CALDUM (Conocimiento Abierto y Libre en la Universidad de Murcia) ha creado un CD-live[68] que incluye GRASS con una base de datos de Murcia. Se basa en los CD-live del proyecto metadistros[69].

6 Entornos de trabajo (INUAMA) y docencia (microaulas)

En este apartado se discuten algunos aspecto relacionados con la utilización de entornos de trabajos para proyectos SIG con software libre. Se basan en la experiencia y los resultados obtenidos a los largo de diez años de trabajo con SL.

6.1 Ventajas de un sistema multiusuario

El trabajo con SIG implica plantear un uso de los ordenadores que está más allá de lo que resulta habitual. En primer lugar se necesita el acceso a un gran volumen de información y esta información ha de ser manipulada por un equipo de trabajo, lo que supone un sistema de control y seguridad.

6.2 Curso de SIG

Del 22 de junio al 17 de julio de 1998 se celebró en la Universidad de Murcia, organizado por el INUAMA, el curso de postgrado denominado: *Técnicas de Cartografía digital, SIG y Teledetección*, que estaba destinado a 20 alumnos.

Como infraestructura se contaba con una microaula, cedida para el curso por la Facultad de Informática, y que contaba con más de 20 terminales. Además la empresa Foxen prestó para el evento dos ordenadores provistos de una CPU 200 Mhz MMX, 64MB de memoria RAM y un disco duro de 40 GB. En ellos se instaló una distribución RedHat 5.2 y una versión GRASS 4.2.

Inicialmente careciamos de experiencia que nos permitiese evaluar el número de servidores que serían precisos para el curso. Pensabamos que razonablemente el número no debía pasar de cuatro. Con el fín de evaluar la carga decidimos utilizar, inicialmente, tan sólo uno de los dos ordenadores como servidor del curso. La pruebas realizadas el primer día demostraron que el servidor era suficiente para arrancar simultaneamente mas de 20 ejecuciones de Netscape. También permitió utilizar la funcion `d.3d` (para la representación de mapas en tres dimensiones) simultaneamente por los 20 alumnos sin problema alguno.

El curso se celebró en su totalidad con la ayuda de un sólo servidor. Sólo se presentaron problemas en una ocasión, la sesión práctica de georeferenciación se proponía que cada alumno referenciase una image escaneada de 8 MB. Cuando los alumnos pusieron en marcha los 20 (o más) procesos la máquina se ralentizó en sobre manera. El problema se debía exclusivamente a la falta de memoria RAM ya que más del 90% de la cpu permanecío IDLE[70]. Tras varias horas el servidor resolvió todas las tareas y proporcionó los resultados deseados.

6.3 Microaulas de terminales

El éxito que supuso el curso anteriormente citado, en relación con las prestaciones de un equipo modesto, nos sugirió la posible utilización de GRASS como una herramienta docente, ya que el único inconveniente

[66] www.sourcepole.com/sources/software/gis-knoppix
[67] http://www.knoppix.org/
[68] caldum.um.es
[69] http://metadistros.software-libre.org/es/
[70] Es decir disponible para realizar otras tareas

era la disponibilidad de un servidor que al menos ofreciese las prestaciones mencionadas en una microaula de menos de 20 puestos. Se puso en marcha un proyecto piloto para dar servicio a la microalua de la Facultad de Biología (con 15 equipos 386) en una red de win3.11 y crear una segunda microaula con 12 equipos "reclicados" 486 que dispondrian de disco duro (para instalar win3.11) y arranque remoto para linux.

Se eligió como servidor un ordenador Pentium II 350 montado sobre una placa base dual (inicialmente con un sólo proecesador) y con 256 MB de memoria RAM y un total de 60 GB en discos IDE.

Posteriormente, al comprobar el rendimiento adecuado se mejorarón las prestaciones del sistema incorporando la senunda cpu y aumentando la memoria RAM a 640 MB e incorporando un disco SCSI de 10 GB.

Las microaulas se actualizaron posteriomente pasando a disponer todas las máquinas de una CPU, de al menos de 350 MHz. También se aumento su dotación y ambas disponen en la actualidad de 20 puestos de trabajo. El servidor permanece funcionando ininterrumpidadmente por periodos superiores a los 6 meses (tenemos problemas de suministro eléctrico).

En la actualidad el servidor proporciona servicios de impresión, de terminales gráficos a la delegación de alumnos de la Facultad de Biología, presta sus discos por samba a las dos microaulas, ofrece servicio de terminal gráfica a estas y además a otra microaula (ADLA[71]), situada en la facultad, y gestionada por ATICA[72].

6.4 El laboratorio de SIG y teledetección del Instituto Universitario del Agua y del Medio Ambiente (INUAMA)

Dados los resultados satisfactorios, del uso combinado de servidores “potentes” y terminales sin disco duro, se dotó al laboratorio de SIG y teledetcción de una infrasestructura basada en estas ideas.

El ahorro que proporcionó la eliminación de discos duros fué equivalente al coste del servidor. Así, utilizando un equipo “obsoleto”, el mismo que se destinó como servidor al curso mencionado anteriormente, como servidor de arranques remotos y el nuevo servidor para realizar exclusivamente las tareas relacionadas con la investigación (elaboración de informes y análisis SIG).

La estrategia para optimizar los recursos y la comunicación se basón en un directorio (`region`) en el que los usuarios incorporaban sus directorios de trabajo de GRASS (`mapset`). Esto proporcionaba el acceso inmediato a las últimas modificaciones y mejoras.

Puede afirmarse que el número de problemas, de mantenimiento de los servidores y administración del equipo, fueron muy reducidos. Hay que destacar una amarga lección: durante un cierto periodo de tiempo el servidor se caia con una cierta frecuencia. Tras revisar minuciosamente el sistema y su configuración, finalmente, la avería de la fuente de alimentación se manifestó claramente. La moraleja: si un equipo con SL falla erráticamente desconfia del hardware.

6.5 Otras estrategias

Con el abaratamiento de los equipos informáticos, ordenadores y comunicaciones, se puede plantear el laboratorio como un sistema base (en su caso puede disponer de un sistema de autentificación LDAP) que sirve por NFS los directorios con las bases de datos que se van a utilizar desde las terminales. Esta terminales pueden ser de tres tipos:

- Terminales gráficas que utilizan la CPU y el disco del servidor

  Ventajas: Pueden utilizarse “viejos ordenadores” como terminales y basta una inversión relativamente pequeña en el servidor (unos 500 euros). No es necesaria una gran experiencia para montar y usar una red.

  Inconvenientes: Todas las terminales utilizan los recursos de un servidor pueden producirse picos, con tareas muy exigentes en CPU o acceso al disco, con la ralentización de todos los usuarios. No resulta conveniente usas administradores de ventanas muy exigentes.

[71] Aula Docente y de Libre Acceso
[72] Área de las Técnologías de la Información y las Comunicaciones

- Sin disco duro y con arranque remoto desde el servidor.

  Ventajas: Las terminales carecen de mantenimiento.

  Inconvenientes: Necesita un experto que pueda montar los arranques remotos.

- Con disco duro y sistema operativo accediendo a la base de datos del servidor (mediante protocolos NFS o SMB).

  Ventajas: Descarga el tráfico en la red y se evita el uso intensivo de la CPU del servidor.

  Inconvenientes: Necesita un mayor grado de conocimientos por parte del administrador.

7 Bibliografía

- Bivand,R. & Neteler,M. (2000) Open Source geocomputation: using the R data analysis language integrated with GRASS GIS and PostgreSQL data base systems in *Geocomputation 2000* http://reclus.nhh.no/gc00/gc009.htm
- Cárdenas Luque,L. (2001) *Curso básico de SQL* (http://rinconprog.metropoliglobal.com/CursosProg/BDatos/SQL_Basico/index.php?cap=1)
- Close,D.B.; Robbins,A.D.; Rubin,P.H.; Stallman,R. y van Oostrum,P. (1995) *The AWK Manual* (http://www.cs.uu.nl/docs/vakken/st/nawk/nawk_toc.html)
- Free Software Foundation (2002) *Bash Reference Manual* (http://www.gnu.org/software/bash/manual/bashref.html)
- Goran, W.D., Dvorak, W.E., Van Warren, L. and Webster, R.D. (1983) Fort Hood Geographic Information System: Pilot System Development and User Instructions, Technical Report N-154, USA Construction Engineering Research Laboratory, Champaign, IL. http://grass.itc.it/fhgis83rep.html
- Hastings,D.A. *The Geographic Information Systems: GRASS HOWTO* http://tldp.org/HOWTO/GIS-GRASS/
- Kernighan,B.W. y Pike,R. (1987) *El entorno de programación Unix* Prentice Hall, 384 pag.
- Kernighan,B.W. y Ritchie,D.M. (1991) *El lenguaje de programación C* Prentice Hall, 294 pag.
- Mike G. (2000) *Programación en BASH - COMO de introducción* (http://es.tldp.org/COMO-INSFLUG/COMOs/Bash-Prog-Intro-COMO/)
- Neteler, M. (2003) *GRASS 5.0.x and PostgreSQL - First Steps* (http://freegis.org/cgi-bin/viewcvs.cgi/ checkout /grass/src.garden/grass.postgresql/tutorial/index.html)
- Neteler, M. & Mitasova, H., (2002) *Open Source GIS. A GRASS GIS Approach* Kluwer, Boston/Dodrecht/London, 434 pags.
- PGDG, PostgreSQL Global Development Group (2002) *PostgreSQL User Tutorial* (http://www.postgresql.org/docs/pdf/7.3/tutorial-7.3.2-A4.pdf)
- Ramsey,P. *PostGIS Manual* (http://postgis.refractions.net/docs/). Traducido al español por A. Martín Martín (www.postgis.refractions.net/postgis-spanish.pdf)
- Universidad de Oviedo *Tutorial de Tcl/Tk* (http://www.etsimo.uniovi.es/tcl/tutorial/)
- Wall,K. (2001) *Programación en Linux* Prentice Hall, 846 pags.
- Worsley,J.C.; Drake,J.D. (2002) *Practical PostgreSQL* O'Reilly, 619 pags.