**UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO**

**FACULTAD DE CIENCIAS**

# MANUAL INTRODUCTORIO A LOS SISTEMAS DE INFORMACIÓN GEOGRÁFICA (SIG) DE CÓDIGO ABIERTO CON APLICACIÓN A DINÁMICAS BIOLÓGICAS ESPACIALES

**T E S I S**

**QUE PARA OBTENER EL TÍTULO DE:**

**B I Ó L O G O**

**P R E S E N T A:**

**JUAN MANUEL ESCAMILLA MÓLGORA**


**DIRECTOR DE TESIS:**
**DR. PABLO PADILLA LONGORIA**

Universidad Nacional Autónoma de México
UNAM
Biblioteca Central
Dirección General de Bibliotecas de la UNAM

**UNAM – Dirección General de Bibliotecas**
**Tesis Digitales**
**Restricciones de uso**

**DERECHOS RESERVADOS ©**
**PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL**

Todo el material contenido en esta tesis esta protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

**Hoja de datos del jurado**

**1.-Datos del alumno.**
Escamilla
Mólgora
Juan Manuel
(044) 55 28 55 89  06
Universidad Nacional Autónoma de México
Facultad de Ciencias
Biología
300625026

**2.-Datos del tutor**.
Dr
Padilla
Longoria
Pablo

**3.-Datos del sinodal 1**
Dra
Almeida
Leñero
Lucía Oralia

**4.-Datos del sinodal 2**
Dr
Becerra
Bracho
Arturo
Carlos II

**5.-Datos del sinodal 3**
Dra
Bonfil
Sanders
María del Consuelo

**6.-Datos del sinodal 4**
Dr.
Olson
Zunica
Mark Earl

**7.-Datos del trabajo escrito**
Manual introductoria a los sistemas de información geográfica (SIG) de código abierto con aplicación a dinámicas biológicas espaciales
143p
2010

# Manual Introductorio a los Sistemas de Información Geográfica (SIG) de Código Abierto.

## *Con aplicación a dinámicas biológicas espaciales*

Juan Manuel Escamilla Mólgora

Tesis de Licenciatura

Facultad de Ciencias, UNAM.

12 de septiembre de 2010

# Índice general

# Prefacio

*Todo lo que sucede, sucede en algún lugar*. Esta frase es la primera que aparece en el libro: *Geographic Information Systems and Science* (Longley et al., 2005) uno de los nuevos clásicos de esta aun joven disciplina. A pesar de que esta ciencia, la geomática, es relativamente nueva (cerca de 30 años) (López, 2004), actualmente ha tenido un crecimiento y diversificación muy grande debido al avance de las nuevas tecnologías como los dispositivos GPS, internet inalámbrico, banda ancha, redes y bases de datos semánticas. Ahora, prácticamente todo es suceptible de referenciarse geográficamente. Estas nuevas herramientas llegan en el momento en que necesitamos resolver serios problemas que involucran el desarrollo y bienestar de las futuras generaciones. El cambio climático, el manejo óptimo de recursos naturales y el ordenamiento territorial son sólo algunos de los problemas a los que las nuevas generaciones de científicos se están avocando; problemas que requieren necesariamente de la componente espacial para ser descritos adecuadamente.

Los Sistemas de Información Geográfica (SIG) permiten integrar congruentemente esta información espacial con cualquier otro tipo de información digital, generalmente bases de datos. Esta integración hace posible la participación de expertos de distintas ciencias, siendo entonces una área idónea para la *inter* y *transdisciplina*. De esta forma se pueden forjar nuevos conocimientos, preguntas y soluciones que con los métodos disciplinares *canónicos* no se hubieran podido siquiera soñar. En particular se facilita la modelación y simulación de fenómenos complejos que ocurren en la Tierra; permitiéndonos tomar decisiones óptimas en el manejo de recursos naturales, desastres, control de epidemias, ordenamiento territorial, planeación de vías de comunicación y uso de suelo, entre muchos otros problemas.

## 0.1. Objetivos del manual

Existe muchísima literatura entorno al uso de SIG para prácticamente cualquier aplicación; desde *geosimulación* con autómatas celulares (Benenson y Torrens, 2006) hasta *geomarketing* (Cliquet, 2006) pero no hay mucha información disponible en español, menos con una orientación ambiental y mucho menos utilizando *Software Libre*. Es importante trabajar en este tema porque a pesar de ser una herramienta poderosa para los biólogos, en el plan de estudios de ésta carrera, en la Facultad de Ciencias de la UNAM, no se contempla su uso. Este manual pretende: *i)* ser una referencia rápida, básica en lenguaje amigable y fácil de entender para estudiantes de ciencias ambientales en nivel licenciatura. *ii)* mostrar que no es necesario gastar miles de pesos

para realizar dichos análisis, todo se puede hacer con *Software Libre*. Es muy importante que los biólogos conozcan las alternativas libres y sepan que hay personas trabajando para que podamos usar, aprender, modificar y distribuir estas herramientas. El comienzo, como todo, cuesta trabajo, pero vale la pena pues tendremos mucha más flexibilidad para trabajar. Sin reparos afirmo: *El Software Libre es vanguardia*.

## 0.2. Estructura

**Capitulo 1:** Se da una introducción al *Software Libre*, su filosofía, historía y algunos conceptos básicos. Se describe cómo instalar un sistema operativo *Libre* de la familia *Linux*, a saber, **Ubuntu**. Se brinda una pequeña introducción al uso de la consola y comandos, instalación de programas y un esquema general de la programación orientada a objetos. [1]

**Capítulo 2 :** Se presentan los conceptos básicos de los SIG, con especial énfasis en los conceptos cartográficos como: coordenadas, proyecciones, *datums*, geoide, elipsoide. Se explican los modelos *ráster*, vectorial y volumétrico. Se hace énfasis en los parámetros, proyecciones y *datums* más usados en México.

**Capítulo 3 :** Se describen las herramientas *libres* que se van a utilizar para los SIG, con énfasis en QGIS. Se hacen ejemplos vectoriales utilizando esta aplicación. Mostrando como bajar mapas, editarlos, buscar valores en la tabla de atributos y georreferenciar, entre otras cosas. Se incluye también una guía de instalación de las bibliotecas GDAL/OGR y su utilización para transformar coordenadas y reproyectar mapas. También se hace una descripción general del SIG GRASS, su instalación y su estructura de datos.

**Capítulo 4 :** Se ejemplifican tareas de geoprocesamiento básico utilizando como plataforma GRASS. Entre las actividades realizadas están: creación de localidades, importar mapas vectoriales y ráster, importar datos en texto simple, extraer subconjuntos de datos de un mapa en particular, conversión de puntos y líneas en áreas, manipulación de la tabla de atributos y base de datos, transformación de mapas vectoriales a ráster y para modelos ráster: fusión de mapas, creación de mapas de relieve topográfico, manipulación de capas de color para imágenes *Landsat* y obtención de mapas de pendientes y curvas de nivel.

**Apéndices :** Consta de tres secciones. La primera describe los 80 comandos más utilizados de la consola en sistemas *Linux* con su correspondiente sintáxis. La segunda consta de un glosario de términos geográficos y computacionales utilizados en el texto. El último apéndice es el código fuente de un programa escrito en *Python* que convierte las coordenadas,

[1] Los paquetes de procesamiento geográfico (i.e. SIG) que se van a utilizar originalmente fueron diseñados para sistemas *UNIX*. Actualmente hay versiones para computadoras con *Windows* y aunque en teoría estas aplicaciones debieran funcionar igual puede haber diferencias en los usos. Si se va a optar por utilizar *Windows* se puede saltar este capítulo.

en grados sexagesimales, de las localidades de los datos del Censo de Población y Vivienda del año 2000 publicado por el (INEGI) y Instituto Nacional de Estadística y Geografía (2009) en grados decimales. Se incluye para ejemplificar la programación en dicho lenguaje y el modo adecuado de documentación e incorporación de la licencia GPL.

## 0.3. Agradecimientos

A mis padres, Sergio Escamilla y Teresa Mólgora por apoyarme en absolutamente todo, especialmente en los momentos más difíciles. A mi tía María Mólgora por su cariño, apoyo y confianza. A Mónica Minjares por todo el tiempo que hemos estado caminando juntos. A Pablo Padilla por confiar en mí. A Rocío Alanís por prestarme algunas imágenes satelitales de su proyecto. A mis amigos: Everardo Robredo, Juan Carlos González, Guillermo Espartaco Orozco, Julia Moreno, Erika Hagman, Bruno Barrales y Emmanuel Gómez por el tiempo compartido, en especial en esos viajes trascendentales que nos cambiaron para bien. Al proyecto de DGAPA (**PAPPIT IN229109-2**) por la beca otorgada. A la comunidad del *Software Libre* por desarrollar y mantener herramientas tan maravillosas. Al pueblo de México.

## 0.4. Software utilizado

El texto fue escrito en el lenguaje LaTeX. Las imagenes fueron creadas con *Inkscape y Gimp*. Los mapas fueron elaborados y preparados en QGIS y GRASS. Para algunas cosillas más se utilizó *Python*. **Este manual es 100 % *SOFTWARE LIBRE*.**

Juan Escamilla Mólgora
Ciudad Universitaria, julio, 2010.

# Capítulo 1

# GNU/Linux para biólogos

*La libertad de los otros prolonga la mia hasta el infinito.*
–Mijaíl Bakunin. [1]

El objetivo de este capítulo es motivar en el lector la migración a sistemas operativos libres ya que por sus características superan por mucho a los paquetes comerciales de código privado. Para lograr tal fin, revisaremos brevemente la historia del movimiento del *Software Libre* y su filosofía. Daré una lectura personal de éste en la actualidad y veremos también las implementaciones de esta filosofía en el desarrollo de GNU/Linux. La segunda sección del capítulo comprende aspectos prácticos, que abarcan la instalación de este sistema operativo, comandos, aplicaciones básicas, instalación de programas y algunos trucos en la red.

## 1.1. Historia y filosofía del software libre

### 1.1.1. Breve historia del Software Libre

En los comienzos del uso generalizado de las computadoras los fabricantes de éstas otorgaban el sistema operativo como un añadido al producto para que los clientes pudieran usarlas. Los programadores y usuarios de éstas cotidianamente intercambiaban sus programas entre sí y con esto podían avanzar más rápidamente en sus proyectos.

Fue hasta finales de los años 70 que las compañías comenzaron el *hábito* de imponer restricciones a los usuarios y desarrolladores con el uso de Acuerdos de Licencia.

La historia del *Software Libre* comenzó cuando el laboratorio de inteligencia artificial *AILab* del *Massachusetts Institute of Technology* (MIT) tuvo problemas con su impresora de red. Ésta tenía un error recurrente de atasco de papel cuando llegaban peticiones de impresión simultaneamente. Desde hacia tiempo las corporaciones tecnológicas "obsequiaban" prototipos a los

[1] Consigna de protesta en las paredes del Liceo Condorcet. París, mayo de 1968.

laboratorios de computo de universidades renombradas como el *MIT y Harvard*, con la intención implícita de reducir los costos de la corrección de errores en el software (*bugs*) ya que los mismos usuarios de estos prototipos sabian de programación y compartian las mejoras y correcciones con la compañía sin costo alguno. Las compañías a su vez, publicaban el código del programa en un lenguaje accesible para el desarrollador. A este tipo de código se le llama *Código Fuente* . Sin embargo, este no fue el caso de la nueva impresora laser desarrollada por *Xerox* y ubicada en algún rincón del *AILab*. Richard Stallman , entonces un estudiante de física en *Harvard* y *hacker* en el AILab decidió darse a la tarea de corregir el problema, similar al que había resuelto antes para la predecesora de la nueva *Xerox*. Lo que finalmente encontró fue un código ininteligible, el código que sólo la computadora puede ejecutar; puros unos y ceros. Cuando Stallman acudió a *Xerox* para la obtención del código fuente este le fue negado. A pesar de esto, con ayuda de otro laboratorio de cómputo en Harvard pudo resolver el problema. A medida que pasaba el tiempo era más notorio que las compañias empezaban a cobrar por el software que antes se distribuía libre y gratuitamente. Se inventaron los *Acuerdos de Licencia* y los desarrolladores, algunos compañeros de Stallman, estaban obligados a no compartir su software incluso si utilizaba código de terceras personas. Era apropiarse del trabajo de muchas personas y ponerle un solo apellido y un solo dueño. Así, la gente del AILab comenzó a vender su código, lo que contradecía los valores éticos de Stallman, quien se fue apartando de sus compañeros convirtíendose, en poco tiempo, en un ermitaño.

Era la epoca en que grandes compañías como *Microsoft* empezaban a crecer y aún no se asociaba el software con el logotipo corporativo. Con esta nueva tendencia el 27 de septiembre de 1983 Stallman se decidió a publicar en UseNet [2] el inicio del proyecto GNU [3], que perseguía la creación de un sistema operativo completamente libre, apegado a las cuatro leyes básicas de ética *hacker* Williams (2002) propuestas en el mismo documento. Así, todo software que cumpliera con estas cuatro libertades se le reconocería como *Software Libre* en inglés *Free Software* . Estas libertades son:

0. La **libertad** de poder ejecutar el programa para cualquier propósito.

1. La **libertad** de aprender y analizar el funcionamiento del programa, adaptándolo a las necesidades de los usuarios.

2. La **libertad** de poder distribuir las copias originales o modificadas.

3. La **libertad** de mejorar el programa y llevar las mejoras al público, beneficiando a toda la comunidad.

Stallman también fue creador de la Fundación del Software Libre , FSF por su acrónimo en inglés, la cual funge como protectora del Software Libre garantizando las cuatro libertades con el uso de la licencia GPL (General Public License) Foundation. Esta licencia es una de las más usadas hoy en día por los desarrolladores del Software Libre.

[2] Usenet es uno de los sistemas más antiguos de comunicaciones entre redes de computadoras, aún en uso.

[3] El nombre GNU proviene del acrónimo *GNU's not Unix;* siguiendo una tradición *hacker* de acrónimos recursivos.

Lo que sigue es un fragmento del comunicado publicado el 27 de septiembre de 1983 Stallman.

*Inicialmente, GNU constará de un núcleo del sistema [*kernel*] más todas las utilerías necesarias para escribir y ejecutar programas en lenguaje C: editor, interface de comandos [shell], compilador C, enlazador, ensamblador, y algunas otras cosas. Después se añadirá un formateador de texto, un juego tipo imperio, una hoja de cálculo, y cientos de otras cosas. Esperamos proporcionar, en el futuro, todas aquellas cosas útiles que normalmente vienen con un sistema Unix, y cualquier otra cosa que sea útil, incluyendo documentación en-línea e impresa.*

**El nacimiento de Linux**

Un sistema operativo se puede entender como una máquina con dos componentes esenciales. Uno llamado núcleo, *kernel*, que es el software encargado de operar las partes físicas de la computadora (*hardware*) i.e. administrar la memoria, la escritura y lectura en el disco duro, conexiones de red, usb, monitor, etcétera. Otra parte se encarga de las aplicaciones, es decir programas; compiladores, interfaz gráfica, reproductores de música, SIG [4], juegos, etcétera.

Cuando Stallman publicó su propuesta para crear un sistema operativo acorde a las libertades de la FSF Stallman (1983) tenía la mayoría de las aplicaciones listas pero le faltaba lo más importante: el núcleo. Esta aportación fue realizada en 1991 por Linus Torvalds, un computólogo finlandés que acababa de liberar bajo la licencia GPL un núcleo del sistema operativo llamado *Minix* que llevaba trabajando algunos años. Torvalds El origen del nombre *Linux* viene de la conjunción de Linus y Unix.

Fue hasta ese momento que nació el primer sistema operativo libre: el sistema operativo GNU/Linux.

### 1.1.2. Lectura actual del movimiento del Software Libre

Un elemento imprescindible para el desarrollo del Software Libre es la comunicación de sus integrantes. Si este movimiento ha crecido tanto en estos años se debe en gran medida a la socialización de la información por el surgimiento de un nuevo medio de comunicación llamado *Internet* . La presencia de este medio se ha hecho, recientemente, más conspicua en todo el mundo. Prácticamente en cualquier sitio urbano se pueden encontrar puntos de acceso inalámbricos, *wifi spots*. Las redes moviles están llegando a una población considerable que puede costear el servicio. En muchas universidades, aeropuertos, sitios y plazas públicas se encuentran disponibles redes de libre acceso. Por otro lado, la creación de proyectos ha trascendido a los desarrolladores mismos. Ahora la comunidad no consta únicamente de programadores y expertos en sistemas. Cualquier persona interesada puede participar ya sea reportando errores en los programas, *bugs*, traduciendo aplicaciones, publicando comentarios, soluciones y configuraciones en foros y creando arte gráfico como, íconos, pantallas de inicio, menús, temas, protectores de pantalla, etcétera. En fin, cada día la comunidad crece más y se

[4] Sistemas de Información Geográfica

integra a su vez con proyectos en principio independientes. e.g la integración del explorador de archivos *Konqueror* con la Wikipedia. En resumen, el Software Libre se ha acoplado a un movimiento todavía más grande de generación de conocimieto colectivo con *copyright* universal o mejor dicho, *copyleft* [5] Foundation universal.

Sin embargo todavía hay muchos problemas por resolver. Esta intricada pero frágil red todavía esta a merced de las grandes compañias y monopolios. México, desgraciadamente, ha sido una tierra fértil para éstos. El informe anual del Programa de las Naciones Unidas para el Desarrollo, *PNUD* indica que en la actualidad (abril, 2009) en el país se paga un acceso a *Internet* caro y lento comparado con la mayor parte del continente, debido, sobre todo, al monopolio de las telecomunicaciones en el país i.e. Telmex. Prácticamente toda la información electrónica que sale de México pasa por alguno de sus servidores. Esto compromete la libertad de los usuarios a compartir información que atente en contra de los intereses de Telmex y de sus socios comerciales, uno de ellos Microsoft, conocido enemigo del Software Libre (S.L.), en particular de la licencia GPL. El crecimiento acelerado de la comunidad del S.L. ha hecho que estas grandes compañías busquen formas para aniquilar el movimiento. Entre las que destacan: propaganda de desacreditación, restricción de controladores del *hardware* e incluso prohibición de acceso a ciertos sitios web por utilizar un sistema operativo abierto. Sólo con la participación de la mayoría de los usuarios podremos detener a esas compañías depredadoras y contribuir así a una verdadera socialización de la información y la tecnología. Ésta es una de tantas razones para seguir utilizando Software Libre e incentivar a otros a utilizarlo.

**Otras razones para utilizar *Software Libre***

Los principios del Software Libre garantizan la posibilidad de auditar el programa en busca de vulnerabilidades y fallos de programación. Esto le brinda al usuario final seguridad en su información y rendimiento en sus operaciones. Además, es tanta la cantidad de usuarios y desarrolladores trabajando en esta comunidad que prácticamente cualquier paquete comercial tiene una versión libre.[6] Esto significa que podemos olvidarnos de comprar costosos programas o en su defecto utilizar herramientas piratas que sólo comprometen nuestro sistema y nuestra información. Utilizando Software Libre como plataforma de trabajo ayudamos a contraer la llamada *brecha digital* entre personas con escasos recursos que tienen dificil acceso a nuevas computadoras. El Software Libre con su constante actualización provee de algorítmos y programas eficientes que aumentan significativamente la vida útil de las máquinas, permitiendo el reuso de computadoras viejas. Lo más importante es que cualquier persona en el mundo tiene la capacidad de darle vida a un proyecto creado por y para la colectividad. Esta forma de trabajo autoorganizada surge y se traduce como una vanguardia en contra de las imposiciones de gobiernos, monopolios y corporaciones en todo el mundo.

[5] Copyleft significa que cualquiera que redistribuya el software, con o sin cambios, no podrá restringir a nadie la libertad de copiarlo, redistribuirlo o cambiarlo. Copyleft garantiza que el usuario mantenga su libertad.

[6] Sitio de Alternativas Libres `www.freealts.com`

El desarrollo en paralelo del software también reduce sígnificativamente los costos y tiempos de producción. Sólo para dar una idea: un estudio sobre la distribución Red Hat 7.1 en 2001 , reveló que ésta posee más de 30 millones de líneas de código real. Utilizando el modelo de cálculo de costos COCOMO (Boehm et al., 1995) , puede estimarse que esta distribución requeriría 8,000 programadores por año para su desarrollo. De haber sido desarrollado por medios convencionales de código cerrado (propietario), hubiera costado más de mil millones de dólares en los Estados Unidos (Wheeler, 2001).

En un estudio posterior (González-Barahona et al., 2001) se realizó el mismo análisis para Debian GNU/Linux versión 2.2.(actualizado al año 2000). Esta distribución contenía más de 55 millones de líneas de código fuente. Habría costado 1,900 millones de dólares desarrollarla por medios convencionales (no libres). En comparación el núcleo de Linux en octubre de 2003 tenía unas 5.5 millones de líneas.

Unix fue pensado para ser un sistema operativo multiusuarios, característica heredada a Linux. Ésta le da al sistema operativo mayor capacidad de gestión de cuentas y permisos. En Linux tenemos total libertad de dar a los usuarios del sistema permisos de ejecución, lectura y escritura de cualquier archivo. Incluso se pueden crear grupos de usuarios con determinados permisos adecuados para un proyecto en particular. Esta es una de las razones por las cuales los virus, troyanos, gusanos, en general todo programa malicioso tiene muy pocas posibilidades de éxito, muchísimas menos comparado con sistemas Windows. Si a esto le sumamos la constante actualización, Linux se convierte en uno de los sistemas operativos más seguros del mundo.

## 1.2. GNU/Linux

### 1.2.1. Breve descripción

GNU/Linux, o como se dice comúnmente Linux, es una familia de sistemas operativos genéricos originalmente desarrollados para ser similares al sistema Unix. El desarrollo de GNU/Linux [7] se debe fundamentalmente a su filosofía de Software Libre y al mejoramiento continuo por parte de la comunidad internacional de desarrolladores. Con el paso del tiempo Linux ha evolucionado en función de las necesidades de sus usuarios, para convertirse en un sistema operativo estable, completo y eficiente; superando por mucho las expectativas originales. Más aún, de los grupos de usuarios surgen *sabores* diferentes que también evolucionan y se diversifican. Las variantes de estos sistemas llamadas *distribuciones* tienen el objetivo de ofrecer un sistema operativo completo y acorde a las necesidades de determinado grupo de personas con oficios afines. Tal es el caso de *Ubuntu Server*, especialmente diseñado para uso en servidores; *Geento*, especializado en rendimiento y configurable al límite; *Debian*, pensado como un sistema operativo totalmente libre y recomendado por la FSF, entre muchos otros. Así, existen tantas distribuciones de Linux como usos variados se le pueden dar a la máquina. Desde computadoras para niños en paises en vias de

[7] En adelante Linux

desarrollo [8] hasta supercomputadoras. [9]. Para darse una idea de la cantidad de distribuciones existentes basta visitar el sitio : `http://distrowatch.com` dedicado exclusivamente a recomendar y analizar prácticamente todas las distribuciones existentes En ocasiones es común encontrar el término *distro* como abreviatura de distribución.

Hay muchas formas diferentes de explorar el mundo Linux. Si no se le conoce o no se ha estado en contacto con él, se puede utilizar una versión *en vivo ( Live-CD )*. Estas distribuciones pueden ser grabadas en un disco compacto, en una memoria USB o en una tarjeta de memoria (Memory Stick, SD/MMC, etc.). La mayoría de las distribuciones nuevas tienen esta opción en el arranque. En Ubuntu esta opción se encuentra al insertar el disco de instalación y reiniciar la computadora en el menú que aparece después de seleccionar el idioma. Como se puede ver en la figura 1.1 lleva el nombre de *Probar Ubuntu sin alterar el equipo*. Con esta opción, el sistema se carga en la memoria RAM, que es la memoria temporal, y no se manipula el disco duro, dejando todos los archivos y el sistema operativo original intactos. Las versiones *en vivo* permiten experimentar, destruir, jugar y en la mayoría de los casos, convencer a efectuar la instalación definitiva en el disco duro. La mayoría de las *distros* tienen una versión *Live-CD*.

**Escoger distribución**

Debido a la gran *diversidad* de distribuciones disponibles. La respuesta a la pregunta ¿qué distribución escojer ? depende de la pregunta más importante de todas :

**¿ Para qué voy a utilizar la computadora?**

Para los fines de este manual supondremos que el lector está interesado en la ecología del paisaje, análisis espacial, Sistemas de Información Geográficos (SIG), análisis estadísticos y demás utilidades para el quehacer del científico, en particular del biólogo. Además de que se utiliza una computadora estándar (PC o Mac) y que no se quiere profundizar en instalaciones a la medida de *hardware*.

Dado este perfil de usuario, es hora de anunciar que este manual se basará en la distribución más popular de todas; **Ubuntu** bajo el entorno gráfico KDE [10]. Utilizamos este *sabor* por las frecuentes actualizaciones que tiene, su facilidad para instalar programas, compatibilidad y fácil configuración de controladores propietarios *(firmware)* , aplicaciones básicas preinstaladas, cantidad de usuarios y de foros en línea. Ubuntu es la forma más sencilla de empezar a explorar este mundo. Si se requiere una distribución apegada más a necesidades específicas, el sitio:
`http://www.zegeniestudios.net/ldc/` es una buena ayuda. Tras una prueba rápida dará la mejor opción.

[8] Proyecto OLPC *http://laptop.org/es*

[9] De hecho la supercomputadora que encabeza la lista (noviembre,2008) del Top 500 en poder de cómputo en el mundo tiene un sistema operativo Linux. Ver : *www.top500.org* .

[10] En realidad a este proyecto se le llama Kubuntu. Pero la distribución esencialmente es la misma.

### 1.2.2. Linux práctico

**Los requisitos mínimos para Ubuntu**

PC

- CPU: Intel Pentiun III o AMD Athlon a 600 MHz o superior.
- RAM: Al menos 384 MB para ejecutar el LiveCD o 256 MB para ejecutar el instalador.
- Disco duro de 3 GB o más.
- Tarjeta de red y conexion a internet (muy recomendable)
- Tarjeta gráfica: Mínimo de 2 MB de video ó 32 MB con aceleracion 3D (recomendable). Para mejor rendimiento visual.
- Opcional:Tarjeta de sonido

Si no se cumple con los requerimientos se puede optar por instalar *Debian* o *DSL (Damn Small Linux)*.

**Obtención del software**

Ubuntu puede ser descargado de dos posibles formas. Mediante su sitio: `http://www.ubuntu.com/getubuntu/download` o por medio de un archivo *Torrent*. En ambos casos debemos bajar una versión dependiendo de la arquitectura del procesador. Si es un procesador de 64 bits su arquitectura será AMD64, x86-64, o x64 (todos estos sinónimos). Al utilizar esta versión se optimiza el rendimiento del procesador de 64 bits. Los procesadores de la serie *Athlon 64 y Core2* son compatibles con ésta. Para más información consulta la página del fabricante. Por el contrario, si es un procesador viejo o no se sabe cuál es su arquitectura entonces se debe de bajar la versión para 32 bits, ésta funciona en ambas arquitecturas.

El archivo que se baja tiene la extención *.iso* (ubuntu[*VERSION*].iso) se debe de *quemar* como *imagen de CD*. Actualmente casi todos los programas para grabar CDs traen esta opción. Al finalizar la grabación se habrá obtenido una copia del disco de instalación del Sistema Operativo Ubuntu.

Otra forma de apropiarse de una copia de Ubuntu es pidiendo el disco de instalación desde la página del proyecto [11] la entrega es gratuita pero tarda alrededor de diez semanas.

**Arranque**

Es necesario arrancar la computadora desde el CD [12], para ello reinicia tu equipo con el disco grabado en el lector de CD/DVD principal. Al arrancar (ver figura 1.1), aparecerá una pantalla en la que nos da la opción de seleccionar el idioma. Una vez hecho esto nos aparece la pantalla de

[11] `https://shipit.ubuntu.com/`

[12] Es posible que la máquina no esté configurada para *cargar* desde CD. Esto se soluciona configurando el *BIOS* de la PC. Frecuentemente sale la opción *Setup* al encenderla. Se debe buscar la opción *Boot loader* o *Boot order*

bienvenida. Seleccionamos en ésta la primera opción si deseamos ejecutar el entorno de Ubuntu corriendo desde el *CD-Rom (Live-CD)* o si tenemos claro que queremos hacer la instalación en el disco duro seleccionamos la segunda opción.


Figura 1.1: Pantalla de instalación

Si elegimos la primera opción, tras unos minutos, dependiendo del equipo, el entorno gráfico de Ubuntu se habrá cargado en la memoria. Así se puede explorarlo probando las aplicaciones para tener una idea de lo que Ubuntu puede hacer sin tener nada instalado. Hay que tener en cuenta que una vez instalado en el disco duro es mucho más rápido que al ejecutarlo *en vivo*. Para iniciar la instalación se debe hacer doble clic en el ícono del escritorio que dice : Install. Si en el paso anterior elegimos la opción dos pasamos directamente al proceso de instalación.


**Atención:** Antes de proceder a la instalación es muy recomendable respaldar la información importante que tengamos, es posible que haya errores en la instalación o el particionado del disco y se pierda irremediablemente la información.

**Instalación y configuración**

Pasos a seguir:

1. Elegir el idioma.
2. Elegir la zona horaria. Pulsar sobre alguna zona para ampliar el mapa y después sobre la ciudad representativa del huso horario deseado.
3. Elegir la distribución del teclado. El teclado común en español para México debería estar seleccionado ya, lleva el nombre de *teclado «Latin America»* (la). Asegúrate de que esto es así escribiendo en la caja de texto que hay en la parte inferior, pulsando algunas teclas específicas del español, como la «ñ» y los acentos.

4. **¡Cuidado!** Este es uno de los pasos más importantes y delicados. Se trata de indicar dónde se debe instalar Ubuntu. Escoje la opción correcta o podrías formatear una partición no deseada. Si se va a compartir el disco duro con otros sistemas operativos hay que hacer un respaldo de la información antes. Hay tres formas de instalación en el disco duro:

   *a*) Formatear todo el disco duro. Elije ésta si deseas borrarlo todo y usar todo el disco duro como único para Ubuntu. Es la opción más fácil y menos problemática.

   *b*) Espacio libre contiguo. Ubuntu usará una parte del espacio libre del disco duro para instalarse. Ésta es la opción más recomendable si deseas conservar tu antiguo sistema operativo o alguna partición con tus datos. Es una opción muy habitual para aquéllos que desean seguir también con un primer o segundo sistema operativo como Windows o MacOSX.

   *c*) Particionamiento manual. Con esta opción, podrás especificar cómo serán las particiones de forma manual. Ésta no es la mejor opción si nunca se ha hecho una partición o se ha instalado Linux antes.
   **Se recomienda sólo para usuarios expertos, se pueden perder todos los datos y el sistema operativo original.**

5. En este paso de la instalación se preguntarán los datos de los usuarios : Nombre real y nombre de usuario. Por ejemplo, el nombre real podría ser: «*Molgor Tourette*» y el nombre de usuario: «`molgor`». Después se tendra que escoger una contraseña y un nombre para la computadora.

6. En la pantalla siguiente el instalador mostrará los datos para que revisarlos. Asegúrate de que todo está en orden y pulsa *Siguiente* para comenzar a copiar los archivos de Ubuntu al disco duro. Si durante el proceso de instalación tenemos conexión a internet, el programa de instalación se conectará y descargará los paquetes necesarios para dejar nuestra instalación de Ubuntu completamente en nuestro idioma.

7. Si todo se instaló correctamente, al final la instalación preguntará si se desea reiniciar (sin el disco) o continuar usando la sesión *en vivo*. Si optamos por la primera opción del menú inicial nos solicitará reiniciar el sistema para completar la instalación.

Esta fue una descripción de la instalación, puede haber algunas diferencias. Si se tienen dudas el sitio de Ubuntu : `http://www.ubuntu.com/` tiene las respuestas a la mayoría de las preguntas.

### 1.2.3. El entorno gŕafico

Un entorno gráfico es una colección de programas que se comunican entre sí y nos dan la facultad de manejar ventanas, tener barra de menú principal, ejecutar programas, ver videos, etc. Estas colecciones de programas dependen a su vez de programas base llamados *bibliotecas* para su funcionamiento y es por esto que podemos hacer una distinción de entornos gráficos con base en las diferentes bibliotecas que utilizan. Cada entorno tiene sus rasgos característicos, programas preferidos para manejo de archivos, navegador web,

mensajero web, etc. Al contrario de Windows. Linux ejecuta la interfaz gráfica como si fuera un programa más en su pila de ejecución. Esto implica que Linux puede funcionar con o sin interfaz gráfica (servidor X). De hecho cuando se quiere utilizar todo el potencial del sistema operativo se suele hacer sin ejecutar ésta. Dependiendo de las necesidades de los usuarios se pueden omitir los entornos gráficos y como veremos más adelante trabajar sólo con líneas de texto/comandos. Existen una variedad notable de entornos gráficos diferentes pero los más populares y por ende con más nivel de desarrollo son:

Gnome Un entorno 100 % software libre. Inicialmente desarrollado por el mexicano Miguel de Icaza, matemático de la Facultad de Ciencias. Gnome pronto tuvo apoyo de muchos programadores y se convirtió rápidamente en un entorno completo con juegos, hojas de cálculo, editor de textos, aplicaciones estéticas, etc. En la actualidad Gnome es de los entornos más utilizados por su fácil acoplamiento con otros programas, incluidos efectos 3D y su buen desempeño en equipos viejos.

KDE Criticado por los más ortodoxos de la FSF debido a que KDE utiliza una librería llamada *qt* que no estaba liberada bajo licencia GPL. Esto ocacionó que su desarrollo se limitara en su mayor parte a grandes compañías desarrolladoras como Novell y que los usuarios, en general, no pudieran aportar algo que fuera totalmente libre. Actualmente *qt* ha sido liberada bajo la GPL restringida (LGPL) y recientemente todo el entorno fue adquirido por Nokia. KDE es a mi parecer el entorno más completo y amigable que hay para Linux. Las aplicaciones son completas y hay más efectos visuales (útiles e inútiles). Todo esto con un costo adicional en rendimiento, comparado con los otros entornos. Si tu computadora tiene suficientes recursos disponibles, KDE [13] es una buena opción para comenzar. En la figura 1.2 se muestra una imagen de un escritorio en KDE 3.x.

Xfc Este entorno fue originalmente pensado para máquinas con recursos muy limitados, por lo que no tenía muchos efectos visualmente asombrosos. Sin embargo ha cobrado tanta popularidad que muchas personas han trabajado en su diseño y estilo, creando un entorno elegante y rápido. Este entorno es ideal para equipos viejos o para usuarios con mucha necesidad de computo.[14]

### 1.2.4. Modularidad en Linux y el paradigma orientado a objetos

Linux originalmente fue desarrollado por programadores, por lo que la estructura del sistema operativo está planeada de forma tal que pueda ser escalable y actualizable fácilmente de acuerdo a nuestras necesidades de trabajo.

Existen muchas formas de programar a las que los desarrolladores llaman paradigmas de programación. Éstas difieren escencialmente en cómo abordar

[13] La distribución de Ubuntu que viene preparada con KDE se llama Kubuntu más información en: `www.kubuntu.org`

[14] La distribución de Ubuntu correspondiente se llama Xubuntu más en: `www.xubuntu.org`

Figura 1.2: Mi escritorio en KDE ( KDE Desktop Environment )

el problema a resolver. Actualmente el paradigma más utilizado es el orientado a objetos, que consiste en crear un objeto abstracto con atributos, funciones (métodos) y parámetros de entrada y salida. Es como una máquina que crea muchos objetos con atributos iguales, en un sentido sustantivo, pero diferentes en un sentido adjetivo.

Cuando se tiene un problema muy grande y complejo el lema maquiavélico *"Divide y venceras "* es la mejor opción; para esto el paradigma orientado a objetos no tiene comparación.

Por ejemplo, pensemos en el siguiente problema:

**Queremos hacer una pequeña base de datos de bichos que hayamos colectado en las localidades visitadas en una práctica de campo.**

.

La forma de implementar el problema sería de la siguiente manera:

1. Creamos una clase [15] que defina un objeto de tipo **Bicho** con los siguientes atributos:
   - `string` Localidad
   - `float[]` Georreferencia
   - `string` Especie
   - `string` Colector
2. Creamos las funciones [16] (acciones) que va a realizar el objeto **Bicho** estas pueden ser:
   - `void` cambiaLocalidad (`string localidad`)
   - `float[]` daGeorreferencia (`void` )
   - `string` quienColectó (`void`)
   - `void` guardaEnDisco(`void`)

[15] Una fabriquita
[16] En muchos lenguajes como Java se les llama métodos

Analicemos un poco esto. En la mayoría de los lenguajes es necesario declarar qué tipo de dato es cada atributo. Esto quiere decir que debemos especificar para cada uno el tipo; si es de punto flotante *(float)*, entero *(int)*, cadena de caracteres *(string)*, booleano *(bool)*, etc. De la misma forma, en los métodos también es conveniente ( en algunos lenguajes obligado) decirle al compilador el tipo de dato que va a recibir y el tipo de dato que va a devolver tras efectuar las instrucciones ahí descritas. Observemos el caso de la función: `void` *cambiaLocalidad* `(string localidadnueva)`. Aquí el método se llama *cambiaLocalidad* y su función es la de cambiar la localidad del objeto. La sintaxis difiere en cada lenguaje pero en general la palabra del principio define el tipo de dato que va a ser devuelto al finalizar la función. En este caso utilizamos la palabra reservada `void`, que significa literalmente *vacío*, lo que implica que este método no va a devolver nada. Tiene sentido pués el método lo único que hace es cambiar el atributo `localidad` del objeto al cual le aplicaremos la función. Esta nueva localidad la vamos definir en la siguiente parte del método, en la parte (`string localidadnueva`). A esta sección se le llama: *parámetros de entrada*. Es aquí donde le diremos al compilador lo que va a recibir, en este ejemplo sería una cadena de caracteres y a esa cadena introducida se le va a dar el nombre temporal de `localidadnueva`. Esta será una variable que únicamente estará definida al interior de la función *cambiaLocalidad*, por eso es temporal. Vemos que hay un tipo de dato distinto declarado en el atributo `float[] Georreferencia` y en el método `float[] daGeorreferencia ( void )`. Este tipo de dato lleva el nombre de *arreglo* y es esencialmente un vector de datos simples, en este caso `float[]` es un vector de valores de punto flotante, lo necesitamos así pues en la georreferenciación necesitamos tres valores, latitud, longitud y altitud. Aclarado esto es fácil ver porque el método `daGeorreferencia` necesita devolver un vector flotante.

Una vez definida la clase, en un archivo aparte se procede a crear el programa principal llamado *main* que es el que hará concreto el problema, implementación en el argot del programador. Hará referencia a la clase donde definimos el objeto **Bicho** utilizará sus atributos y funciones y construirá un objeto Bicho distinto por cada muestra que hayamos tomado del campo.

Por ejemplo: hemos regresado del campo con tres muestras: una planta, un hongo y una garrapata. Quisieramos meter estos datos en la base de datos para después hacer análisis o simplemente guardarlo en el disco.

En el programa *main* se puede hacer lo siguiente:[17]

- `Usa: Bicho ;`
  Aquí le decimos al lenguaje (compilador) que queremos usar la clase Bicho.

- `Bicho planta ( Cuetzalan, helianthum, Juan);`
  Esto crea (construye) un objeto de tipo Bicho llamado planta que se colectó en Cuetzalan, cuyo género es *Helianthum* y su colector fue Juan.

- `Bicho hongo(Zempoala,Amanita, Tobías);`

- `Bicho garrapata(San_Agustin, Ixodes, El Ingeniero);`

[17]Esto sólo es una explicación, cada lenguaje de programación tiene su propia sintaxis, pero la estructura es esencialmente la misma.

Supongamos ahora que nos equivocamos de localidad en el bicho llamado *hongo*. Como ya tenemos un *método* llamado `cambiaLocalidad(L)` sólo hay que invocar esta función, para hacer esto bastará con llamarlo con el operador ·, a esto se le llama *instanciar* un método. En el ejemplo seria así:

- `hongo.cambiaLocalidad(CuatroCienagas);`

Aquí se utiliza el método `cambiaLocalidad` y se le introduce como parámetro un nuevo nombre, i.e. CuatroCienegas. Ahora el bicho *hongo* tendrá como atributo de localidad CuatroCienegas.

Al crear la clase **Bicho** no nos preocupamos por utilizar ahí sus métodos, estos los utilizamos –instanciamos– sólo en la clase **main**. De esta forma podemos añadir cualquier elemento nuevo a la base de datos sin tener que modificar el archivo donde se define a la clase **Bicho**.

Esta forma de programación ha resultado muy efectiva para desarrollar aplicaciones colectivamente. Existen muchos lenguajes de programación orientados a este paradigma. Los más comunes son: C++, Java y Python. Este último es el más nuevo y sencillo de aprender, es una revolucion en el software libre por la facilidad con que se puede acoplar a otras aplicaciones. Recomiendo aprenderlo, Lutz y Ascher (1999) tienen un buen libro para empezar.

**¿Por qué es importante saber usar esto?**

Anteriormente comentamos que Linux se basa en *bibliotecas*. Las bibliotecas contienen código y datos que proporcionan servicios a programas de forma independiente. Esto permite que el código y los datos se compartan y puedan modificarse de forma modular. Algunos programas ejecutables pueden ser a la vez programas independientes y bibliotecas pero la mayoría de éstas no son ejecutables. La mayoría de los sistemas operativos modernos como los basados en *Linux* proporcionan bibliotecas que implementan la mayoría de los servicios del sistema. De esta manera, estos servicios se convierten en una ”materia prima”que cualquier aplicación moderna espera que el sistema operativo ofrezca. Cuando se ejecuta un comando o un programa, éste por lo general depende de distintas *bibliotecas* para su ejecución. Por ejemplo si utilizamos el reproductor de música Amarok (el reproductor más completo creado hasta ahora) éste utiliza una biblioteca para la interfaz gráfica (botones, colores, etc), otra para manipular el hardware de sonido, otra para decodificar y reproducir los archivos de música, otra para conectarse a internet y bajar información de la canción etc. Amarok depende de varias bibliotecas que son independientes unas de otras, a diferencia de Windows que muchos de sus programas son independientes y tienen sus propias bibliotecas. Por ejemplo el tener instalado varias versiones de *Microsoft Office* no significa que ambas versiones compartan bibliotecas, cada versión tiene las propias en lenguaje de máquina, es decir sólo ese programa las puede utilizar. Esta es una de las razones por la cual los paquetes en Windows ocupan siempre más espacio que los paquetes de Linux.

Esta estructura hace, en principio, que la aproximación a Linux por usuarios *no programadores* sea lenta y sin sentido cuando se compara con los S.O. comerciales. Sin embargo, es la razón de fondo que lleva a Linux a superar

por mucho a los otros sistemas. El éxito de la migración consiste en cambiar el paradigma de procesos independientes (como *Windows*) al de procesos dependientes comunicantes. Cada programa cumple su función y se comunica con los demás programas por medio de estándares establecidos llamados mensajes o llamadas a función. Esto permite que el sistema operativo de poder sea modular y pueda configurarse al gusto del usuario. Cuando creamos una clase, lo que estamos haciendo es creando un tipo de biblioteca que puede ser compartido con otros programas. Muchas de las herramientas libres útiles para nuestro oficio están implementadas bajo el paradigma orientado a objetos por lo que entenderlo cabalmente es esencial para poder realizar nuestras tareas con mayor eficiencia.

## 1.3. Instalación de aplicaciones

Prácticamente todo lo que se pueda hacer con software comercial se puede hacer en Linux con Software Libre. Los paquetes a los que estamos acostumbrados en Windows como: *Office, Messenger, Internet Explorer, Statistica, ArcMap, ArcInfo, etc* tienen su contraparte libre *OpenOffice, pidgin, Mozilla, GNU R, Quantum GIS y GRASS*, respectivamente. Más aun, en este mundo hay una gran variedad de paquetes diferentes que cumplen la misma función. El sitio: `http://osluz.unizar.es/aplicaciones` nos da una lista detallada de las aplicaciones libres equivalentes a programas comerciales. Sin embargo, como explicamos en la sección anterior, hay programas que son necesarios para muchos otros, lo que significa que para que se ejecute un programa, este debe saber específicamente dónde están los demás programas que necesita. A estos programas se les llama *dependencias* . Anteriormente las dependencias, programas y bibliotecas, convertian el proceso de instalación en un problema muy engorroso. Para solucionarlo, Debian inventó el `apt`, un sistema de gestión de paquetes que hace referencia a bases de datos de dependencias e instala todos los paquetes necesarios para que pueda funcionar el que queramos instalar, usando línea de comandos o si se prefiere con interfaz gráficá, ésta se llama *synaptic*.

### 1.3.1. Los repositorios. El limbo de los programas.

Un repositorio es un sitio centralizado donde se almacena y mantiene información digital, habitualmente bases de datos o archivos informáticos. En Ubuntu y su padre Debian, los repositorios son sitios de internet donde están almacenados una variedad grandísima de paquetes a instalar, incluidas todas sus dependencias. Cada repositorio almacena un grupo de programas específicos y una versión en particular. Por ejemplo, los programas con licencia GPL están en el repositorio base de Ubuntu. Si queremos instalar programas que no estén del todo liberados se tendrá que habilitar el repositorio *multiverse*, esto es necesario para instalar *flash player*. Esto se puede hacer con un *gestor de paquetes* como: Synaptic, Adept Manager o Aptitude. Recuerda, para ejecutar cualquiera de estos programas necesitas ser administrador. La figura 1.3 muestra la pantalla del gestor *Adept Manager* de KDE.

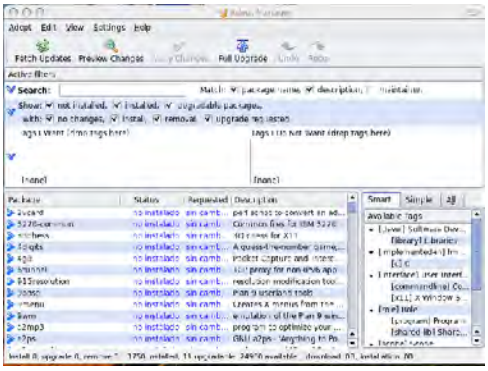


Figura 1.3: Gestor de paquetes *Adept*

La forma en la que funcionan estos programas es que hacen referencia al comando `apt-get` que a su vez hace referencia al comando `dpkg` y éste utiliza la lista de archivos instalables y sus dependencias. `apt-get` los baja de la red y los instala y `Adept` nos da una interfaz amigable. La diferencia con Gnome radica únicamente en este último, la interfaz es algo diferente, pero utilizan los mismos programas base. Para saber como funcionan estos comandos o cualquier se puede utilizar el comando `man` en la terminal escribiendo:

`man apt-get`, `man dpkg`, `man synaptic`; en general, `man [comando]`

### 1.3.2. La consola en Linux y sus comandos básicos

La consola o *shell* es el intérprete de órdenes y comandos de un sistema Unix/Linux. No hay que confundirla con el sistema operativo. El sistema operativo es el núcleo y la consola es una interfaz que nos proporciona utilidades de trabajo y permite establecer una relación con el núcleo. Es decir, la consola reconoce una serie de órdenes, las interpreta y lanza los procesos necesarios para su ejecución. Hay varios tipos de *shells*, entre los que se pueden mencionar:

- *Bourne Shell (sh)*
- *Korn Shell (ksh)*
- *Shell C (csh)*
- *Bourne Again Shell (bash)*

Éste último es el más popular y desarrollado. Para ver más información acerca de los diferentes *shells* se puede consultar la referencia (Figgins y Love, 2000).

#### Entradas y salidas estándares y de errores

Un sistema Unix dispone de tres vías o canales para comunicarse con el exterior de forma estándar. Una de ellas, *la entrada estándar*, se utiliza para introducir datos en la consola; de forma predeterminada está asociada al

teclado. La *salida estándar* se utiliza para mostrar información de salida de una ejecución. De forma predeterminada está asociada al monitor.

Por último, existe un canal dedicado a mostrar la salida de errores, que de forma predeterminada está asociado a la *salida estándar*. Cuando la ejecución de una orden falla por cualquier motivo, el sistema lo notifica a través de este canal. Normalmente, como el canal va asociado a esta salida, los errores aparecen en pantalla. En múltiples ocasiones nos puede interesar redirigir alguna de estas salidas a otro canal; para realizar esto utilizamos los siguientes metacaracteres :

$<$ Redirige la entrada estándar. Podemos utilizar este metacaracter cuando queramos sustituir una serie de entradas por teclado por el contenido de un fichero. De esta forma en vez de escribir los comandos a ejecutar, los podemos tener en un archivo de texto.

$>$ Redirige la salida estándar. Si esta redirección es a un fichero, lo crea nuevo en caso de que no exista, y si existe elimina su contenido previo. Un descuido con la redirección nos puede hacer perder un fichero fácilmente.

$>>$ Redirige la salida estándar. Si esta redirección es a un fichero, y el fichero existe, la añade al final de éste, si no existe lo crea. Utilizando esta redirección podremos añadir fácilmente una línea a un fichero. Utiliza la operación *concatenación*.

$2>$ Redirige la salida de errores. Si esta redirección es a un fichero, lo crea nuevo en caso de que no exista, si existe elimina su contenido previo.

$2>>$ Redirige la salida de errores. Si esta redirección es a un fichero, y el fichero existe la añade al final de éste, si no existe lo crea.

## Algunos comandos básicos

Esta es una lista de los comandos más utilizados en la consola. Para ver más comandos útiles revisa el apéndice al final del libro

- **cat** Muestra el contenido del archivo en pantalla en forma continua, el prompt retornará una vez mostrado el contenido de todo el archivo. Permite concatenar uno o mas archivos de texto.
  `Sintaxis: cat nom_archivo.`
- **cd** Cambia de directorio.
  `Sintaxis: cd nom_directorio.`
- **chmod** Cambia los permisos de lectura, escritura o ejecución de los archivos. r:lectura w:escritura x:ejecución +: añade permisos -:quita permisos u:usuario g:grupo del usuario o:otros
  `Sintaxis: chmod permisos nom_archivo`
- **cp** Copia archivos en el directorio especificado.
  `Sintaxis: cp nom_archivo nom_directorio.`
- **grep** Busca patrones en archivos. Escribe en salida estándar aquellas líneas que concuerden con una cadena de caracteres.
  `Sintaxis: grep [-cilnv] expr nom_archivos.`
- **history** Lista los comandos más recientes que se hayan ejecutado en la consola.
  `Sintaxis: history`
- **ls** Da una lista de los archivos y directorios dentro del directorio de trabajo en el cual se ejecute.
  `Sintaxis: ls.`
- **rm** Remueve o elimina un archivo. Utilizar el sufijo -r para eliminar recursivamente todos los archivos de ese directorio.
  `Sintaxis: rm nom_archivo.`

- **ssh (Secure Shell Client)** Interfaz de consola remota. La comunicación entre el usuario y la máquina remota está encriptada para evitar que terceras personas lean la información. Se debe introducir la dirección del servidor o *host* y el nombre de usuario. Para iniciar sesión es necesario que se introduzca la contraseña del usuario.
  `Sintaxis: ssh usuario@maquina_remota.`
- **sudo** Ejecuta un comando dado a nombre del super usuario.
  `Sintaxis sudo [comando]`

El mundo de Linux es muy grande. Este capítulo ha sido pensado para cubrir las generalidades del sistema operativo, en particular de Ubuntu. Faltarían muchas cosas más que mencionar pero saldrán a medida que nos vayamos familiarizando con los sistemas de información geográfica (SIG). El aprendizaje de los SIG de código abierto es un poco más lento que con los SIG comerciales, pero vale la pena por su capacidad de análisis y costos.

# Capítulo 2

# Conceptos básicos de los Sistemas de Información Geográfica

*Qué inapropiado llamar Tierra a este planeta,*
*cuando claramente es Océano.*
–Arthur C. Clarke.

Esta sección tiene como objetivo introducir los conceptos elementales para trabajar con cualquier SIG, sea libre o no. La primera parte nos muestra como representamos a la superficie terrestre mediante la elaboración de mapas. Aquí abordaremos algunos conceptos básicos de cartografía, como *modelos terrestres, proyecciones y coordenadas*. La segunda sección muestra como podemos representar fenómenos en la superficie terrestre, o mejor dicho, en algún modelo de la superficie terrestre. Revisaremos los tipos de datos espaciales utilizados en los SIG, *raster y vectorial*, algo de sus *atributos* y por último representaciones volumétricas. Al terminar este capítulo el lector podrá comprender y manejar, al menos básicamente, cualquier Sistema de Información Geográfica.

## 2.1. Algunas definiciones de los SIG

Existen varias definiciones de los *Sistemas de Información Geográfica* (SIG). Una de las más utilizadas es la descrita por Burrough y McDonnell (1998) (Un SIG es) –*Un conjunto de herramientas para reunir, introducir, almacenar, recuperar, transformar y cartografiar datos espaciales sobre el mundo real para un conjunto particular de objetivos.*

A medida que se han desarrollado sistemas más sofisticados se han establecido otras definiciones más relacionadas con las áreas en donde se utilizan. Por ejemplo un enfoque computacional para los SIG es el descrito por Longley et al. (2005), un texto clásico en esta ciencia: – *Un Sistema de Información Geográfica es un sistema computacional que consiste en una base de datos que almacena*

*información espacial y descriptiva de un entorno geográfico como parte del mundo real; además de permitir la entrada, mantenimiento, análisis, transformación, manipulación y presentación de datos espaciales, de algún punto geográfico en particular.*

Como corolario a esta definición el libro de DeMers (2009) ejemplifica el enfoque ambiental con la siguiente afirmación: *las aplicaciones de los SIG son útiles desde el inventario de los recursos naturales y humanos hasta el control y la gestión de los datos catastrales, de propiedad urbana y rústica (catastro multipropósito), la planificación y la gestión urbana y de los equipamientos*.

En conjunto estas tres definiciones conceptualizarán, en lo abstracto, los Sistemas de Información Geográfica de que hablaremos a lo largo de todo el libro.

## 2.2. Representaciones de la superficie terreste

Cuando vamos por la carretera y vemos las montañas y valles, nos damos cuenta que el suelo no es en absoluto uniforme. De hecho, raras veces estamos parados en una superficie verdaderamente plana. Parece que, sin importar la escala podemos asegurar que la Tierra no es plana, lisa o como dirían los matemáticos, *derivable*. A pesar de esto, desde hace siglos, los humanos hemos construido modelos simplificados de la compleja superficie terrestre para resolver problemas relacionados con el espacio en el que vivimos. e.g. navegación, reparto de tierras, caminos y avisos (USGS Mapping Applications Center, 2000). Gracias a estos modelos, es posible construir mapas, asignar coordenadas a lugares en el mundo y encontrar rutas óptimas.

### 2.2.1. Modelos de la Tierra

Existen tres modelos básicos para representar la superficie terrestre, que se representan el orden del más simple hasta el más complejo:

- Modelo esférico
- Modelo elípsoidal
- Modelo geoide gravimétrico.

#### El modelo esférico

La hipótesis de la cual parte este modelo es la de suponer que la distancia del centro de la Tierra a cualquier lugar de su superficie es la misma; aproximadamente 6371 km (Longley et al., 2005). La figura 2.1 nos muestra un diagrama de una superficie esférica en el espacio de tres dimensiones. El centro de masa de la Tierra queda en el eje de rotación. Al plano que corta perpendicularmente a este eje y que pasa por el centro se le llama *Ecuador*. Podemos definir entonces un conjunto de planos, paralelos al eje de rotación, que pasan por el centro y que cortan al *Ecuador* en ángulos constantes de 6 grados, a éstos *cortes* los llamaremos *meridianos* ( ver figura 2.2). Por *convención* los *cero* grados se localizan en el *Real Observatorio de Greenwich* en el Reino Unido. A partir de ahí se construyen 30 meridianos al *Este* y 30 al *Oeste*, Suman 60 meridianos en total que abarcan los 360 grados del plano ecuador. Es común

nombrar a los meridianos del *Oeste* con números negativos y a los del *Este* con positivos. Así se construye la coordenada *longitud*.

La otra coordenada, *latitud* , (ver figura 2.2). Se divide el ángulo entre el ecuador y el eje de rotación en sentido norte o sur, nombrando de 0 a 90 grados *latitud norte* y de 0 a 90 grados *latitud sur*, respectivamente.

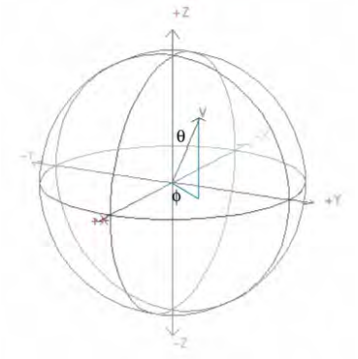



Figura 2.1: Modelo básico de una esfera en tres dimensiones

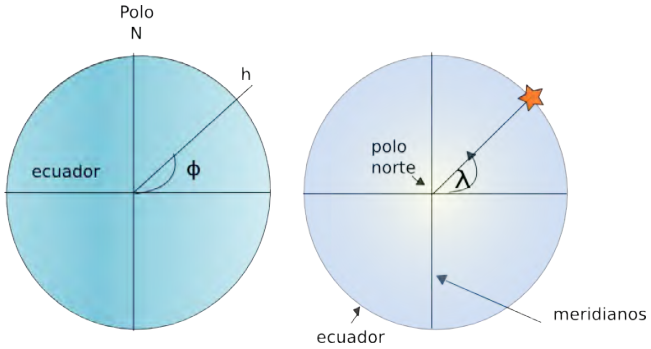



Figura 2.2: Modelo esférico de la Tierra, en donde $\phi$ es la latitud y $\lambda$ la longitud

**Esferoide**

El modelo esférico, dada su simplicidad, funciona sólo para darnos una idea de como están definidas las coordenadas geográficas, en particular la *longitud*. Un modelo más preciso para definir puntos en la Tierra es el modelo elípsoidal o *esferoide*, porque la distancia del centro de la Tierra a los polos es menor que la del centro al *Ecuador*. Esta proporción de achatamiento se denota con la letra $f$ y se calcula con la siguiente fórmula:

$$f = \frac{diámetro\ polar - diámetro\ ecuatorial}{diámetro\ ecuatorial} \simeq \frac{1}{300}$$

Por tanto, la distancia del centro de la Tierra al Ecuador es la distancia del centro a los polos mas un número cercano a $\frac{1}{300}$ (Iliffe, 2003). Esto le da a la Tierra una forma elíptica o "achatada". El objeto geométrico que describe

mejor esta forma se llama *elipsoide de revolución* y se forma al girar 360 grados una elipse -de dos dimensiones- sobre uno de sus ejes. Para el caso de la Tierra sería sobre el eje de rotación. La figura 2.3 muestra la dirección de este giro. La elipse tiene dos parámetros que la definen, que son las distancias del origen a: *i)* el eje $x$ y *ii)* al eje $y$; a la mayor se le llama *semieje mayor (a)* y semieje menor($b$) a la segunda. Para la Tierra, el valor aproximado de $a$ es igual a 6, 378. 137 Km (Iliffe, 2003). Como sabemos la proporción de *achatamiento ( f )* podemos utilizar la fórmula siguiente para calcular $b$.

$$b = a(1 - f)$$

Que es aproximadamente: 6,356.900 Km. Como vemos, es posible definir en su totalidad un elipsoide sólo con dos parámetros, que pueden ser: $b$, $a$ o $f$. Estos últimos son los más utilizados para determinar elipsoides y llevan el nombre de *parámetros elípticos*. Durante más de 200 años en diferentes paises del mundo se ha trabajado para encontrar el elipsoide que mejor se aproxime a la forma de la Tierra en algún lugar en particular, con el fin de poder producir mapas precisos. Numerosos paises crearon sus propios modelos con parámetros $a$ y $f$ diferentes y generalmente no estaban centrados en el centro de masa de la Tierra. Fue hasta la decada de los 60 con el desarrollo de la tecnología satelital y, por desgracia, de los misiles interoceánicos, que fue necesario llegar a una estandarización internacional. Hoy en día el modelo internacional más aceptado es el **WGS84** (World Geodetic System of 1984) , aunque también es frecuente encontrar mapas o datos que usan como modelo otro elipsoide. Entre los más conocidos están: *NAD27, NAD83, Clark66, GRS80, Helmert*. Los usuarios de los SIG frecuentemente tenemos que hacer conversiones entre estos *datums* [1] para uniformizar los datos que usamos. De otro modo, al relacionar un mapa con otro es posible que no correspondan adecuadamente.

**Coordenadas** Una vez construida la superficie de revolución será necesario definir sus coordenadas. Para el caso de la *longitud* estará definida de la misma manera que en el modelo esférico. De hecho, todos los planos paralelos al Ecuador son círculos (por como fue construida la superficie). La partición de los meridianos será la misma, i.e. a cada 6 grados empezando por *Greenwich*. La *latitud* será ligeramente diferente.

**Construcción:** Consideremos un punto $p$ en el esferoide. Hagamos pasar un plano tangente [2] a la superficie en el punto $p$, al que llamaremos Π. Dado Π, es posible construir una recta perpendicular a éste y que pase por $p$. A esta recta se le llama *normal esferoidal* . Esta recta intersecta con el *Ecuador* en algún punto. El ángulo resultante entre el *Ecuador* y la *normal esferoidal* definirá la *latitud*. La figura 2.3 esquematiza esta construcción.

La latitud varía entre los 0-90 grados Sur y los 0-90 grados Norte. Es común encontrar coordenadas negativas para el hemisferio sur y positivas para el hemisferio norte. Usualmente se utiliza la letra griega *phi* $\phi$ como símbolo para la latitud y la letra griega *lambda* $\lambda$ para la longitud. En términos matemáticos $\phi$ y $\lambda$ toman los valores: $-90 \leq \phi \leq 90$ y $-180 \leq \lambda \leq 180$

[1] Explicación detallada más adelante

[2] i.e. Un plano que toque a la superficie sólo en un punto

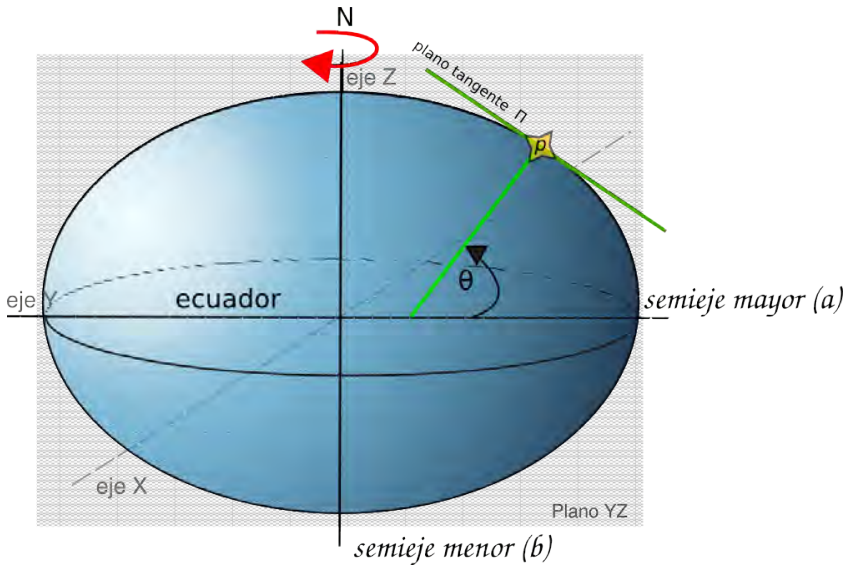



Figura 2.3: Modelo elipsoidal o *esferoide* de la Tierra. El ángulo $\phi$ es el valor para la *latitud*. Nótese que éste se forma por la intersección de la *normal esferoidal* con el Ecuador. La estrella amarilla representa un punto cualquiera en la superficie. La flecha de giro al norte indica la dirección de giro en el plano YZ para crear el esferoide de revolución.

### El Geoide

El modelo esferoidal descrito anteriormente es una buena aproximación de la forma de la Tierra, pero no es una representación exacta. La superficie del planeta depende de factores bióticos y abióticos, pero la fuerza principal que *moldea* la superficie es la gravedad. Cuando una fuerza de este tipo actúa en una superficie recibe el nombre de *superficie equipotencial*, definición que hace referencia a la conservación del trabajo (fuerza por distancia); es decir el trabajo de ir a un lugar y regresar al punto de partida es cero. [3] La corteza terrestre está llena de irregularidades y además está en continuo movimiento. Una forma menos compleja para caracterizar la superficie es con el nivel medio del mar, que es perpendicular a la gravedad, por lo que sigue correspondiendo a una *superficie equipotencial*.

A la *verdadera* forma de la Tierra se le conoce como *Geoide*, (ver figura 2.4) y se define como *la superficie equipotencial que más se aproxime al nivel medio del mar* (Iliffe, 2003). Como el Geoide es muy irregular, para representar puntos en él se le hace coincidir con algún modelo *esferoidal*. Ambos tienen que coincidir en el origen. Matemáticamente es una proyección del campo vectorial equipotencial dependiente de la gravedad sobre el espacio en coordenadas elípticas. De esta forma es posible asignarle a cada punto del Geoide uno y sólo un punto en el elipsoide. El lector quisquilloso se podrá dar cuenta que la dimensión del Geoide así como del elipsoide es dos, por tanto son topológicamente iguales

[3] Si una función vectorial $F : \mathbb{R}^m \longrightarrow \mathbb{R}^n$ es el gradiente de una función escalar $\phi : \mathbb{R}^n \longrightarrow \mathbb{R}$ entonces $\phi$ se llama función potencial de $F$. Para una explicación más detallada ver: (Apostol, 2001).

a un plano. La Tierra y todo lo (material) que hay en ella está en un espacio de tres dimensiones. [4] Esto quiere decir que para tener un buen modelo de coordenadas necesitamos de una tercera variable. Esta será la *altura* o *altitud* y se define como la diferencia entre el Geoide y el elipsoide. Es decir, si $p_0$ es un punto en el Geoide, por como fue construido el elipsoide, existe un punto $q_0$ en él, tal que apunta en la misma dirección que $p_0$ pero que no necesariamente tiene la misma magnitud, es decir, son de la forma:

$$p_0 = \mu q_0$$

Para algún valor $\mu \in \mathbb{R}$. Como son vectores podemos calcular su norma [5]. La diferencia entre la norma de $p_0$ y la de $q_0$ nos dará la *altitud*. Es común encontrar el término *separación geoide-esferoide (N)* refiriéndose también a la altura (ver figura 2.5).

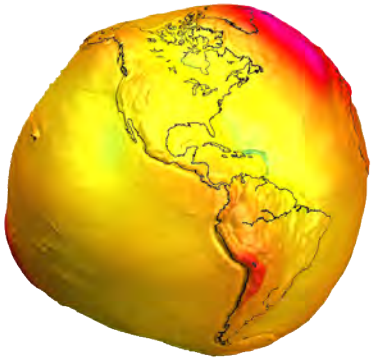


Figura 2.4: Representación del Geoide. Debido a su irregularidad es necesario construir un modelo elipsoidal que se adapte mejor a nuestras necesidades. Tomado de: *http://op.gfz-potsdam.de/grace/results/grav/g003_eigen-cg01c.html*

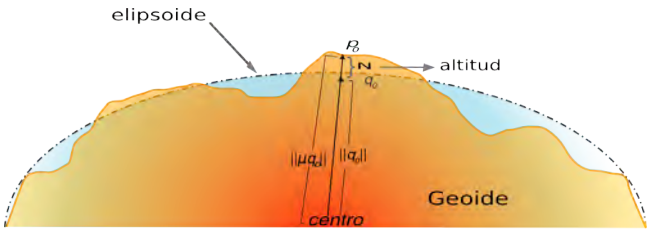



Figura 2.5: Vista de una sección en la cual coinciden en un punto el Geoide y el elipsoide. La altura *N* está definida como la diferencia de las distancias de $p_0$ y $q_0$.

El Geoide se puede hacer coincidir con muchos modelos elipsoidales distintos. Hemos visto anteriormente que con dos parametros $a$ y $f$ es posible

[4] Muy al pesar de los seguidores de la teoría de cuerdas.

[5] La distancia del punto al centro. i.e. Para tres dimensiones $\|p\| = \sqrt{x_p^2 + y_p^2 + z_p^2}$

definir un elipsoide. También vimos que cada elipsoide tendrá diferencias en las distancias del Ecuador y de los polos, dependiendo de $a$ y de $f$. Con esto claro podemos definir de una vez por todas el concepto de *datum*.

**¿ Qué es un *datum* ?**

Intituivamente el mejor modelo elíptico que se debería usar en cartografía y por esto en la geomática debería ser el elipsoide, cuyos parametros $a$ y $f$ sean los más aproximados al tamaño de la Tierra. Es decir, cuyo parámetro $a$ sea el radio del Ecuador y $f$ la proporción de éste con los polos (pág 21 sección 2.2.1).

Sin embargo, cuando construimos un elipsoide con estas características, como el *WGS84*, obtendremos errores considerables, que pueden oscilar entre $(-100, 100)$ metros en cada punto. Estos errores se pueden acumular dependiendo del análisis que realicemos. Para corregir esto es necesario cambiar los parámetros del esferoide, para que en el área de estudio, el Geoide y el elipsoide coincidan en un punto, llamado *punto fundamental*. Con éste podemos crear un nuevo elipsoide con sus correspondientes parámetros $a$ y $f$. En este punto la altura $N$ es cero.

El **datum** es la información necesaria para construir un elipsoide tangente a una zona, región o país de estudio. Es decir es el conjunto de parámetros que establecen el origen para las coordenadas terrestres latitud y longitud. Dos datums distintos tienen orígenes distintos. El *datum* esta compuesto por:

- Los parámetros $a$ y $f$ de algún elipsoide adecuado.
- El *punto fundamental*

Con el *datum* podemos tener distancias y áreas precisas de cualquier zona. En la práctica se pueden encontrar diferentes *datums* con el mismo modelo de elipsoide (e.g GRS90 o WGS84), i.e. parámetros iguales. Sin embargo la diferencia está en su *punto fundamental*, i.e. en su origen. Cuando estemos haciendo análisis de algún tipo es **muy importante** que todos nuestros datos: mapas, puntos, rutas, etc. Esten georeferenciados bajo el mismo *datum* y la misma proyección de mapa. Los SIG descritos en este manual tienen herramientas de conversión de *datums* que veremos más adelante.

**Tipos de *datums*** Los datums se clasifican en dos grandes grupos; *datums globales* y *datums locales*.

Los *datums* locales están diseñados para tener una representación precisa de una región o de un país. Existe una variedad muy grande de *datums* locales. Pueden tener en común modelos elipsoidales; la diferencia estará en su *punto fundamental* , el punto de origen del sistema de coordenadas. Un ejemplo de este tipo es el *North American Datum, 1927* (NAD27) usado ampliamente en México hasta finales del siglo pasado.

Los *datums globales* generalmente han sido creados para tener una representación exacta de toda la Tierra. Tienen su punto de origen en el centro de masa de la Tierra, son *geocéntricos*, y sus parámetros son los que mejor se aproximan a los del Geoide. En este caso el *datum* corresponde con el elipsoide. Estos *datums* se usan para aplicaciones mundiales, en especial para los sistemas de posicionamiento global.

Desde que empezó la tecnología satelital, en los comienzos de los años sesenta, se ha intentado crear un *datum* estándar para uso a nivel mundial. El más utilizado hoy en día es el *World Geodetic System 1984* (WGS84) mantenido por el departamento de defensa de los Estados Unidos. Los parámetros que utiliza (G, 1993) son: $a$ = 6378137 y $f$ = 1/298.257223563 . Otro *datum* global es el *Geodetic Reference System 1980* (GRS80), que es similar al WGS84 pero con un factor de achatamiento ($f$) distinto.

**Consideraciones para México** Hasta 1998 México utilizaba oficialmente el *North American Datum 1927* (NAD27) (Navarro). Por disposición de INEGI éste fue cambiado por el ITRF92 que es mucho más preciso y tiene tres dimensiones espaciales. [6]

El *datum* ITRF es global y se llama: *International Terrestrial Reference Framework* . Es un *datum geocéntrico* (i.e. su origen es el centro de masa de la Tierra) muy preciso llegando al nivel de milímetros. Al ser tan exacto, toma en consideración el movimiento tectónico de las placas continentales, por lo que tiene varias *familias* como ITRF92, ITRF94 e ITRF96.

A apartir de las modificaciones hechas a la norma técnica de levantamientos geodésicos en 1998 el INEGI estableció que todo punto perteneciente a un levantamiento geodésico deberá estar referido al *Marco de Referencia Terrestre Internacional (ITRF)* del Servicio Internacional de Rotación de la Tierra (IERS) de 1992. Éste es el nuevo *Sistema Geodésico de Referencia* oficial para México (Navarro). Los parámetros de los elipsoides para WGS84 e ITRF92 son prácticamente los mismos, la única diferencia está en el factor de achatamiento $f$. El cuadro 2.1 muestra esta diferencia.

Cuadro 2.1: Comparación de los parámetros de elipsoide utilizados en dos *datums* globales. ITRF92 utiliza el elipsoide GRS80.

| Parámetro | WGS84 | ITRF92 |
|---|---|---|
| Semieje mayor ($a$) | 6378137 m | 6378137 m |
| Semieje menor ($b$) | 6356752.3114 m | 6356752.3114 m |
| Achatamiento ($f$) | 1/298.257222101 | 1/298.257223563 |

Prácticamente el ITRF92 es idéntico al WGS84 para resoluciones mayores a 10 cm. INEGI ha preferido usar este datum porque, a diferencia del WGS84, la información es abierta, se tienen más de 300 estaciones alrededor del mundo, una en México, y la calibración se realiza con cinco técnicas distintas que hacen referencia a objetos extraterrestres como estrellas y pulsares.

**Consideraciones al georreferenciar.** Cuando georeferenciamos algún punto en un lugar, generalmente lo hacemos con el datum WGS84 o NAD27 . La precisión del punto depende del dispositivo GPS, de la red de *efemérides* satelitales y del *datum*. TODOS los puntos a analizar deben estar referenciados bajo el mismo *datum* y elipsoide. Hemos visto que para datos con *datum* WGS84 no hay diferencias significativas con el ITRF92. Sin embargo sí las hay para

[6] El datum NAD27 no tiene valores de altura, sin embargo, gran parte de la cartografía existente está referenciada con él.

los datos con referencia NAD27 ya que, dependiendo del lugar donde sean tomados pueden existir variaciones entre 100 y 200 metros. En estos casos sí será necesario hacer conversiones entre *datums*, que veremos en el siguiente capítulo.

Existe otra alternativa al GPS . Este es llamado *Global Navigation Satellite System* (GLONASS) y es la variante Rusa del GPS. Este sistema utiliza su propio *datum* llamado *Soviet Geodetic System 1990* (SGS90 o PZ90); también es global y difiere un poco del WGS84. Actualmente es posible encontrar receptores de georreferenciación satelital que permiten recibir señales pertenecientes a los dos sistemas; GLONASS y GPS. Esto hace posible tener una mayor exactitud, pudiendo llegar a precisiones de ¡ 1-2 mm !. [7]

Recuerda siempre convertir todos los datos con datums diferentes a uno solo. Para una revisión más profunda de estos temás se pueden consultar las siguientes referencias (Iliffe, 2003) y (Dana, 1995)

### 2.2.2. Mapas y proyecciones

Un mapa es una representación bidimensional de ciertos atributos de la superficie terrestre. Los mapas varían dependiendo de estos atributos, a pesar de estar representando el mismo lugar. Para crear un mapa debemos hacer coincidir un punto de la Tierra con un sólo punto en el plano, más en concreto, en la hoja de papel. A esta acción se le llama *proyección*. A pesar de que el conjunto de todas las proyecciones de la esfera al plano sea infinito no númerable, un infinito muy grande, [8] por la topología de la esfera no es posible proyectar todos los puntos de la esfera uno a uno, contínuamente, a algún punto del plano.[9] Esto quiere decir que no podremos representar toda o una parte de la Tierra en una superficie plana sin que exista algun tipo de deformación o sobrelapamiento.

A pesar de esto, la humanidad ha utilizado mapas desde hace muchos siglos. Los cartógrafos y matemáticos han construido muchas formas para proyectar todo o una parte del globo en el papel. Con el desarrollo de esta ciencia, la cartografía, se han diseñado proyecciones que preservan distancias, direcciones, áreas o formas. Así, los usuarios de los mapas pueden escoger qué tipo de proyección es útil para sus necesidades. Por ejemplo, un navegante utilizaría un mapa que preserve mejor las direcciones, en tanto que un grupo de ejidatarios utilizaría un mapa que preserve con exactitud las áreas.

Para poder entender mejor las características de los mapas es necesario definir algunos conceptos primero, que se presentan a continuación:

[7] La Unión Europea también planea establecer su propio sistema de navegación satelital para uso civil. Lleva el nombre de *Galileo*

[8] De hecho, la cardinalidad (tamaño) de este conjunto es igual a la cardinalidad del conjunto potencia (el conjunto de todos los posibles subconjuntos) de los números reales. i.e. infinitamente más grande que los números reales.

[9] Ver teorema de la invarianza del dominio.

**Conformalidad** Un mapa es *conformal* si para cualesquiera dos puntos en la superficie terrestre, el ángulo formado por estos dos puntos se preserva en la proyección, dicho de otra forma, preserva dirección.[10] Esto implica que los meridianos y paralelos intersecan en ángulos rectos, también las formas de áreas pequeñas se preservan.

**Áreas iguales** Una proyección es de *áreas iguales* si cualquier fragmento de algún área de la superficie terrestre es proporcional a la región correspondiente del mapa. El factor de proporcionalidad i.e. escala tiene que ser la misma en todo el mapa y en cualquier región. Ningún mapa puede ser conformal y de áreas iguales.

**Gratícula** La gratícula es el sistema de coordenadas esféricas basado en líneas de latitud y longitud. También se le llama *grid*.

**Círculos mayores** A los círculos producidos por la intersección de la esfera con un plano cuyo origen sea el origen de la esfera misma se les denomina *círculos mayores*. Para todo par de puntos $a$, $b$ en la esfera, existe un único *círculo mayor* $C$, tal que $a$ y $b$ están en $C$. A este segmento de arco, en la esfera, le corresponde la mínima distancia entre estos dos puntos,llamada *geodésica*.

**Tipos de proyecciones**
Con base en la superficie sobre la cual se puede proyectar la Tierra. Se acostrumbra clasificar a las proyecciones en tres tipos: cilíndricas, azimutales y cónicas, que se explican a continuación.

# Proyecciones cilíndricas

Este tipo de proyecciones se construyen proyectando la esfera alrededor de un cilindro (ver figura 2.6). El ejemplo típico de esta proyección es la de Mercator y fue presentada en 1569 (USGS Mapping Applications Center, 2000). Esta proyección consiste en hacer un cilindro tangente al Ecuador de la esfera aunque en ocasiones éste puede ser secante. Las proyecciones cilíndricas más usadas son:

[10] Matemáticamente; supongamos que la Tierra es una superficie diferenciable, denotémosla por $T$. Sea $V$ una vecindad abierta en $T$ tal que $p, q \in V$. Si $\Pi$ es una proyección conformal, entonces $< p \cdot q >= \|p\|\|q\|cos(\theta) = \mu\|\Pi(p)\|\|\Pi(q)\|cos(\theta) = \mu < \Pi(p) \cdot \Pi(q) >$ para algún $\mu \in \mathbb{R}$

## Proyección Mercator

Utilizada principalmente para la navegación en regiones ecuatoriales. Cada línea vértical es llamada *línea de rumbo*. Estas preservan dirección a todo lo largo. Generalmente éstas *líneas* no son las trayectorias más cortas. La figura 2.6 nos muestra esta proyección.

**Distorsiones**

Se incrementan a medida que la latitud crece, llegando al máximo en las regiones polares. Sin embargo es **conformal** en áreas pequeñas.

**Distancias**

Las distancias son verdaderas *sólo* en el Ecuador. Alredor de los 15 ° del Ecuador, al norte y al sur, las distancias son razonablemente cercanas a la realidad.

**Características**

El Ecuador y los demás paralelos son líneas rectas horizontales. La distancia entre ellos se incrementa a medida que se van acercando al polo. Los meridianos son rectas verticales y el ángulo de intersección de éstos con los parelos es recto. La distancia entre meridianos es constante. No aparecen los polos.


Figura 2.6: Proyección Mercator un ejemplo de proyección cilíndrica. Modificado y traducido del original con permiso del *U.S. Geological Survey.*

## Proyección Mercator Transversal

Es una proyección **cilíndrica**. En ésta, el cilindro no es tangente al Ecuador sino a algún meridiano en particular. Por eso el nombre *Transversal*, (figura 2.7). Como vimos en la proyección Mercator, ésta preserva distancias verdaderas sólo en la parte en la que es tangente el cilindro. En este caso será alrededor de un meridiano. Es por esta razón que se definen diferentes zonas de proyección dependiendo del lugar de estudio. Las coordenadas UTM, de las que hablaremos más adelante, están construidas bajo este modelo de proyección. Se usa principalmente en mapas de áreas largas que se extienden de norte a sur. La USGS [a] la utiliza para muchos de sus mapas cuadrangulares, con escalas desde 1:24,000 a 1:250,000.

- **Distorsiones**
  Después de la banda de los 15 ° de longitud, las distancias, formas y direcciones se distorsionan significativamente. El mapa es **conformal** es decir, en un area suficientemente pequeña, las formas y los ángulos son verdaderos, pero no en áreas grandes.

- **Distancias**
  Se incrementan significativamente a partir de la banda de los 15 ° alrededor del meridiano central.

- **Características**
  El Ecuador es una recta horizontal. Algunos paralelos se transforman en curvas cerradas encerrando al polo más cercano.

[a] Agencia de Investigación Geologica de los Estados Unidos. (*U.S.Geological Survey*) `web:http://usgs.gov`

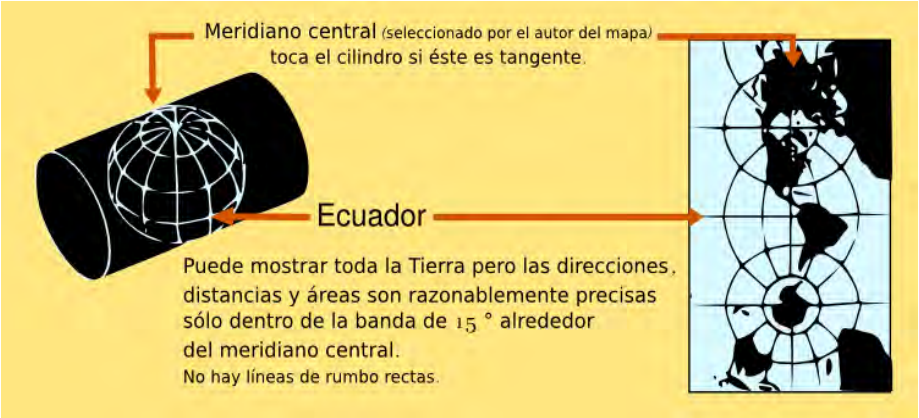



Figura 2.7: Proyección Mercator Transversal. Modificado y traducido del original con permiso del *U.S. Geological Survey*.

## Proyección Senoidal de Área Equivalente

Se utiliza frecuentemente para mostrar patrones de distribución. Estrictamente no es una proyección cilíndrica sino *pseudocilíndrica*. Se puede construir con un meridiano central o con varios, en este caso recibe el nombre de *forma interrumpida* (ver figura 2.8). Se usa en mapas de áreas muy grandes como Sur América y África. Tiene escalamiento lineal en áreas.

- **Distorsiones**
  Las formas se distorsionan más a medida que se alejan de los meridianos centrales y cerca de los polos.
- **Distancias**
  Sobre los paralelos y los meridianos centrales, las distancias son correctas.
- **Características**
  Este mapa tiene la propiedad de **equivalencia de áreas**, esto es que el área de cualquier parte del mapa es **proporcional** al area representada en la Tierra.

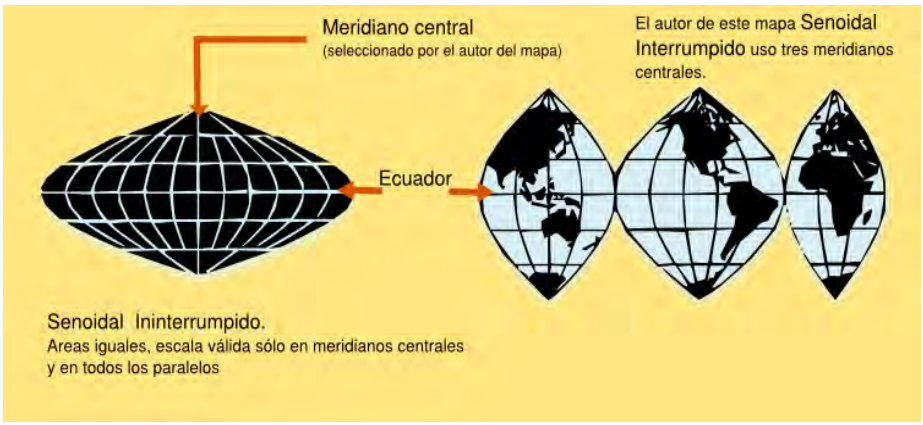



Figura 2.8: Proyección Senoidal de Área Equivalente. Modificado y traducido del original con permiso del *U.S. Geological Survey*.

**Otras proyecciones cilíndricas** Existen otras variantes a las proyecciones ya descritas. Algunas, como la *Mercator Oblicua* generaliza la superficie tangente a cualquier *círculo mayor* de la Tierra [11] Otras como la *Mercator Oblicua Espacial* está diseñada para que el *círculo mayor* tangente sea la órbita de algún satelite. Esta proyección se ha utilizado para imagenes satelitales como las *Landsat*, ya que no presentan distorsiones sobre la curva de barrido terrestre del satelite. Así se pueden obtener mapas continuos de imagenes satelitales. Este barrido se puede ver en tiempo real en la página:

[11] No sólo el Ecuador y los meridianos.

`http://earthnow.usgs.gov/earthnow_app.html`

Las propiedades de distorsiones y distancias son similares en todas las proyecciones cilíndricas.

# Proyecciones azimutales

Las proyecciones azimutales se construyen proyectando el elipsoide en un plano. Se selecciona un punto en la superficie terrestre y se hace el plano tangente a este punto. Existen muchas proyecciones de este tipo. Algunas son *conformales*, otras *preservan áreas*, otras *distancias* y otras únicamente perspectivas. Veamos algunas de las proyecciones azimutales más utilizadas en la actualidad.

## Proyección Gnomónica

Esta proyección transforma los *círculos mayores* en líneas rectas. Así, es fácil encontrar las *geodésicas* de dos puntos cualesquiera con una línea recta en el mapa. En la navegación se utiliza esta proyección junto con la de Mercator ya que mientras esta última indica el rumbo verdadero, la gnomónica indica las trayectorias más cortas. Se utiliza también para trabajos de propagación de sismos, ondas electromagnéticas y aeronáutica; es decir, todo lo que involucre distancias mínimas. Esta proyección es útil para mostrar rutas de migración. Muheim et al. (2003) hicieron un análisis de las rutas de migración de algunas aves encontrando una correlación espacial con un tipo de curvas llamadas loxodrómicas. Se denomina loxodrómica o loxodromia, a la curva que une dos puntos cualesquiera de la superficie terrestre y que corta todos los meridianos con el mismo ángulo. La loxodrómica, por tanto, es fácil de trazar manteniendo el mismo rumbo marcado por una brújula, en el caso de las aves, por su orientación.

- **Distorsiones**
  De forma y de área, se incrementan al alejarse del centro.
- **Distancias**
  La escala aumenta rápidamente en dirección opuesta al centro.
- **Características**
  En la proyección polar todos los meridianos son líneas rectas y se disponen radialmente. En la proyección ecuatorial son líneas rectas el ecuador y los meridianos y se disponen verticalmente. En la proyección oblicua son líneas rectas el ecuador y los meridianos. Ver figura 2.9

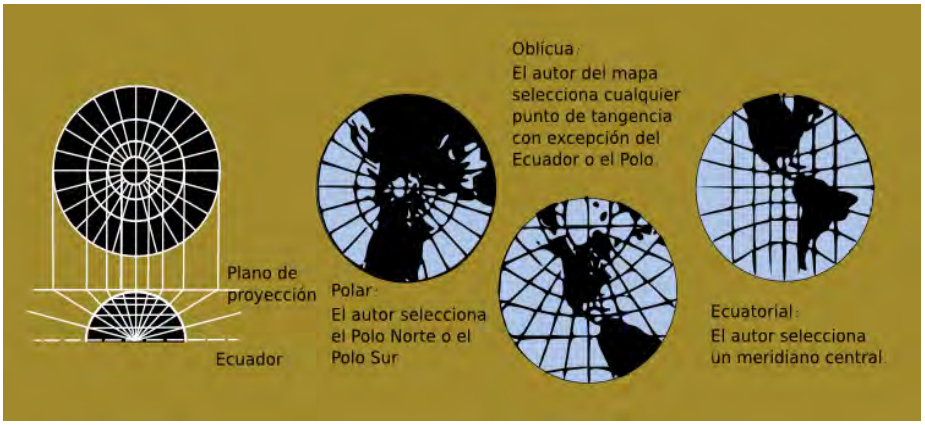



Figura 2.9: Proyección Gnomónica es tres aspectos diferentes; polar, oblicua y ecuatorial. Modificado y traducido del original con permiso del *U.S. Geological Survey*.

## Proyección Azimutal Equidistante

Se usa en mapas de gran escala y en trabajos de sismología y radio. El aspecto oblícuo es usado en mapas continentales, el aspecto polar es usado para mapas mundiales. Los mapas polares y el emblema de la Organización de las Naciones Unidas (ONU) tienen esta proyección (ver figura 2.10).

- **Distorsiones**
  Se incrementan al alejarse del centro de proyección.
- **Distancias y direcciones**
  Son correctas **sólo** desde el centro de proyección a cualquier punto del mapa. Las distancias son verdaderas sólo para puntos que sean colineales con el centro de proyección.
- **Características**
  Toda línea recta que pase por el centro es un *círculo mayor*.

**Otras proyecciones azimutales:** Hay una gran variedad de proyecciones azimutales; cada una sirve para un proposito específico. La proyección *Azimutal de área equivalente de Lambert* conserva el tamaño de áreas proporcional a la realidad en cualquier región del mundo. Por otro lado hay otras como la estereográfica que se utiliza en la navegación polar y en geofísica para resolver problemas relacionados con la geometría esférica. Para una descripción más a fondo de estas proyecciones se puede revisar la referencia (USGS Mapping Applications Center, 2000) y para ver detalladamente las ecuaciones y el tratamiento geométrico diferencial de éstas proyecciones, las notas de Madrid et al. (1998) explican detalladamente esto.


Figura 2.10: Proyección Azimutal Equidistante. Modificado y traducido del original con permiso del *U.S. Geological Survey*.

# Proyecciones cónicas

La proyección cónica se obtiene proyectando la superficie del esferoide en una superficie cónica tangente o secante. Se toma como vértice el punto colineal al eje que une los dos polos, es decir el eje de rotación. La primera representacion de este tipo fue hecha por Ptolomeo aproximadamente en el año 150 A.D (Longley et al., 2005). Entre las proyecciones cónicas más usadas están las *conformales*, las *equidistantes* y las de *aŕeas iguales*.

## Proyección cónica de *Área Equivalente de Alberts*

Para su construcción se deben escoger dos paralelos estándar. Como se ve en la figura 2.11 el cono, en este caso, no es tangente sino secante. Dos mapas se pueden *pegar* si: *i)* tienen la misma escala y *ii)* tienen los mismos paralelos estándar. Es útil para regiones que se extienden de este a oeste y que requieren de una representación de áreas precisa. Se utiliza mucho en *mapas temáticos*.

- **Distorsiones**
  La escala se preserva sólo a lo largo de los paralelos estándar.
- **Distancias**
  Son verdaderas en ambos paralelos estándar.
- **Direcciones**
  Razonablemente precisas en regiones limitadas.
- **Características**
  Todas las áreas en el mapa son proporcionales a las mismas áreas en la Tierra.

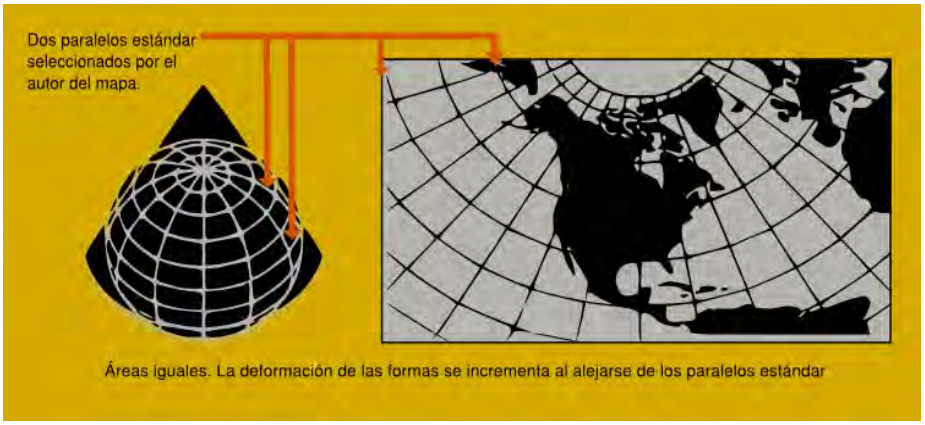



Figura 2.11: Proyección de Área Equivalente de Albers. Modificado y traducido del original con permiso del *U.S. Geological Survey*.

## Proyección cónica *Conformal de Lambert* (CCL)

Similar a la de Albert de área equivalente. Sin embargo la separación de la *gratícula* es distinta. Se deben escoger dos paralelos estándar (ver figura 2.12). El cono es secante a la Tierra. También es útil para regiones que se extienden de este a oeste. Se utiliza mucho en *mapas temáticos*.

- **Distorsiones**
  De áreas y formas mínimas cerca de los paralelos estándar, pero aumentan al alejarse. Las *formas*, en escalas pequeñas, son esencialmente verdaderas.
- **Distancias**
  Verdaderas a lo largo de los paralelos estándar y razonablemente precisas en regiones limitadas.
- **Direcciones**
  Precisas.
- **Características**
  Actualmente es la proyección más utilizada en Norte América, en especial en México.

**Consideraciones para México.** Esta proyección, junto con la UTM (zonas 11 – 17) son las más utilizadas en México. La *CCL* se usa para regiones grandes que abarcan todo o gran parte del territorio nacional, la UTM para estudios regionales. Los parámetros oficiales para la proyección CCL en grados decimales son:

- Latitud del primer paralelo estándar: 17.5 N
- Latitud del segundo paralelo estándar: 29.5 N
- Latitud del origen de la proyección: 12.0 N
- Longitud del meridiano central: -102.0 (102.0 W)
- Este del origen [a]: 2,500,000.00
- Norte del origen [b]: 0.00 .

[a] También llamado *Este falso*
[b] También llamado *Norte falso*

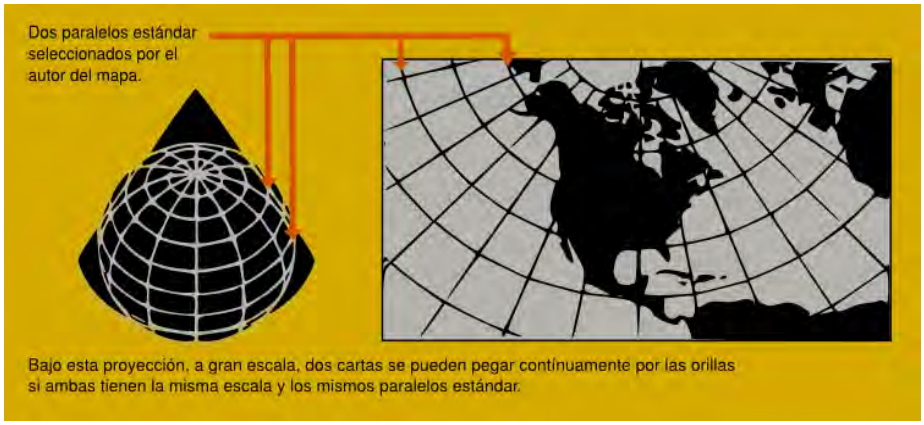



Figura 2.12: Proyección Conformal de Lambert. Modificado y traducido del original con permiso del *U.S. Geological Survey*.

## Proyección cónica equidistante

Ideal para representar regiones en latitudes medias. De manera similar a las otras se deben escoger dos paralelos estándar, por eso el cono es secante. (figura 2.13)

- **Distorsiones**
  Se incrementan al alejarse de los paralelos estándar.
- **Distancias**
  Verdaderas a lo largo de todos los meridianos y sobre los paralelos estándar.
- **Direcciones, formas y áreas**
  Razonablemente precisas.
- **Características**
  El mapa no es conformal ni preserva áreas pero es intermedio entre estos dos, i.e. entre la proyección Conformal de Lambert y la proyección de Área Equivalente de Albers.

Al trabajar con algún *SIG* debemos escoger la proyección que mejor preserve las propiedades con las cuales vamos a trabajar y que queramos preservar al elaborar el mapa resultante en el análisis. Por ejemplo, si el análisis involucra cambios en área, una proyección que mejor preserve ésta sería la más conveniente para usar y para visualizar los resultados. La buena noticia es que la mayoría de los paquetes de *SIG* ya traen incorporadas herramientas para convertir mapas de una proyección a otra. Estas herramientas también están disponibles en el mundo del *Software Libre* y aprenderemos a utilizarlas más adelante. Por ahora es importante tener en cuenta que al realizar una conversión con cualquier herramienta se producen errores de aproximación, mismos que pueden acumularse conforme convertimos multiples veces un mapa. Eventualmente podremos tener un error suficientemente grande como para tener un impacto negativo en el análisis.


Figura 2.13: Proyección Equidistante, *Cónica Simple* Modificado y traducido del original con permiso del *U.S. Geological Survey*.

Para realizar cualquier análisis o actividad en un SIG nos tenemos que fijar en **¡ tres cosas importantes !**

- El tipo de **proyección**.
- El tipo de **datum**.
- Las **unidades** en las que están los datos (metros, millas, kilómetros, ángulos, etc.).

Recuerda: todos tus datos deben tener estos tres atributos en común.

**El formato *+proj* y el EPSG**

Hemos revisado distintas proyecciones y datums; también vimos que estos son independientes entre sí. Un mismo datum puede tener diferentes parámetros elipsoidales (datum) y también puede estar proyectado de varias formas. El número de combinaciones posibles es grandísimo. Prácticamente todos los SIG tienen la capacidad de analizar distintos datums con distintas proyecciones, e incluso algunos pueden hacer conversiones entre una proyección y otra. Este manual carecería de sentido si no existieran herramientas libres capaces de procesar y transformar **cualquier** tipo de proyección a cualquier otra, incluidos los respectivos datums. Las herramientas más usadas para estos propósitos son las bibliotecas *Proj4* (Evenden et al.) y *GDAL/OGR* (Warmerdam, 1998-2010). Muchas aplicaciones SIG libres, sean de interfaz gráfica o de línea de comandos, utilizan estas bibliotecas internamente, *under the hood*. En el capítulo tres se describe cómo usar e instalar estas herramientas. Por ahora basta mencionar que éstas traen consigo algunas aplicaciones para transformar proyecciones, convertir datums y calcular geodésicas. Todos estos comandos utilizan un tipo especial de sintaxis llamado: *formato +proj*, el cual define la proyección, el elipsoide , el datum, las unidades y los paralelos o meridianos estándar, según sea la proyección deseada de cualquier mapa. Estos parámetros se definen en un archivo de texto común llamado generalmente WKT *Well Known Text*. El archivo siempre empieza con el signo + y la sintaxis es la siguiente:

```
+proj=[proyección] +zone=[sólo en UTM] +ellp=[elipsoide]
            +datum=[ ] +units=[m,km,mi,etc.]
```

Si se tiene un equipo Linux con la biblioteca ProJ.4 instalada (esta instalación se verá más adelante) podemos encontrar un listado de todas las proyecciones soportadas por este formato en: `/usr/share/proj/epsg`. [12]

El conjunto de parámetros geodéticos puede ser muy largo y difícil de recordar. Para simplificar el trabajo el *Grupo de Investigación Petrolera Europea* (EPSG por sus siglas en inglés) ha estandarizado un listado de códigos para cada sistema de coordenadas, unidades y demás parámetros acerca de todo tipo de proyección y datum. Este código se ha convertido en un estándar internacional y prácticamente todas las aplicaciones SIG pueden utilizar el código EPSG como entrada para hacer transformaciones. ProJ.4 no es la excepción, el listado con las equivalencias +proj con EPSG está en el mismo archivo antes mencionado. Se puede encontrar más información en la página del proyecto `EPSG`. [13].

Cabe señalar que en ocasiones, sobre todo en imágenes, estas transformaciones tienen pérdida de información, por lo que lo mejor es transformar los datos sólo una vez y no varias. Está pérdida de información es inherente a todo algoritmo de transformación; sea libre o no.

### 2.2.3. Sistemas de coordenadas geográficas y el sistema UTM

Cuando proyectamos una región de la Tierra en un mapa, la localización de puntos cambia. Es decir, ya no podemos asignarle valores angulares a la la latitud y a la longitud como se hacía en el modelo esferoidal. Esto es necesario si queremos hacer mediciones de distancias y áreas. En general lo que hacemos al proyectar la superficie a un mapa es un cambio de coordenadas esféricas o elipsoidales a coordenadas rectangulares o cartesianas, es decir al sistema de coordenadas $(x, y)$ que conocemos desde la secundaria. Como hemos visto, hay proyecciones que tienen diferentes características, lo que implica que la forma en la que corresponden las coordenadas $(x, y)$ con las de *(longitud, latitud)* van a ser diferentes. Podemos decir que a cada proyección (con su respectivo datum y elipsoide) le corrresponde un sistema de coordenas diferentes. La gratícula o gradilla que tienen los mapas es una indicación de la longitud, en lineas verticales, y la latitud, en lineas horizontales. En general esta *cuadrícula* no tiene que ser regular, es decir el tamaño de los cuadros varía según su localización.

Uno de los sistemas de coordenadas más utilizados es el sistema *Universal de Mercator Transverso*, conocido por sus siglas como **UTM**. Ya vimos que *Mercator Transversa* es un tipo de proyección cilíndrica que, entre las bandas de los 15 ° alrededor del *meridiano central*, la proyección es conformal es decir, preserva formas, ángulos y distancias. Es fácil ver que si trasladamos el *meridiano central* a cualquiera que nos interese estas propiedades también se preservan en bandas alrededor de los 15 °, es decir con distancias que varían de 1670 km en el Ecuador a 1181 km a los 45 ° de latitud. Entre más nos acerquemos a los polos será mayor la reducción de estas distancias.

[12] Con el comando `more`, en una terminal, de la siguiente manera: `usuario@maquina:~ $more /usr/share/proj/epsg`

[13] `http://www.epsg.org`

Afortunadamente no tenemos que preocuparnos por construir una proyección UTM para cada región de interés, ya que el Sistema UTM divide a la Tierra en 60 zonas verticales distintas (ver figura 2.14), cada una correspondiente a un *meridiano central*. Esta división se hace desde los 84° latitud norte hasta los 80° latitud sur (DeMers, 2009), cada una con 6° de longitud de ancho; esto es alrededor de 666 km en el Ecuador a 474 km a los 45° de latitud. La figura 2.15 muestra como se reducien las distancias conforme la latitud se aleja del Ecuador.

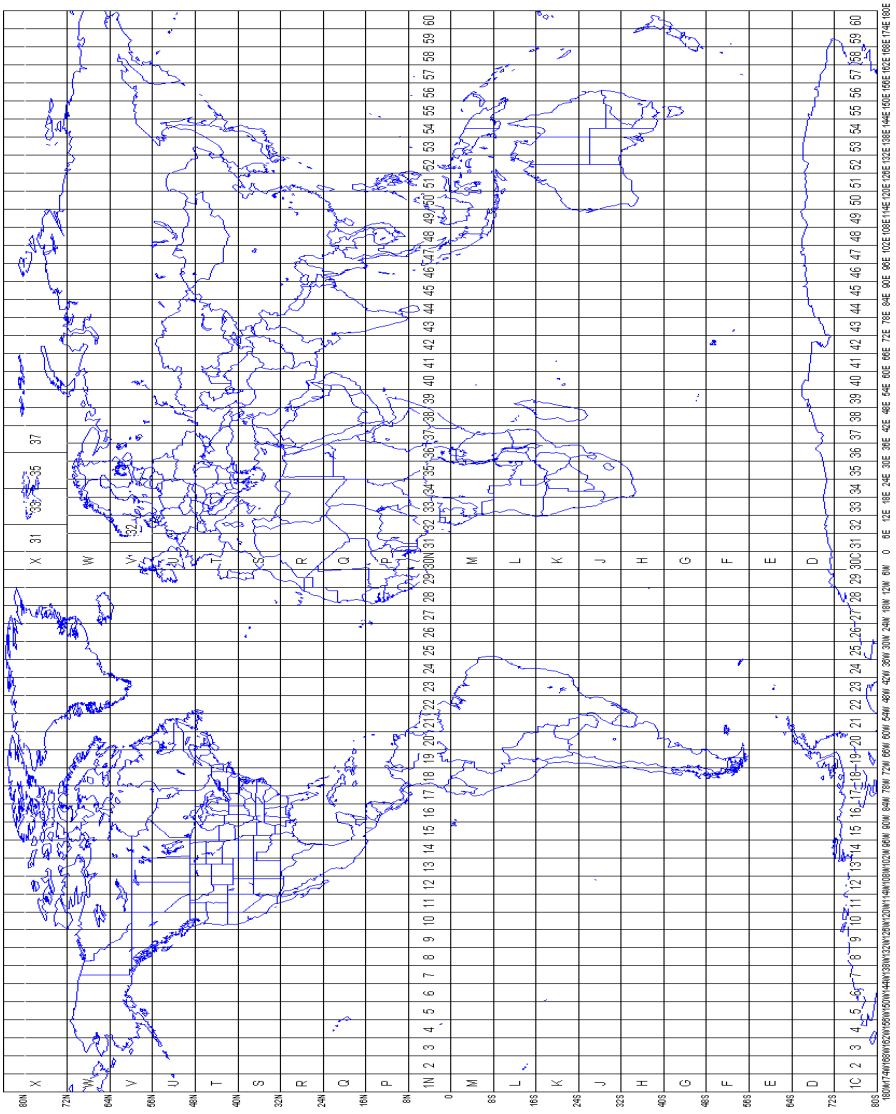


Figura 2.14: División de la superficie terrestre en 60 zonas **UTM**.

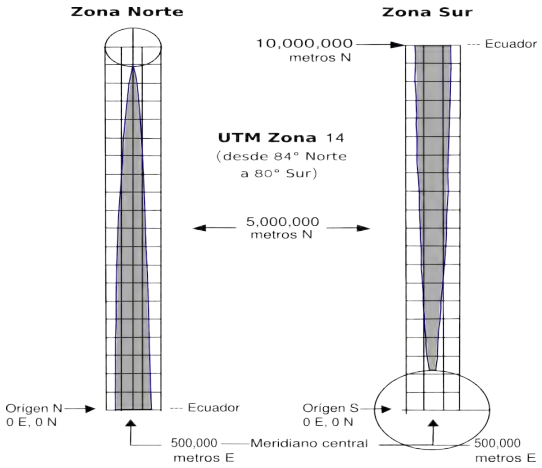



Figura 2.15: Disminución de las áreas (regiones sombreadas) en el sistema UTM conforme se alejan del Ecuador. El meridiano central es el correspondiente a la zona 14 norte, 99° oeste. Las unidades están en metros; al norte **N** (*eje y*) y al este **E** (*eje x*). La división está de los 84° latitud norte a los 80 ° latitud sur. Basado en los diagramas de (Dana, 1995) y (DeMers, 2009).

**Zonas UTM para México** En el caso de México se utiliza oficialmente esta proyección para estudios regionales . México, por su extensión, está entre las zonas 11 y 17, por lo que dependiendo del sitio de estudio, deberemos trabajar con alguna de estas zonas. La figura 2.16 muestra como está particionado el territorio mexicano en estas seis zonas. Si la región de interés está justo en la frontera de dos zonas, se debe escoger aquella en la que su ubique la mayor parte de la región esté, o sí hay imprecisión se puede optar por la que mejor convenga.

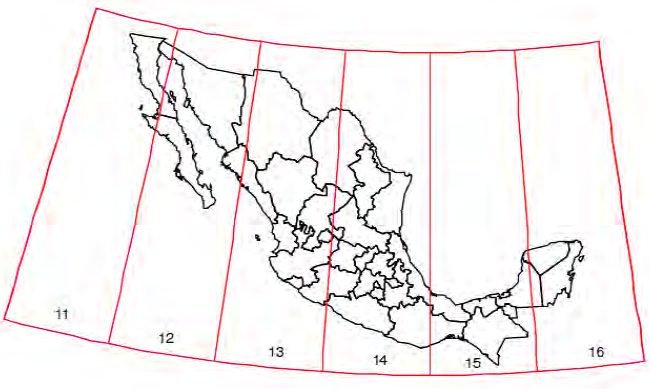



Figura 2.16: Particionado oficial de México en las diferentes zonas UTM . Tomado de SEMARNAT en `http://inforteca.semarnat.gob.mx`

## 2.3. Representaciones de los fenómenos en la superficie terrestre

La información en una base de datos georeferenciada es, esencialmente, una representación digital y simplificada de ciertos datos o mediciones en alguna región de la Tierra. Estos datos pueden estar organizados bajo diferentes criterios que en el lenguaje de los sistemas de información geográfica se llaman *capas*. Cada capa representa diferentes conjuntos de datos o relaciones de éstos. Estas capas deben estar representando una región y deben tener un mismo sistema de coordenadas. Así podemos aplicar diferentes *acciones* y con esto crear nueva información acerca de un problema en particular que involucre al espacio.

Cada capa debe estar almacenada bajo un modelo de datos apropiado que depende de la fuente de la infomación y su uso potencial.

### 2.3.1. Modelos de datos georreferenciados

Llamamos datos georeferenciados a la información que tiene asignadas coordenadas espaciales y por tanto se puede relacionar con puntos en la superficie terrestre. Este tipo de datos poseen dos componentes principales: la componente *espacial* (gráfica o geométrica), que describe la localidad o distribución de fenómenos geográficos y la componente de *atributos* usada para describir sus propiedades. La componente espacial puede representarse usando uno de los siguientes tipos datos:

- Representación de *campo*. Aquí cada elemento de área (pixel) en el espacio tiene asignado un valor. Se puede interpretar como una matriz (n,m) donde cada entrada representa una unidad de área (pixel). Esta implementación de dato se llama *Modelo de dato ráster* o simplemente *ráster*. Para este tipo de datos se deben tener en cuenta dos parámetros. *i)* la *región* (el tamaño de la matriz) y *ii)* la *resolución*, que representa la unidad de área mínima que representa cada pixel. Dependiendo de si los datos están proyectados o no, este parámetro se puede definir en unidades cartesianas ($m^2$, $km^2$, ha, acre) o en grados sexagesimales (si no están proyectados).
- Representación con *objetos geométricos*. Aquí las características geográficas se definen con puntos, líneas y áreas, todos definidos por sus coordenadas. Este tipo de modelo recibe el nombre de *Modelo de dato vectorial*.

La característica geográfica de las representaciones puede cambiar conforme cambiemos la escala. Por ejemplo, un camino puede representarse como una línea a una escala pequeña [14]. Al aumentar la escala, el camino se puede representar como un objeto con otra dimensión, que sería el ancho del camino.

[14] Recordemos que una escala es una proporción entre las unidades de la imagen y el tamaño del sitio que se representa. Cuando se menciona la palabra escala pequeña, esta se refiere a una proporción pequeña. Esto es que el denominador es muy grande; ejemplo: 1:1,000,000. Por otro lado una escala grande es cuando el denominador no es tan grande; ejemplo: 1:100. La resolución, entonces, es proporcional al tamaño de la escala.

De igual forma un poblado se puede representar como un punto o como un área dependiendo de la escala.

Para un manejo efectivo del SIG es importante conocer las funciones y propiedades básicas de estos modelos.

**Modelo de datos ráster**

Un ráster se puede representar de dos formas: como una matriz o como una cuadrícula. Éste puede representar *campos escalares* continuos, como elevación, humedad, tempertura, densidad de clorofila, etc. En este caso los valores asignados están determinados por los vértices de la cuadrícula. Este tipo de dato a veces es llamado *látiz*. Si los valores son asignados en las celdas de la cuadrícula, entonces ésta representa una imagen y cada entrada de la matríz es un valor de color. Por ejemplo, puede ser que cada entrada corresponda con un pixel. Generalmente es el caso, aunque no lo es siempre. Las imágenes pueden ser de cualquier tipo: satelite, mapa digitalizado, ortofotografía, mapa topográfico, precipitación, humedad, viento, etcétera. El valor de las celdas puede representar valores distintos al color, por ejemplo categorias (números enteros); de esta forma se le pueden asociar uno o más atributos que describan otras características del lugar, como por ejemplo textura, pH, cobertura, entre otros. El área representada por cada celda es calculada por el largo del lado llamado *resolución*. Este parámetro controla el nivel de detalle espacial dado por el dato ráster (ver figura 2.17). En general se utilizan *rásters* de dos dimensiones (plano). En este caso la unidad de área se llama *pixel*. Existen también modelos *ráster* en tres dimensiones, que guardan información acerca de volúmenes; en este caso la unidad mínima de trabajo se llama *voxel*. Un ejemplo de este tipo de dato son las resonancias magnéticas. La máquina va apilando rásters bidimensionales de forma tal que se consigue al final un "cubo" de datos que representa las lecturas hechas a un objeto con un número grande de planos, que globalmente representan un volumen. Existen también *rásters* n-dimensionales utilizados para datos con dinámicas espacio-temporales o multiespectrales. [15]

Existe una variedad muy grande de formatos para este tipo de datos entre los más frecuentes se encuentran:

- JPEG2000 (.jp2, .j2k)
- Modelos de elevación digital (.dem)
- Erdas Imagine (.img)
- GeoTIFF o TIFF con archivo *world* (.tif, tfw)

Como veremos más adelante, el formato *ráster* es utilizado frecuentemente para estudios de sistemas biológicos. como vegetación, diversidad, flujo de agua, etc. En general es el objeto de estudio de la percepción remota. El trabajar con modelos raster tiene ventajas y desventajas. Esto no debe ser una limitación para nosotros cuando estemos trabajando con un SIG *multipropósito*, i.e. capaz de analizar datos ráster y vectoriales; sin embargo es bueno saber las capacidades y limitaciones de los tipos de datos que utilizan los SIG.

[15] Ver HDF format: `http://www.hdfgroup.org/`

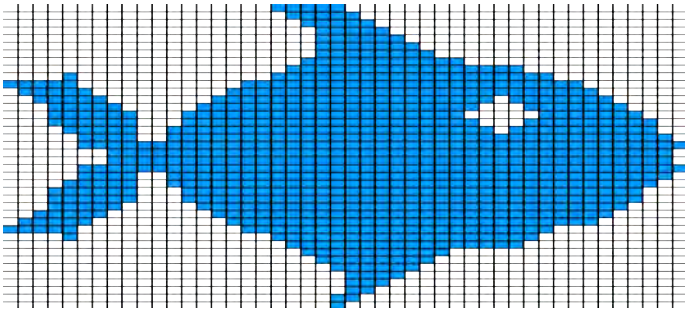


Figura 2.17: Ejemplo de una imagen ráster monocromática. El tamaño de la celda representa la resolución.

PROS
- **Simplicidad en el manejo de datos continuos** por sus algoritmos de análisis y modelación, incluyendo álgebra de mapas.

CONTRAS
- No es eficiente para análisis con datos discretos, como distancias óptimas, análisis de redes y otros tipos de datos dependientes directamente de primitivas geométricas como puntos, líneas y polígonos. e.g. fronteras estatales, ejidales, áreas de polígonos, etc.
- Suceptible a errores y distorsiones al transformarse y aumentar la escala.

**Modelo de datos vectoriales**

Este modelo fue diseñado para representar objetos con muchos atributos y que se pueden modelar con áreas, líneas y puntos (*primitivas geométricas*) . Estas primitivas, dependiendo de la escala pueden representar objetos de una, dos o tres dimensiones. Estos objetos están hechos a partir de puntos por tanto no se deforman por cambios en la escala, rotaciones ni traslaciones i.e. invariantes bajo transformaciones afines. El modelo de dato vectorial esta basado en el *arc-node*, es decir de líneas que no se intersectan llamadas *arcos*. Éstos son almacenados como una lista de puntos de la forma *(x,y) o (x,y,z)*. Los dos extremos del arco se llaman *nodos* y los elementos de la lista se llaman *vértices*. Dos puntos consecutivos en la lista se llaman *segmentos de arco*. Los arcos forman primitivas de nivel mayor como *líneas* (e.g. caminos, carreteras) o *áreas*. Los arcos que definen un área (polígono) se llaman en conjunto *frontera*. A cada característica del mapa le es asignada una categoría numérica, que relaciona el dato geométrico con un dato descriptivo de algún atributo. Los datos vectoriales generalmente tienen incorporada una base de datos. Cada objeto de un mapa vectorial puede tener un número arbitrario de atributos, a diferencia del modelo ráster. Por ejemplo, si se tiene el mapa vectorial de las estaciones del metro, a cada vértice se le puede asociar una categoría que corresponde a la línea a la que pertenece (verde, rosa, azul, etc.), otra del promedio de viajeros diarios, otra de velocidades y tiempos, etcétera. La figura 2.18 muestra un mapa de este tipo de modelo. Es común encontrar distintos tipos de formatos. Los más comunes son:

- Shapefiles (.shp)

- datos de GPS (.gpx)
- ArcInfo (.e00)
- MapInfo(.tab, .mid, .mif)

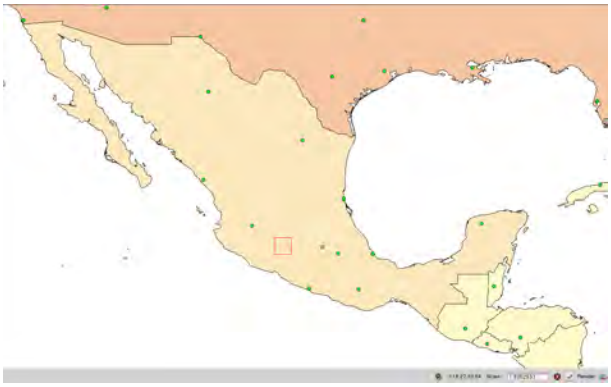


Figura 2.18: Ejemplo de un mapa vectorial de la Republica Mexicana. En verde los puntos correspondientes a algunas localidades, todo el territorio nacional es un polígono.

Usos y limitaciones de este modelo:

PROS
- **Son muy eficientes con datos discretos** que pueden ser descritos como: puntos, líneas o polígonos con geometría simple, como caminos, redes, fronteras de propiedades privadas, planos, etc. Incluso se pueden representar datos continuos utilizando isolíneas, isosuperficies, gradientes y *grids*.

CONTRAS
- Para analizar y modelar representaciones más complejas con variables continuas el tiempo de cómputo de los algoritmos para este tipo de datos puede ser muy grande, infinitamente grande.

Estos dos modelos pueden ser cargados simultáneamente como *capas* en un SIG *multipropósito*, como los libres que veremos más adelante. De esta forma se pueden utilizar las ventajas de ambos, ya que sus *pros y contras* se complementan entre si.

#### Atributos y bases de datos

Los atributos son la otra componente de los datos georeferenciados. Es decir, sabemos que un objeto está en determinada localidad pero no sabemos que objeto es ¿Cómo distinguirlo entre otros objetos? Los atributos son datos descriptivos que proveen información asociada a los datos geométricos. Éstos pueden ser administrados en bases de datos externas o internas al SIG e incluso pueden estar incrustadas (*embeded*) en los mismos datos vectoriales. Los SIG usan las correspondientes coordenadas o números de identificación para relacionar los atributos con los datos geométricos. Algunas bases de datos

como PostGIS [16] basada en PostgreSQL permiten al usuario almacenar datos geométricos.

#### Transformaciones entre modelos de datos

Un mismo fenómeno puede ser representado por diferentes modelos de datos. Los SIG usualmente incluyen herramientas para transformaciones entre distintos modelos (vectorial, ráster). Por ejemplo, la elevación puede ser medida con datos vectoriales y luego interpolada a un mapa *ráster* que después se utiliza para obtener curvas de nivel de tipo vectorial. Las transformaciones entre diferentes datos no son perfectas, se puede perder información, tener deformaciones o desplazamientos por la transformación. Más adelante veremos también que en determinadas circunstancias deberemos transformar un modelo en él mismo, pero bajo otra proyección u otro formato.

### 2.3.2. SIGs de escritorio

El mundo de los SIG es muy diverso. Existen diferentes paquetes informáticos que ofrecen distintas herramientas de acuerdo a nuestras necesidades y presupuestos. Aún así podemos separlos en dos grandes grupos: *a)* los SIG basados en el esquema cliente-servidor, como podrían ser: Google Earth, Guía Roji, Google Maps y su variante libre OpenStreetMap; [17] *b)* el otro grupo de SIG son los llamados *SIG de escritorio* (e.g. GRASS, QGIS, ArcGIS). Los primeros recopilan la información de la red, es decir, desde el el servidor y mediante una interfaz web nos muestra los datos. Estos sistemas usualmente responden a consultas básicas y generalmente no se pueden crear datos nuevos en ellos. Los SIG de escritorio, por el contrario, son plataformas mucho más versátiles; podemos crear, modificar y procesar datos geoespaciales con total libertad. Este manual tiene como propósito aprender a utilizar los SIG de escritorio.

## 2.4. Perfiles de usuario

Cada persona puede utilizar algún tipo de SIG para realizar alguna tarea particular. Estas pueden variar desde una sencilla visualización de datos georeferenciados en campo (e.g. colectas) hasta análisis complicados de terrenos, usos de suelo, modelados hidrológicos, etcétera. Dada esta *diversidad* de motivaciones para iniciarse en el mundo SIG, agruparemos a los usuarios en tres categorías:

[16] PostGIS , `http://postgis.refractions.net`

[17] OpenStreetMap es un proyecto colaborativo para crear mapas libres y editables. Los mapas se crean utilizando información geográfica capturada con dispositivos GPS móviles, ortofotografías y otras fuentes libres. sitio:`http://wiki.openstreetmap.org/wiki/Main_Page`

| | |
|---|---|
| **Casual** | Aquél que busca únicamente visualizar sus datos y exportarlos como imágenes. |
| **Intermedio** | Aquél que quiere hacer algo de análisis espacial, además de visualizar. |
| **Avanzado** | Aquél que busca hacer análisis complejos como geoestadística, simulaciones y crear aplicaciones particulares para resolver un problema en concreto. |

A lo largo de este libro abarcaremos estas categorias, cada una contenida en la siguiente. Para el usuario casual, las herramientas básicas de un SIG amigable como *uDig* o *QGIS* pueden satisfacer sus necesidades. El intermedio tendrá que invertir un poco más de tiempo en aprender algo de bases de datos y utilizar otras funciones más avanzadas. Talvez con *QGIS* tenga suficiente para trabajar, pero necesitará bibliotecas especiales como GDAL/OGR y una que otra aplicación más. Su curiosidad lo puede llevar más lejos y comenzar con GRASS.

El mundo de los SIG de código abierto (OSGIS) es muy grande se puede consultar:

- http://www.osgeo.org
- http://www.free.gis.org
- http://opensourcegis.org
- http://maptools.org

He escogido los proyectos más maduros y robustos para que podamos construir nuestra *caja de herramientas geomáticas* y podamos hacer las mismas cosas o más que con las costosas herramientas SIG tradicionales de licencia privada. Esto permitirá satisfacer las necesidades de los usuarios casuales e intermedios y guiará el camino de los más avanzados.

# Capítulo 3

# Herramientas geomáticas de código abierto -OSGIS-

En este capítulo revisaremos dos de las principales aplicaciones SIG libres: GRASS y QGIS, pues son las más usadas hoy en día. Veremos las capacidades de cada uno, así como algunos ejemplos, formas de instalación e integración con otros programas para extender más sus capacidades.

El capítulo está estructurado en dos partes: herramientas *gráficas* y herramientas en *línea de comandos*. En la primera se hace una descripción general de las aplicaciones antes mencionadas. Se explican sus respectivas formas de instalación, que aunque sencillas, ciertamente son diferentes a las instalaciones en máquinas con Windows.

La segunda parte son las herramientas de línea de comandos, que permiten optimizar el análisis mediante instrucciones sencillas llamadas *scripts*. Veremos como hacernos de una caja de herramientas (*GeoKit* ) con todos los programas necesarios para poder realizar prácticamente cualquier tarea que involucre visualización, análisis, conversión e incluso geoestadística y simulación de datos georeferenciados, utilizando 99 % Software Libre. La parte del procesamiento espacial lo dejaremos para el último capítulo.

## 3.1. Principales SIG Libres

Existe una infinidad de SIG Libres, cada uno diseñado con un propósito específico y bajo un lenguaje de programación específico también. Para ver todos estos programas se puede visitar el sitio:

`http://freegis.org/database.`

Para los fines prácticos de este manual sólo vamos a trabajar con los SIG: *Quantum GIS (* **QGIS***)* y *Geographic Resources Analysis Suport System* **GRASS**. Estos son los más desarrollados y dependiendo de sus versiones también son los más estables. El orden es directamente proporcional a su capacidad de análisis e inversamente al tiempo requerido para aprender a usar sus herramientas básicas (*curva de aprendizaje*). Recordando los *perfiles* de usuario definidos en el capítulo anterior, podemos clasificar a QGIS en la categoría

*Casual* a *Intermedio* y a GRASS en la categoría *Avanzada*. Esto no quiere decir que aprender a usar el paquete más desarrollado y poderoso, es decir GRASS, sea muy complicado. Si bien no es tan amigable como el otro, posee una capacidad de análisis impresionante. No nos debe preocupar aprender dos formas distintas de trabajar pues existe una interfaz (*plugin*) de QGIS que nos permitirá trabajar en GRASS sin salirnos del amigable QGIS. Hay muchas tareas que se pueden hacer con SIG sencillos. Es justo darle a todos los lectores la opción que mejor convenga a sus necesidades. Una vez finalizado esta introducción a las herramientas, el paso siguiente será enfrentarnos a un problema de real.

### 3.1.1. Quantum GIS

*Quantum GIS* (QGIS) es un Sistema de Información Geografía de Código Abierto (*OSGIS*) por su acrónimo ingles, que está liberado bajo la licencia pública de GNU GPL . Esta aplicación puede correr en Linux, MacOSX o Windows y puede soportar capas de datos ráster, vectoriales y bases de datos. El proyecto de QGIS empezó en mayo del 2002 por iniciativa de *Gary Sherman* (Sherman, 2008), que buscaba un visualizador de capas SIG para Linux que fuera rápido y soportara una gran variedad de formatos de datos. La primera versión sólo podía leer bases de datos y se estrenó el 19 de Julio de 2002 (Quantum GIS Development Team, 2009). QGIS se ha desarrollado por la comunidad libre desde entonces, convirtiéndose en uno de los SIG libres más usados y fáciles de manejar. Entre sus principales características están (Quantum GIS Development Team, 2009):

- Visualización y sobrelapamiento de capas en distintos formatos, sin necesidad de exportarlos a un formato particular.
- Conversión en tiempo real (*al vuelo*) de proyecciones y *datums*.
- Entre los formatos soportados incluye :
  - Modelos vectoriales como: *ESRI Shapefile, MapInfo, SDTS y GML.* [1]
  - Modelos ráster como: *Landsat, DEM y Erdas* [2].
  - Bases de datos como: *PosgreSQL* y su variante para SIG *PostGIS*.
  - Servicios web de datos espaciales como: *WMS y WFS*.
- Digitalización, creación y exportación de datos espaciales usando:
  - El *plugin* georreferenciador (*Georreferencer*), que proporciona una interfaz sencilla para referenciar objetos como puntos, líneas, polígonos e incluso imágenes.
  - Herramientas para descargar, actualizar e introducir datos en dispositivos GPS mediante el módulo *GPSBabel*.
  - Herramientas para digitalizar y crear datos vectoriales tipo *Shapefile* o tipo GRASS.

[1] En general todos los de la biblioteca OGR.
[2] En general, los de la biblioteca GDAL

- Herramientas de análisis espaciales para datos tipo ráster con el *plugin* de GRASS o con las herramientas brindadas por la biblioteca *FWTools* para datos vectoriales.

  Entre los modulos de análisis disponibles estan:

  - Análisis de terreno
  - Álgebra de mapas
  - Modelos hidrológicos
  - Análisis de redes

- Se puede adaptar a las necesidades específicas de los usuarios por medio de *scripts* y de un API (ver glosario) sencillo basado en Python.

**Ejemplos sencillos**

QGIS se instala con relativa facilidad siguiendo los pasos mencionados más adelante (pág. 78, sección 3.2.2). Por ahora veremos algunos ejemplos para mostrar cómo se puede manipular información georreferenciada.

**Cargar mapas vectoriales** Una fuente importante de información relativa al territorio nacional es el servidor SIG de la *Comisión Nacional para el Conocimiento y Uso de la Biodiversidad* ( CONABIO ). Este servicio se encuentra en el sitio:

`http://www.conabio.gob.mx/informacion/gis/`

que ofrece información georreferenciada relacionada con la topografía, hidrología, uso del suelo, biodiversidad, edafología, división política, entre otras, del territorio mexicano en formato vectorial tipo *ESRI Shapefile* y por tanto compatible con QGIS y GRASS.

**Mapa de densidad poblacional en formato vectorial tipo *ESRI Shapefile*.** Para propósitos del ejemplo utilizaremos el mapa llamado *Densidad de población por entidad federativa, 2000* publicado por CONABIO en 2006 en el sitio mencionado arriba. Dicho mapa se puede bajar con cualquier navegador web que soporte *Java Script*[3]. Al cargar el sitio del SIG, hay una sección en el menú del lado izquierdo que dice: `Población/ Aspectos Generales`. El mapa se puede visualizar en la parte izquierda del *applet*[4] en el navegador web. Ahí, en la parte superior, hay una pestaña que tiene el nombre de: `metadatos`. Al hacerle click, aparecerá toda la información relacionada al archivo; fuente de información, *datum*, tipo de proyección, autores, etcétera. Busquemos una liga llamada: `Datos:ESRI Shapefile(SHP)`. Hay dos ligas con el mismo nombre, la diferencia está en la proyección utilizada. Sin pérdida de generalidad elegiremos el que tiene sólo coordenadas geográficas *WGS84*. Una vez bajado el archivo y previa descompresión se puede proceder a cargarlo en QGIS. Vemos que el archivo descomprimido tiene muchos otros archivos. Por ahora el que nos interesa es el que tiene la extensión `.shp`. Para cargarlo basta con abrir una terminal y escribir en la línea de comandos:

[3] Los navegadores más populares como: Firefox, Crome e Internet Explorer lo soportan.
[4] ver glosario.

```
juan@consola:/home/GIS/tutorial$ qgis
```

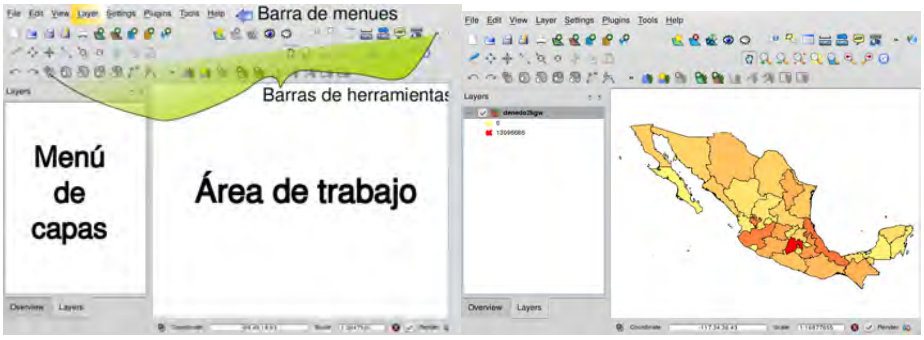
La figura 3.1a muestra una captura de pantalla del programa recién ejecutado nombrando los elementos de la interfaz gráfica. [5] En la barra de menues, si damos click en: `Layer` (resaltado en amarillo en la figura 3.1a) y después en `Add vector layer` aparecerá un cuadro de diálogo donde podremos seleccionar el archivo con formato `shp`. El programa lo cargará pero seguramente nos mostrará el mapa en un sólo color. Supongamos que queremos ver qué estados son los más densamente poblados. La mayoría de los SIGs pueden representar esto fácilmente utilizando un gradiente de color en donde el usuario debe asignar uno para el valor máximo y otro para el valor mínimo, el SIG hace una interpolación lineal a los valores intermedios asignándole un color distinto a cada uno. Para realizar esto en QGIS debemos dar click derecho en el ícono con el nombre del mapa cargado en el menú de capas, que es la barra del lado derecho (ver figura 3.1a), en este caso el mapa es llamado: denedo2kgw (ver figura 3.1b). Aparecerá un menú emergente con varias opciones. Seleccionemos la que lleva el nombre de Preferencias (*Preferences*), esta abrirá un cuadro de diálogo con varias pestañas. De momento vayamos a la sección: `Symbology`. Aquí podremos seleccionar algún atributo de la base de datos asociada al mapa vectorial y visualizarlo de varias formas, dependiendo nuestras necesidades. Podemos dividir los valores en clases y darle un color a cada una. Si son puntos podemos asignarle íconos de diferentes formas, colores o tamaños, incluso cargar íconos personalizados. Para este ejemplo asignemos un color contínuo *(Continuos Color)* a un campo de área (polígonos) como el de `PO_TO_2000`, que es el que representa los valores de la población total por estado. Elijamos un color rojo para el valor máximo (Distrito Federal) y uno amarillo para el mínimo (Baja California Sur), del resto se encarga QGIS. Dentro del cuadro de Preferencias existen otras secciones que es importante mencionar para conocerlas.

**General:** En esta sección podemos cambiarle el nombre a la capa, así como la proyección y *datum* para que concuerde con otras. Esta opción se puede ejecutar *al vuelo* (i.e. en tiempo real), pero consume más recursos.

**Metadata:** Aquí se encuentra toda la información referente al archivo que estemos utilizando: su proyección, fecha de creación, ubicación en disco, sistema de coordenadas, *datum*, número de atributos y primitivas geométricas.

**Labels:** La mayoría de los datos tienen campos tipo `String` (cadena de caracteres) que hacen referencia a nombres, adjetivos, códigos, etcétera. Estas cadenas pueden insertarse en el mapa en forma de etiquetas (*labels*) para tener una visualización directa de su localización. En esta sección se gestionan todas las configuraciones referentes a este tema. En el mapa de densidad utilizado se pueden hacer estos ejercicios en el campo `Entidad`.

[5] Atención: La versión de QGIS que estamos manejando en este manual está en inglés por tanto, los nombres de las herramientas y los menús estarán dados en este idioma. Esta aclaración es necesaria porque recientemente han salido algunas versiones en español y debido a la traducción los nombres pueden variar un poco.




(a) Captura recien ejecutado

(b) Captura visualizando el mapa de densidad poblacional de CONABIO Comisión Nacional para el Conocimiento y Uso de la Biodiversidad (CONABIO) (2006). Se utiliza un gradiente de color para el atributo `De_po_2000`.

Figura 3.1: Capturas de pantalla de QGIS

**Action:** Es posible ejecutar una tarea, aplicación, cálculo, etcétera, cada vez que se le pida a QGIS información sobre algún atributo del mapa. Qgis mandará el valor del atributo a alguna otra aplicación externa, pasándola como argumento. Por ejemplo, podríamos querer hacer una busqueda en Internet de algún campo en particular, buscar en una base de datos externa, hacer referencia a un archivo, crear reportes de localidades, mostrar gráficas de algún tipo, etcétera. Las posibilidades son tantas como nuestra creatividad y necesidad para resolver los problemas que algún proyecto nos demande. Para crear una *acción* bastará con introducir la información requerida en el cuadro de diálogo (ver figura 3.2). El campo importante es el llamado `Action` porque aquí es donde ejecutaremos la acción. Por fortuna tiene la misma sintáxis que el *shell*. Si quisieramos buscar en *Google* algún valor de un atributo en el mapa, por ejemplo, el nombre que algún objeto del mapa tenga en el campo `Entidad`, debemos poner primero el comando que vamos a llamar, en este caso sería *firefox* seguido de la URL: `http://www.google.com/search?q=` y el nombre del atributo antecedido por el signo `%` que servirá como variable. Una vez hecho esto damos click en `Update action` y cerramos el cuadro. Podemos correr la *acción* desde la tabla de atributos o con la herramienta *Identify Features* (ubicada en la barra de herramientas) que despliega un cuadro con todos los atributos relacionados a un punto seleccionado por el usuario. Una vez abierto este cuadro, al darle click **derecho** al atributo de interés, aparecerá el nombre de nuestra acción en el menú emergente ( Así se le llama al menú que sale en pantalla al dar click derecho en cualquier aplicación).

**Attributes:** Nos servirá para editar la tabla de atributos del mapa, incluso también se pueden agregar más de éstos. Utiliza la biblioteca OGR.

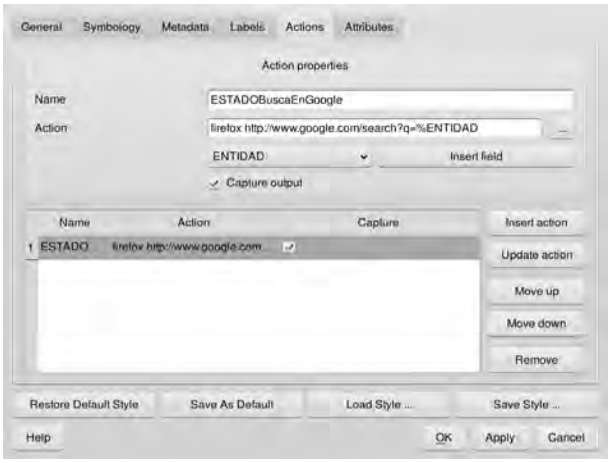


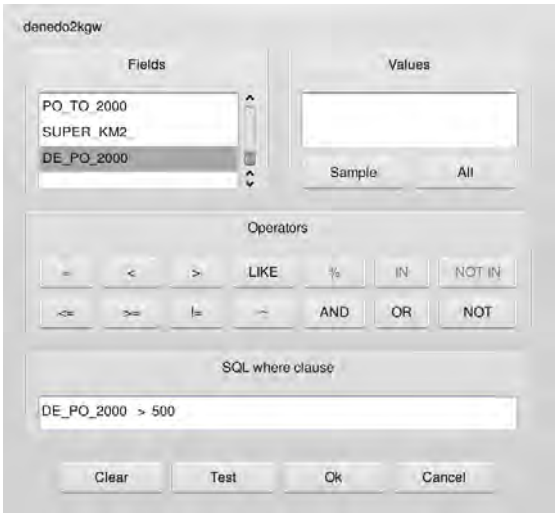

Figura 3.2: Cuadro de diálogo para crear *acciones*.

**Busqueda de atributos** La tabla de atributos proporcionada por el archivo *Shapefile* es una base de datos que puede ser accedida por QGIS gracias a la implementación del estándar del Lenguaje de Consultas Estructuradas *SQL*. De esta forma podemos hacer consultas complejas utilizando los operadores de *SQL*. Por ejemplo, si quisieramos encontrar los estados del país con más de 500 habitantes por $km^2$ deberíamos abrir la tabla de atributos y dar click en el botón: `Open Attribute Table`. Aparecerá una tabla con toda la información de los atributos. Como nuestra busqueda es más compleja de lo que nos ofrecen las opciones a la vista, debemos dar click en el botón: `Advanced Search` que tiene puntos suspensivos (...) . Ahí podemos realizar busquedas más complejas. En este caso buscamos en el campo `den_pob_2000` los valores que sean mayores a 500. Al finalizar podemos visualizar los lugares de interés presionando las teclas: `Ctrl + J`. La figura 3.3 muestra los resultados de esta operación.

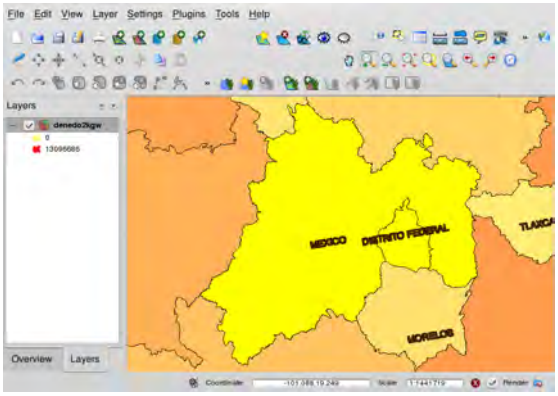
El proyecto multidisciplinario llamado UNIBIO (Instituto de Biología, 2008) administrado en la UNAM, pretende sistematizar y publicar en Internet toda la información sobre biodiversidad albergada en esa institución. El sitio está actualmente en construcción pero promete tener información georrefenciada de colectas y distribución de especies en todo el país con acceso remoto via PostGIS. Forma parte del macroproyecto *Sistema de Informática para la Biodiversidad y el Ambiente* (SIBA), el cual junto con el Instituto de Geografía (2008) ponen a disposición pública información geográfica útil para los científicos ambientales.

**Mapas de calles y carreteras** Hay un proyecto similar al conocido *Google Maps* llamado *OpenStreetMap*. [6] La diferencia es que toda la información mostrada es *Libre* bajo la licencia GPLv3. Usuarios de todo el mundo han recorrido millones de caminos para que podamos contar con esta plataforma. Los datos pueden ser bajados en la sección `export` dentro del sitio. Un "inconveniente" es que estos datos tienen un formato propio (`.osm`). Por fortuna las nuevas

[6] http://openstreetmap.org




(a) Cuadro de *busqueda avanzada*. Utiliza algunos comandos de SQL para realizar consultas.




(b) Visualización de los estados que cumplen la propiedad buscada resaltados con color amarillo.

Figura 3.3: Búsqueda y visualización de un atributo particular, en este caso estados con densidad poblacional $> 500\ hab/km^2$.

# ¿Sabías que:

existe una implementación *libre* del estándar SQL llamado *PostgreSQL*? Se ha desarrollado un *plugin* para este administrador de bases de datos para que soporte datos georreferenciados. Esta aplicación, llamada *PostGIS* es utilizada por la gran mayoría de los SIG tanto *libres* como comerciales. QGIS no es la excepción, y posee herramientas de conexión, creación y conversión de datos *Shapefiles* a PostGIS . Esta herramienta se llama *SPIT* y viene en la instalación base de QGIS. Se puede buscar en el menú `Plugins`.

versiones de QGIS incorporan un *plugin* que permite la lectura y escritura en este formato. En la versión actual (*MIMAS*) hay errores en el módulo de descargas automáticas que se pueden evitar descargando los datos del sitio en el formato `OpenStreetMap XML` y cargándolos manualmente en QGIS con el *plugin* mencionado. Para ejemplificarlo he descargado una región al sur de la Ciudad de México. Como se puede ver en la figura 3.4b el mapa contiene terrenos (polígonos), caminos (líneas) y sitios de "interés" (puntos), divididos en tres mapas.

**Servicios web WMS** El *Open Geospatial Consortium* diseñó las especificaciones para poder proporcionar un servicio web de mapas llamado WMS (*OpenGIS Web Map service* ). Esta implementación provee una interfaz HTTP simple para hacer consultas de mapas ráster o vectoriales a distintas bases de datos geospaciales que pueden estar distribuidas en varios sitios (como el caso del servidor JPL de la NASA que provee imágenes satelitales en mosaico mostrando todo el globo terráqueo en los doce meses del año pudiéndose ver como crecen y disminuyen los hielos invernales en los dos hemisferios simultáneamente). Una petición WMS define el área de interés que se desea procesar, así como una capa donde se va a desplegar esa información. La petición regresa una o más imagenes georreferenciadas que pueden tener distintos formatos (JPEG, PNG,GIFF, etc). Éstas se pueden visualizar en cualquier aplicación que soporte la especificación WMS, puede ser el navegador mediante un applet Java o un SIG de escritorio con soporte web. La especificación completa se puede revisar en (Open Geographic Consortium, 2009).

Tanto QGIS como GRASS soportan este servicio. GRASS baja la información a la computadora y luego la convierte a su formato nativo, mientras que QGIS la manipula directamente.

El sitio UNIGEO (Instituto de Geografía, 2008) tiene por objetivo crear un sistema avanzado de información para ordenar, sistematizar y analizar en línea la vasta información geográfica primaria, del Instituto de Geografía de la UNAM, y sentar las bases para extender este proceso a otros institutos y facultades que producen información espacial, tanto primarias como sus derivados temáticos. Actualmente proveen servicios WMS disponibles en:

`http://132.248.26.13:8080/geoserver/wms` [7]

**Ejemplos WMS** QGIS tiene, de forma predeterminada, cargados varios servidores WMS. Un ejemplo ilustrativo es el servicio *Blue Marble* de la Agencia Espacial Norteamericana (NASA) . El servidor WMS de la NASA se llama *One Earth Web Map Server* y posee un catálogo grandísimo de información ráster acerca de todo tipo de fenómenos planetarios. La mayoría sólo cubre el territorio estadounidense, pero algunos cubren todo el globo como el *Blue Marbel* (BMNG) (Stockli et al., 2005). Este servicio es un mosaico sin nubes y con correcciones ópticas para mostrar las características de la superficie terrestre a lo largo del año, tal y como el ojo humano lo podría ver desde el espacio. Una capa WMS se puede abrir desde el menú: `Layer / Add WMS layer`. Aparecerá un cuadro de diálogo con los servidores disponibles. Si aún no los hay debemos dar click en el botón `Add default servers`. Para ver el

[7] Para acceder a esta información es necesario un SIG que soporte WMS

servicio *Blue Marbel* debemos conectarnos al servidor `NASA (JPL)` y escoger la carpeta `BMNG`; ahí podremos seleccionar cualquier archivo disponible, haciendo click en `add` se carga la capa seleccionada (ver figura 3.4a).

El *Instituto Nacional de Geografía y Estadística* (INEGI) tiene para disposición pública un servicio WMS de ortofotografías (INEGI, 2009) de todo el territorio nacional, en dos escalas diferentes; 20 km y 3 m. Para acceder a esta información es necesario un SIG que soporte WMS (como QGIS). La dirección de este servicio es:

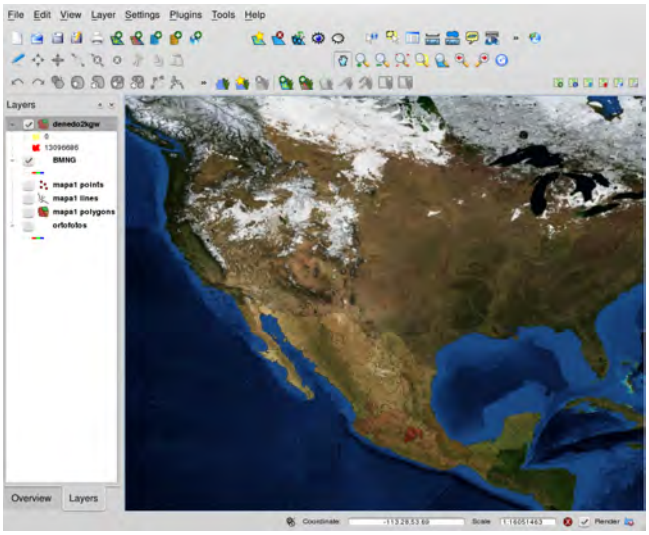
`http://antares.inegi.org.mx/cgi-bin/map4/mapserv_orto`

Para cargar este sitio en la lista de QGIS basta localizar y hacer click en el ícono de la barra de herramientas que lleva el nombre de `add WMS layer` y tiene un globo terraqueo que lo identifica y seleccionar el botón `NEW`. Dependiendo de la escala seleccionada tendremos que tener una región en esa escala para poder ver los rasters.

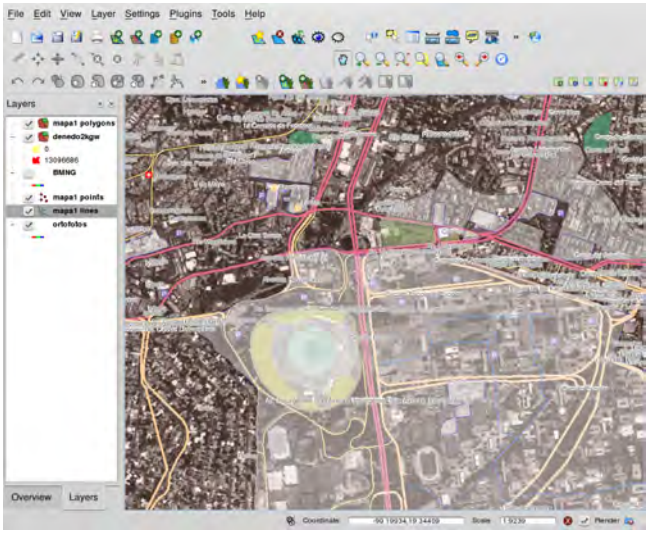
**Creación y edición de mapas vectoriales.** Supongamos ahora que quisieramos hacer un mapa vectorial de la Reserva del Pedregal de San Ángel (REPSA) en Ciudad Universitaria y zonas aledañas. Lo primero que necesitamos es un polígono del área y algunos caminos y veredas. Esto quiere decir que debemos de tener polígonos y líneas como datos, para lo cual debemos hacer un mapa shapefile con la información del área (polígonos). Para esto nos servirá la capa que bajamos del proyecto *OpenStreetMap* de la zona sur de la Ciudad de México. Cargándola con el *plugin* OSM que ya vimos, podemos seleccionar las tres áreas pertenecientes a la zona núcleo definidas en el sitio oficial de la reserva (UNAM, 2007). Esto lo podemos hacer de varias formas; una puede ser seleccionando cada una de las zonas y guardando cada selección como un archivo *Shapefile* en el menú `Layer` y luego hacer un archivo general con la unión de estas tres zonas. Esta operación, junto con otras operaciones de conjuntos como intersección, complemento y resta, se puede hacer con las herramientas de geoprocesamiento incluidas en el menú `Tools / Geoprocessing tools`. Otra forma de hacernos del mapa es por medio de la aplicación de búsqueda avanzada en la tabla de atributos. En este caso sólo bastó introducir la siguiente consulta SQL en el campo correspondiente:

```
name = 'Reserva del Pedregal de San Ángel' OR name = '
                        forest'
```

Recordemos que es importante ver primero los nombres de los atributos y los valores buscados para tener éxito en la consulta. En este caso el nombre del atributo es: *name* y el valor que nos interesa es *Reserva del Pedregal de San Ángel* o *forest*, ya que por alguna razón la persona que subió la información del lado este de la reserva no fue consistente con el nombre verdadero y utilizó el genérico *forest*. La búsqueda regresa un aviso diciendo que se encontraron tres registros válidos, mismos que se seleccionan automáticamente, por lo que sólo hará falta crear el mapa *.shp* ( `Layer / Save selection as Shapefile` ). También nos gustaría cambiarle los nombres *forest* por los correctos. Para esto, ya habiendo creado el mapa *.shp*, podemos entrar a sus propiedades y en la pestaña `Attributes`; en la columna `edit widget` seleccionamos `value map`. Se abrirá un cuadro de diálogo con la capa a




(a) Mosaico del proyecto *Blue Marbel* adquirida con el servicio *WMS*. Correspondiente a diciembre de 2004. (Stockli et al., 2005), *NASA Earth Observatory (NASA Goddard Space Flight Center)*




(b) Imagen compuesta por las capas de *calles y terrenos* del archivo de OpenStreetMap con valor transparente. Ortofotografías del servicio *WMS* de INEGI en 2009 para resolución de 3M. Mapa de densidad poblacional (Comisión Nacional para el Conocimiento y Uso de la Biodiversidad (CONABIO), 2006) superpuesto con valor *alpha* (transparente) y etiquetas del mapa *calles* con *buffer* blanco para mejor visualización.

Figura 3.4: Ejemplos de visualización multicapa con servicios *WMS*

cargar y demás opciones útiles, seleccionamos el campo *forest* y escribimos el nombre adecuado i.e. `Reserva del Pedregal de San Ángel (REPSA) núcleo oriente`. Hay algunas regiones como el Jardín Botánico, el Espacio Escultórico y algunos otros edificios que según *Programa Universitario del Medio Ambiente* (PUMA-UNAM, 2007) no forman parte de la reserva, por tanto hay que quitarlos del área.

**Es hora de editar nuestro mapa**. Debemos comenzar por seleccionar la capa de interés (donde está el objeto a editar) en el menú de capas; después en la barra de menú damos click en `Layer / toggle editing`. Aparecerán ahora, en la barra de herramientas, herramientas de edición que antes no estaban disponibles, dependiendo del tipo de *geometría* que estemos utilizando (i.e. puntos, líneas o áreas). Para este caso en particular (área) podemos utilizar la herramienta para agregar agujeros (`add ring` ver figura 3.5), [8] además de herramientas básicas de edición como: agregar, borrar y mover vértice, también hay otras un poco más avanzadas que nos servirán mucho para el trabajo de digitalización y edición, que es el más engorroso. Estas herramientas son:

`Add island` Se puede crear un área nueva fuera de la región de interés.

`Create / Delete ring` Crea o borra un agujero dentro de la región de interés, para así poder hacerla simplemente conexa.

`Delete part` Borra una subregión del área de interés.

`Simplify feature` Elimina vértices del perímetro del área, i.e. lo hace más poligonal, más cuadrado.

`Reshape feature` Seleccionando una región de algún objeto podemos trazar una nueva línea para redefinir su frontera (perímetro).

`Split feature` Elimina una región del objeto seleccionado; la región eliminada dependerá de como fue trazada la línea de corte.

La edición comienza cuando el cursor cambia de forma y se le da click con el botón primario (izquierdo) al objeto a editar en el punto a empezar. Para terminar un trazo o alguna otra acción de edición debemos dar click con el botón secundario (derecho). Sólo puede editarse en el objeto que ha sido seleccionado con anterioridad. Para que los cambios surjan efecto se debe remover la selección de edición en: `Layer / Toggle Editing`. Los nombres de los atributos pueden modificarse también si esta opción está activada.

**Creación de mapas vectoriales.** Muchas veces tendremos la necesidad de construir nuestros propios mapas tomando las formas de atributos de otros e incorporando nueva información (no disponible en otros mapas). A este proceso se le llama *digitalizar* y es un proceso tardado porque generalmente tiene que hacerse a mano con las herramientas de edición que ya hemos visto. La forma más común de encontrarnos con estos problemas es cuando tenemos datos ráster y el proyecto requiere obligadamente datos en formato vectorial. Existen herramientas de conversión automática pero muchas veces

[8] Matemáticos: Con esta herramienta podemos transformar una región simplemente conexa en una múltiplemente conexa

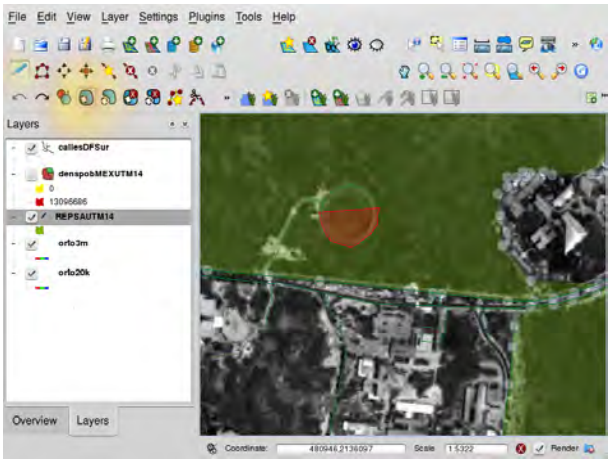
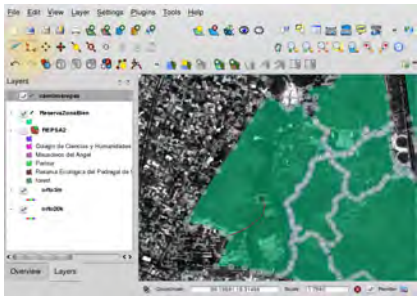



Figura 3.5: Modo de *Edición de mapas vectoriales*. En amarillo se resalta la herramienta de *agujeros* (`Add Ring`) que es la que se usa para eliminar una región en el interior de un polígono.

es necesario hacer algunas correcciones a mano. Un ejemplo en concreto: con las ortofotos provistas por INEGI tenemos que hacer un mapa de los caminos dentro de la reserva ( REPSA ). Para fines prácticos, tomaremos por caminos los manchones claros carentes de vegetación que se ven en las ortofotos (ver en la figura 3.6a). QGIS tiene una interfaz para hacer estos mapas. Vayamos a `Layers / New Vector Layer`, y aparecerá un cuadro de diálogo con la información requerida para crear el nuevo mapa. El apartado llamado `Type` es donde se define la geometría de los datos. i.e. puntos, líneas o áreas (*polygon*). Para este ejemplo deberá ser `Line`. El siguiente apartado `Attributes` es donde se especifican los atributos que pueden ser nombres, categorías, valores enteros o flotantes. Se debe especificar el nombre del atributo así como su tipo, es decir si es cadena de caracteres (`String`) y el tamaño máximo de caracteres permitido (`Width`), entero (`Integer`) o flotante (`Float`). Se pueden agregar tantos atributos como sea necesario. Para este ejemplo sólo he agregado un atributo llamado `NOMBRE`. Cuando se le da *Aceptar* se guardará como un archivo `ESRI Shapefile`. Una vez guardado el mapa se cargará como una nueva capa sin información. Para empezar la digitalización debemos ejecutar las herramientas de edición antes mencionadas (`Layer / Toggle Editing`). Para empezar la digitalización debemos utilizar la herramienta `Capture Line`; en la figura 3.6b se puede ver resaltado en amarillo el ícono correspondiente. El cursor del ratón cambiará, señalando que está activa la opción de captura. Ahora podemos empezar a trazar los caminos de la reserva. Cuando hayamos terminado de trazar un camino podemos registrar el valor de los atributos con el botón derecho del ratón. Una vez que hayamos terminado deberemos salirnos de la opción de edición ( nuevamente `Layer / Toggle Editing` ). Una vez salvado el trabajo habremos creado un mapa *.shp* con los caminos de la REPSA que puede ser utilizado en cualquier SIG; incluso se puede exportar y subir a *OpenStreetMap* utilizando el correspondiente *plugin*.

(a) Ortofotográfia de la Reserva del Pedregal de San Ángel (REPSA) tomada del acervo en línea del INEGI. Escala 1:11400 con resolución de 20$km$. Lo que se digitalizó como *caminos* son las líneas claras al interior de la REPSA.



(b) Captura de pantalla de QGIS mostrando las herramientas de edición en el modo de *captura de línea*. Resaltando el ícono en amarillo.

Figura 3.6: Creación / digitalizacíon de mapas vectoriales con QGIS

**Georreferenciar imagenes.** Es posible que tengamos imágenes en bruto (sin referencia geográfica) de algunas regiones y queramos incorporarlas al proyecto en forma de ráster. Estas imágenes pueden brindar información muy importante pero debido a que no tienen un sistema de referencia geográfica, se desperdician en el cajón o en el disco duro. Los desarrolladores de QGIS han pensado en esto y han incorporado un *plugin* para empalmar de forma muy precisa fotografías o imágenes en mapas georreferenciados bajo cualquier sistema de coordenadas o proyección. Este *plugin* se llama *Georeferencer GDAL* y se puede encontrar con la herramienta de administración de *plugins* en: `Plugins / Manage Plugins`. Una vez ejecutado este módulo se abrirá un cuadro de diálogo preguntándonos el lugar donde se encuentra la imagen a georreferenciar, los algoritmos de interpolación disponibles, el nombre del futuro ráster y el nombre del archivo *world file (.wld)* que es donde se guardará la metainformación del ráster nuevo, i.e. la información referente al *datum* y a la proyección.
Para seguir con la REPSA probemos con una imagen aerea en bruto que se puede descargar de:

`http://www.repsa.unam.mx/imagenes/reserva_aerea_600.png`

Como vemos es una imagen común y corriente con una resolución muy pequeña. A pesar de esto tiene el suficiente detalle para poder georreferenciarse con las ortofotos de INEGI con las que hemos venido trabajando. Para empezar el proceso debemos abrir el *plugin* y cargar la imagen. Una vez cargada aparecerá un cursor en forma de cruz en el cuadro sobre la imagen (ver figura 3.7). El proceso es el siguiente: debemos encontrar puntos en la imagen que sean facilmente identificables para después capturarlos en el mapa cargado en la pantalla de QGIS ( este lugar se conoce como *Map Canvas*). De esta forma le asignamos a cada punto nuevo de la imagen en bruto un punto con coordenadas geográficas, como se muestra en la figura 3.7.

Se pueden agregar tantos puntos como se quieran. Entre más y mejor ubicados estén, más precisa será la interpolación y el empalme. También es mejor utilizar una interpolación no lineal para tener un error de propagación

menor; el inconveniente es que el tiempo de cálculo puede ser muy largo. Si son pocos puntos y la región es pequeña no hay problema en usar una interpolación lineal.

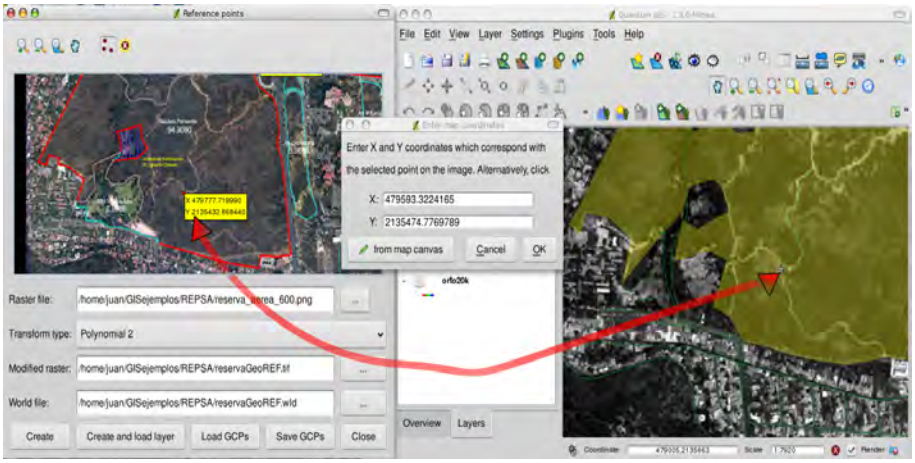



Figura 3.7: Módulo para georreferenciar imagenes y transformarlas a imagenes ráster georreferenciadas. Utiliza la biblioteca GDAL

**Importar datos en formato de texto.** En muchas ocasiones nos enfrentaremos al problema de procesar información que no está estructurada en mapas. No siempre existirá un mapa adecuado para nuestras necesidades y muchas veces tendremos que crear los nuestros propios. Podría tratarse de puntos de colecta, muestreos u otros sitios de interés. Tanto QGIS como GRASS (que veremos más adelante) tienen la capacidad de cargar esta información y desplegarla en forma de un mapa vectorial. [9]
Para cargar estos datos, primero necesitamos un archivo de texto simple ( `txt`, `dat` o `csv` [10]), que debe tener la siguiente estructura:

- Un caracter (símbolo) llamado *delimitador*, que servirá para separar los datos en diferentes variables o campos.
- Un *encabezado* en la primera línea del archivo que consta de los nombres de las variables separados por el *delimitador.*
- Un *cuerpo* donde cada tipo de dato estará separado por el *delimitador.*

Como se puede ver, el archivo de texto debe tener una estructura similar a una tabla (cuadro), en la que cada columna representa los posibles valores de una variable en particular. El nombre de ésta se define en la primera entrada de esa columna. Como cargará una capa vectorial, cada variable será un atributo del mapa. La figura 3.8 muestra un ejemplo de la estructura de un archivo de este tipo.

[9] Sobra decir que para que podamos hacer esto, los datos a cargar tienen que tener valores de longitud y latitud.
[10] *comma separated value*

QGIS tiene un módulo para cargar este tipo de datos. Este *plugin* se puede cargar con el `Administrador de Plugins (Plugin Manager)` y se llama `Add delimited text layer`. Al ejecutarlo aparecerá un cuadro de diálogo con las opciones necesarias para cargar los datos, como el *delimitador* antes mencionado.

```
"NOM_LOC"|"LATITUD"|"LONGITUD"|"ALTITUD"|"POB_TOT"|"POB_MASC"|"POB_FEM"
"Aguascalientes"|21.88|-102.3|1870|663671|319649|344022
"Granja Adelita"|21.87|-102.37|1900|10|"*"|"*"
"Agua Azul"|21.88|-102.36|1850|30|13|17
"Rancho Alegre"|21.85|-102.37|1880|0|0|0
"Los Arbolitos (Rancho de los Arbolitos)"|21.78|-102.36|1880|6|"*"
"Ardillas de Abajo (Las Ardillas)"|21.95|-102.19|2030|0|0|0
"Arellano"|21.8|-102.27|1890|1349|661|688
"Bajio los Vazquez"|21.75|-102.12|1960|68|32|36
"Bajio de Montoro"|21.76|-102.29|1870|3|"*"|"*"
"Residencial San Nicolas (Baños la Cantera)"|21.85|-102.36|1860|85|46|39
"Buenavista de Peñuelas"|21.72|-102.3|1810|821|410|411
"Cabecita 3 Marias (Rancho Nuevo)"|21.77|-102.41|1900|185|85|100
"Cañada Grande de Cotorina"|21.78|-102.24|1980|359|183|176
"Estacion Cañada Honda"|22.01|-102.2|1910|414|210|204
"Los Caños"|21.78|-102.47|1900|1053|507|546
"El Cariñan"|21.89|-102.37|1880|298|140|158
"Granja el Carmen"|21.89|-102.32|1860|0|0|0
"El Cedazo (Cedazo de San Antonio)"|21.7|-102.31|1855|227|117|110
"Centro de Arriba (El Taray)"|21.73|-102.5|1860|1076|494|582
"Cieneguilla (La Lumbrera)"|21.73|-102.45|1800|853|399|454
"Cobos"|21.83|-102.24|1960|9|5|4
```

Figura 3.8: Formato de un archivo de texto para ser cargado y procesado por QGIS o GRASS. Obsérvese el *delimitador* | y el encabezado. Datos tomados del *Conteo de Población y Vivienda 2005, (INEGI, 2005)*

.

**Preprocesamiento de texto.** En la red hay mucha información disponible en formato simple. Quise utilizar datos del territorio nacional para ejemplificar el geoprocesamiento por esta vía. Sin embargo aunque sea fácil crear y cargar datos de esta forma, muchas veces los datos que necesitamos no se pueden cargar directamente y se tienen que editar antes. Si son pocos datos, la edición puede hacerse manualmente modificando el archivo con cualquier editor de texto (*Gedit, Kate, Emacs, Notepad (Windows)*). Si son algunos cientos, la edición podría hacerse en una hoja de cálculo como *OpenOffice SpreadSheet, Excell (Windows)*. Pero cuando son miles de datos la única solución es hacer un pequeño programa que lea todo el archivo y lo convierta a uno con el formato que deseamos. A esta acción se le llama *preprocesamiento* y los bioinformáticos son expertos en esto. Generalmente se utilizan lenguajes diseñados para manejar cadenas de caracteres, como `Perl`, `Rubi o Python`; personalmente prefiero este último. Aprender a programar no es difícil y sobre todo es muy recomendable porque nos permite ahorrar mucho tiempo.

**Ejemplo: El caso del Catálogo de INEGI.** El portal de INEGI tiene mucha información geoespacial disponible de acceso público. Un servicio de éstos es el *Catálogo de entidades federativas, municipios y localidades* XII Censo de Población y Vivienda (2005). Este catálogo tiene la localización, población total, femenina, masculina y número de viviendas habitadas de todas las localidades del territorio mexicano, un total de 293 716. El sitio tiene la opción de bajar los datos en texto simple [11], así que lo natural sería bajarlos y cargarlos en QGIS directamente. INEGI advierte que la latitud y longitud que acompañan a todos sus datos están en grados *sexagesimales* y si se quiere utilizarlos en un SIG se debe hacer una conversión. Una *sencilla* conversión que requiere obviamente de un programa de preprocesamiento para esos datos, que puede ser hecho por nosotros. La pregunta obligada es: ¿ Por qué INEGI teniendo toda esa información no proporciona los datos ya convertidos ? Esto no nos va a detener. Una guía de programación va más allá de las intenciones de este manual pero no pude evitar hacer algo al respecto. Hice un *script* en `Python` que resuelve este problema creando un nuevo archivo llamado `salida.cvs`. El código del programa está en el apéndice.

Con este *script* podemos cargar el archivo `salida.cvs` en QGIS utilizando el símbolo | como delimitador. Es importante hacer notar que el sistema de coordenadas de estos datos esta en *WGS84* y no esta proyectado bajo ningún modelo, sólo está definido el elipsoide. Si se necesita cambiar de coordenadas se debe hacer una conversión *al vuelo* (sólo en QGIS) o crear un nuevo mapa o conjunto de datos con la nueva conversión.

**Lectura y escritura en dispositivos GPS** El *plugin* `GPS Tools` brinda herramientas para cargar datos de dispositivos GPS de varias marcas. También tiene la opción de subir datos (rutas y puntos) al aparato. De esta forma podríamos subir caminos, sitios de colecta, parcelas y planificar mejor muestreos y experimentos. Este módulo usa la biblioteca *gpsbabel*, por tanto antes de usarla se requiere la instalación de ésta. Si nuestro dispositivo no esta soportado por esa biblioteca o el controlador está restringido a *Windows* podemos descargar el archivo `.gpx` del dispositivo desde otra computadora y cargarlo en QGIS con mismo módulo `GPS Tools`. Para subir datos al dispositivo, así como para modificarlos debemos tener un archivo `.gpx`.

Con estos ejemplos hemos visto algunas generalidades de QGIS, podremos volver a él más adelante cuando estemos estudiando GRASS. QGIS es un proyecto que está creciendo muy rápido y eso se debe en gran medida a la capacidad que tiene para trabajar con módulos (*plugins*); profundizar en cada uno sería desgastante, si se quiere revisar alguno en particular o buscar una explicación más detallada se puede hacer en la página del proyecto o en el *wiki*-manual de usuario en: (Sherman et al., 2009).

### 3.1.2. GRASS

El nombre *GRASS* viene del acrónimo en inglés para *Geographical Resources Analysis Support System*, algo así como Sistema Geográfico de Apoyo para

[11] Ellos lo llaman Texto DOS

Análisis de Recursos. *GRASS* fue creado por el *Cuerpo de Ingenieros del Laboratorio de Investigación de Ingeniería de la Construcción* de la armada de los Estados Unidos, *USA-CERL* por sus siglas en inglés, entre 1982 y 1995 como herramienta para administrar el territorio de las instalaciones militares de E.U. En 1995 el *USA-CERL* abandonó el desarrollo de GRASS (M.Neteler y Mitasova, 2008) y lo adoptó el laboratorio de geografía física y ecología del paisaje en la Universidad de Hannover bajo la dirección de *Marcus Neteler*.Quien liberó el código en 1998 con la versión 4.2.1. El desarrollo de la versión 5.0 comenzó cuando se liberó el código con la licencia *GNU-GPL* en 1999, conviertiéndose en el primer SIG Libre. A partir de entonces numerosas universidades, agencias gubernamentales y compañias han usado y desarrollado GRASS y se ha convertido en el proyecto SIG de Software Libre más grande y completo del mundo (GRASS Development Team, 2009).

En la versión estable de GRASS, la 6.4.x, existen alrededor de 350 módulos, que en conjunto combinan motores de procesamiento avanzados para datos geoespaciales tipo ráster y vectoriales, que se integrán en un sólo paquete para hacer análisis espaciales, procesamiento de imágenes, visualización, geoestadística y modelado discreto y continuo. Para mayor información se puede revisar el manual compilado por *GRASS Development Team (2009)*.

En el sitio del proyecto : `http://grass.itc.it` se puede encontrar toda la información necesaria para trabajar con GRASS, desde pequeños tutoriales, ejemplos, documentación, preguntas frecuentes, formas de citar el proyecto y de contribuir al mismo.

**Instalación Estable**

GRASS tiene una versión estable en los repositorios de Ubuntu. Para instalarlo se puede utilizar la interfaz gráfica (*Synaptic o Adept Manager*) o la línea de comandos (i.e. el *shell*). Esta última opción se instala con el comando siguiente:

```
sudo apt-get install grass
```

Una de las *desventajas* de GRASS es que utiliza su propio formato de datos [12] Esto quiere decir que debemos convertir nuestros datos a uno adecuado para él. Otro de los *inconvenientes* es que primero necesitamos definir una localidad para comenzar a cargar los mapas y poder hacer los análisis. Esto podría parecer engorroso al principio pero los desarrolladores de GRASS han optado por mantener esta estructura así porque a medida que los proyectos geomáticos van creciendo, los datos y los resultados también. El tener el material organizado hace que se pueda trabajar más fácil y eficientemente, en especial para trabajos en equipo.

[12]Cuando decimos *datos* nos referimos a cualquier tipo de información georeferenciada. Pueden ser mapas ráster, vectoriales, bases de datos, puntos, caminos e incluso recursos de internet.

**Estructura de datos**

GRASS está programado bajo el modelo *cliente-servidor*, esto es que organiza los datos de forma tal que un número ilimitado de usuarios [13] puede acceder a la base de datos, hacer análisis y guardar sus resultados simultáneamente sin afectar ni modificar el trabajo de los otros usuarios, en otras palabras, cada usuario tiene o debe tener su propio espacio de trabajo.

Los proyectos de GRASS están organizados por *localidades*. Cada localidad es administrada por una persona (administrador) y ésta tiene permiso para subir, modificar y eliminar los datos que van a estar a disposición de los demás usuarios del proyecto, es decir, de los usuarios de la *localidad*. Automáticamente cuando el administrador crea la localidad se crea una carpeta llamada `Permanent`. Cada usuario que trabaje en la *localidad* tendrá su propia carpeta donde podrá leer y escribir datos libremente. Normalmente todas las localidades de GRASS se guardan en la misma carpeta llamada *GRASS database*. La figura 3.9 muestra como es la jerarquía de los archivos.

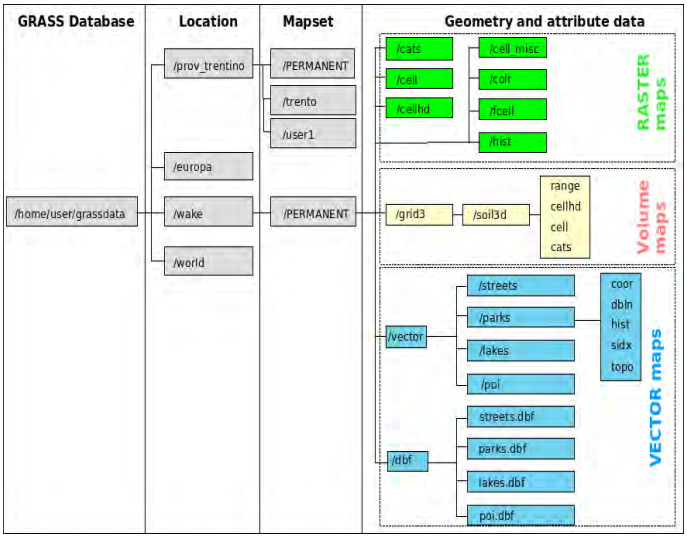



Figura 3.9: Jerarquía de localidades,usuarios y datos en *GRASS*. Tomado de http://grass.itc.it/

Por ahora sólo nos limitaremos a esta descripción. Veremos como utilizarlo en el siguiente capítulo, dedicado al geoprocesamiento de datos. Ahí aprovecharemos las capacidades de GRASS haciendo énfasis en datos con modelos ráster.

## 3.2. GeoKit de herramientas

Hemos hablado de las ventajas de la modularidad en los sistemas basados en *UNIX*, pero hasta ahora esta descripción no nos ha motivado realmente a escalar la *empinada* curva de aprendizaje en estos sistemas. Ahora

[13] El número de usuarios depende de la capacidad de la computadora que esté corriendo GRASS, del tamaño de los datos y de la memoria en el sistema.

vamos a poner en práctica estas metodologías o estándares y veremos que efectivamente al realizar tareas de mayor complejidad tanto para datos como para procesos, el tiempo se acorta de forma significativa.

Primero empezaremos por instalar un kit de herramientas geomáticas de línea de comandos para el *shell*, diseñadas especialmente para manejo de objetos georeferenciados, i.e. modelos *ráster o vectorial*. Una de estas herramientas, llamada GDAL/OGR, es esencialmente un conjunto de bibliotecas y controladores que nos pemitirán, entre otras muchas cosas, convertir objetos de un tipo a otro y transformarlos de un formato a otro. Esta herramienta es imprescindible porque la mayoría de las aplicaciones OSGIS (*Open Source GIS*) utilizan estas bibliotecas dentro de su *maquinaria*. Otra herramienta, llamada GMT (*Herramientas de mapeo genéricas*), por sus siglas en inglés, está diseñada para crear los mapas una vez procesados los datos, y permite crear mapas de excelente calidad desde la línea de comandos.

### 3.2.1. Herramientas en línea de comandos

La línea de comandos nos asusta a muchos al principio porque son muchos símbolos *"sin sentido"* que pocas veces producen resultados visualmente atractivos. No debemos olvidar que muchas de nuestras aplicaciones favoritas, especialmente en el mundo del *Software Libre* funcionan por dentro con estas herramientas. Aprender a utilizarlas nos ahorrará mucho tiempo y a la larga será imprescindible para trabajos más elaborados.

Una de las bibliotecas-aplicaciones de este estilo es la llamada *Biblioteca Geoespacial de Abstracción de Datos*, comúnmente *GDAL*. Brevemente mencionamos en la introducción que GDAL y su compañero OGR son bibliotecas que soportan una variedad muy grande de formatos de datos, tanto para modelos vectoriales como ráster. De hecho la utilería que proporciona soporte para modelos ráster es GDAL, OGR se usa para modelos vectoriales.

#### GDAL

GDAL es una biblioteca para leer, escribir y transformar varios formatos ráster geoespaciales. De hecho puede soportar los más populares; si se necesita de uno que no esté soportado en el paquete básico será necesario bajar el controlador, tal vez construir dependencias y compilar el código fuente[14]. La lista completa de formatos soportados por defecto se pueden encontrar en la página del proyecto [15] así como las posibles formas de instalación de otros formatos como MrSID (Warmerdam, 1998-2010). GDAL además de ser una biblioteca (i.e. define objetos ráster en abstracto) posee también varios comandos que se pueden utilizar directamente en el *Shell* para hacer transformaciones de coordenadas, proyecciones, fusionar imagenes, crear mosaicos (*merging*) e incluso quitar y corregir distorsiones (*warping*). Para ver la lista completa de estas utilerías se pude revisar la página: `http://www.gdal.org/gdal_utilities.html`, donde también se encontrará la documentación y ayuda para utilizar estos comandos.

[14] Los usuarios de Ubuntu pueden prescindir de todo este proceso con la anexión de un par de repositorios disponibles en el sitio de *launchpad* las instrucciones de esto en la sección **UbuntuGIS al rescate** (pág 78).

[15] `http://www.gdal.org/`

**Formatos *ráster* soportados**

GDAL soporta más de 90 formatos distintos (Warmerdam, 1998-2010). Entre los más populares, según Sherman (2008), podemos nombrar los siguientes:

GDAL: Formatos soportados

- Arc / Info ASCII Grid
- Arc / Info Binary Grid(.adf)
- USGS DOQ (.doq)
- ESRI .hdr Labelled
- ENVI .hdr Labelled Raster
- GMT Compatible netCDF
- GRASS Rasters
- TIFF/GeoTIFF (.tiff)
- GXF–Grid eXchange
- Hierarchical Data Format Release 4 (HDF4)
- Hierarchical Data Format Release 5 (HDF5)
- Erdas Imagine (.img)
- JPEG JFIF (.jpg)
- JPEG2000 (.jpg, .j2k)
- MrSID
- Portable Network Graphics (.png)
- ArcSDE Raster
- USGS SDTS DEM (*CATD.DDF)
- USGS ASCII DEM (.dem)

**Ejemplos GDAL | Reproyecciones ráster** Hemos visto en el capítulo anterior que aunque el modelo elipsoidal cuente con parámetros (*Semi-eje Mayor, $a$* y *Proporción de achatamiento, $f$*) muy aproximados a la "realidad" (como el *WGS84*) si queremos medir distancias o áreas entre objetos será imposible, pues las unidades de este modelo son en segmentos de arco, es decir, ángulos. Para que podamos medir en unidades cartesianas (metro, pie, milla, kilómetro, etc) es necesario trabajar con datos proyectados bajo algún modelo de los que ya hemos visto (ver pág 27).

**Averiguar la *Metainformación*** Para que los datos puedan ser procesados por cualquier SIG es necesario que, aparte de la información que están describiendo *per se*, tengan información acerca de su proyección, *datum*, parámetros del geoide, fecha de elaboración, autoría, etcétera. Al conjunto de toda información se le llama *Metadatos* . Estos datos pueden estar embebidos en el mismo archivo que utiliza el SIG para procesarlo (como el caso del formato ráster *GeoTIFF*) o puede estar en archivos separados (como los `.prj`) de los *Shapefile*. Estos *metadatos* por lo regular son archivos muy largos y se puede perder mucho tiempo buscando la información que necesitemos, que por lo general es la de la proyección. GDAL/OGR tiene un comando llamado `gdalinfo` que nos brinda la información necesaria, casi siempre suficiente, sin tener que explorar todo el archivo, que puede llegar a miles de líneas. Para ejemplificar esto usaremos un archivo ráster correspondiente a una imagen satelital de la región del Volcán de Cólima, que la bióloga Rocío Alanís Anaya nos dió como cortesía; forma parte de su trabajo de maestría relacionado con

el estudio de la sucesión vegetal en esa zona. Alanís-Anaya (2010). El comando se usa con la siguiente sintaxis: `gdalinfo [NombreArchivo.ext]`

La extensión debe ser una soportada por GDAL. Este comando tiene otras opciones que se pueden ver en el manual del mismo (i.e. `$: man gdalinfo`).

Para este ejemplo el archivo se llama `VolcanColima.tiff` y sus metadatos están embebidos en él mismo. Por tanto es imposible leer la metainformación directamente, pero, con este comando sí. La información obtenida de este archivo es:

```
juan@consola:/home/GIS/tutorial$ gdalinfo VolcanColima.tif
Driver: GTiff/GeoTIFF
Files: VolcanColima.tif
Size is 7747, 7357
Coordinate System is:
PROJCS["WGS 84 / UTM zone 13N",
    GEOGCS["WGS 84",
        DATUM["WGS_1984",
            SPHEROID["WGS 84",6378137,298.257223563,
                AUTHORITY["EPSG","7030"]],
            AUTHORITY["EPSG","6326"]],
        PRIMEM["Greenwich",0],
        UNIT["degree",0.0174532925199433],
        AUTHORITY["EPSG","4326"]],
    PROJECTION["Transverse_Mercator"],
    PARAMETER["latitude_of_origin",0],
    PARAMETER["central_meridian",-105],
    PARAMETER["scale_factor",0.9996],
    PARAMETER["false_easting",500000],
    PARAMETER["false_northing",0],
    UNIT["metre",1,
        AUTHORITY["EPSG","9001"]],
    AUTHORITY["EPSG","32613"]]
Origin = (510777.00000000000,2184582.00000000000)
Pixel Size = (28.50000000000,-28.50000000000)
Metadata:
  AREA_OR_POINT=Point
  TIFFTAG_XRESOLUTION=72
  TIFFTAG_YRESOLUTION=72
  TIFFTAG_RESOLUTIONUNIT=2 (pixels/inch)
Image Structure Metadata:
  INTERLEAVE=BAND
Corner Coordinates:
Upper Left  (  510777.000, 2184582.000)
            (104d53'49.67"W, 19d45'24.79"N)
Lower Left  (  510777.000, 1974907.500)
            (104d53'53.80"W, 17d51'42.77"N)
Upper Right (  731566.500, 2184582.000)
            (102d47'24.97"W, 19d44'35.81"N)
Lower Right (  731566.500, 1974907.500)
            (102d48'53.47"W, 17d50'58.80"N)
Center      (  621171.750, 2079744.750)
            (103d51'0.40"W, 18d48'21.30"N)
Band 1 Block=7747x1 Type=Byte, ColorInterp=Gray
```

Vemos que la línea 1 nos dice que el mapa tiene formato tipo `GeoTIFF` (soportado por GDAL), las líneas 5 y de la 14 a la 23 nos da información sobre la proyección, así como el código EPSG correspondiente al modelo elipsoidal, datum y proyección (este último será el global para todo el mapa). Es claro ver que la proyección es *Transversal Universal de Mercator* (UTM) en zona 13 norte, la cual, como hemos visto es una buena proyección conformal que dada la zona tiene pocas distorsiones y conserva bien las distancias. Para efectos didácticos haremos dos transformaciones de proyección. Una será convertir el mapa de *densidad poblacional* que hemos venido utilizando. Este mapa originalmente lo bajamos sin proyección, es decir, sólo está definido con un modelo elipsoidal tipo WGS84, sus unidades son en segmentos de arco y por tanto no podemos efectuar mediciones. Lo transformaremos a una proyección UTM Zona 14 Norte para poder sobrelapar (en forma de capas) el ráster del Volcán de Colima, que es UTM 13 N. La otra transformación que haremos será reproyectar este mapa (el del Volcán) en UTM 14 N para que quede acorde con el mapa de *densidad poblacional*. Como hemos visto GDAL se encarga de procesar sólo los datos ráster por lo que no podremos hacer la reproyección del mapa de *densidad poblacional* con este comando pues el mapa es vectorial (para esto utilizaremos OGR (ver pág 71) pero si podremos transformar el ráster del Volcán de Colima. Esta transformación se puede realizar con el comando `gdalwarp` y tiene varias opciones disponibles así como distintos métodos de interpolación (como es ráster, la transformación no es uno y sólo a uno, hay pérdida de información que se tiene que "rellenar" adecuadamente) que pueden ser: *lineal, bilineal y cúbica*. Para transformarla a UTM 14N lo haremos con las opciones *de facto*. [16] La forma de utilización es por línea de comandos, ésta y el mensaje exitoso de la transformación se muestran a continuación.

```
juan@consola:/home/GIS/tutorial$ gdalwarp -t_srs EPSG:32614
VolcanColima.tif VolcanColimaUTM14.tif
Creating output file that is 7991P x 7615L.
Processing input file VolcanColima.tif.
0...10...20...30...40...50...60...70...
80...90...100 - done.
```

Hay dos puntos importantes que mencionar: *i)* Primero se pone el archivo a procesar ( VolcanColima.tif ) y después el archivo transformado ( VolcanColimaUTM14.tif ). *ii)* la opción `-t_srs EPSG:32614` quiere decir que el sistema de coordenas del archivo transformado (*target)* será el equivalente al código EPSG 32614, que es el equivalente al *datum WGS84* con proyección UTM zona 14 norte ( ver página 2.2.2 ). Si se prefiere utilizar el formato *proj* en vez del código EPSG éste sería:

[16] Si se quiere profundizar en otras formas de transformación véase el manual del mismo (`$ man gdalwarp`)

```
juan@consola:/home/GIS/tutorial$gdalwarp -t_srs " +proj=utm
+zone=14 +ellps=WGS84 +datum=WGS84 +units=m +no_defs"
VolcanColima.tif VolcanColimaUTM14.tif
Creating output file that is 7991P x 7615L.
Processing input file VolcanColima.tif.
0...10...20...30...40...50...60...70...
80...90...100 - done.
```

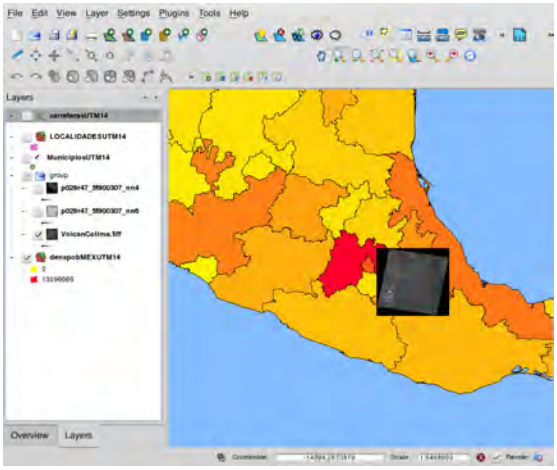
Una vez finalizado el cálculo se puede cargar el nuevo archivo (VolcanColimaUTM14.tif) en cualquier SIG que soporte formato GeoTiff; como QGIS o GRASS. Al haber hecho la reproyección, el archivo original en UTM 13 N sigue existiendo. Suponiendo que tenemos el mapa de *densidad de población* ya transformado a UTM 14 N podemos hacer el sobrelapamiento de los dos rásters (UTM 13 N y UTM 14 N) del área de Colima para ver que efectivamente la transformación es adecuada. La figura 3.10 muestra que en *(a)* el sobrelapamiento del ráster con proyección UTM 13 N sobre el mapa de *densidad de población* (en UTM 14 N) no está bien pues el ráster de Colima está entre Puebla, Tlaxcala y Morelos. Por otro lado, haciendo la reproyección adecuada (i.e. a la misma del mapa base) como en 3.10b tenemos un sobrelapamiento perfecto del ráster de Colima con el vectorial de *densidad de población* en UTM 14 N. Para hacerlo más explícito, nótese la línea costera de ambos, ¡ es prácticamente la misma!
La biblioteca GDAL tiene muchos comandos para manipulación de archivos ráster. Entre los más útiles están:

`gdal_translate` Si se quiere pasar a otro formato ráster, extraer una parte de alguno (*clipping*) o darle valores de transparencia a los valores nulos.

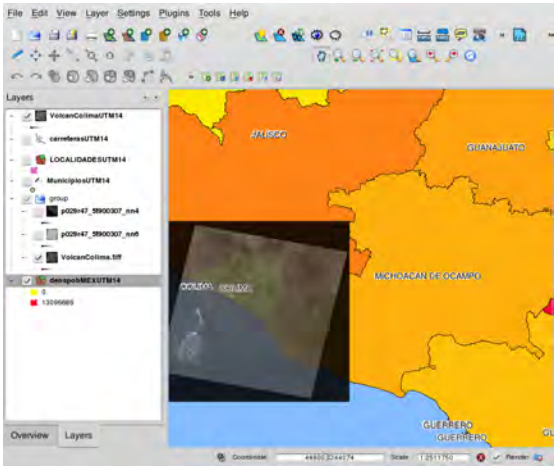
`gdal_contour` Crea un ráster de curvas de nivel a partir de un ráster de elevación digital.

`gdal_grid` Crea un rejilla (gradilla) que representa las líneas de meridianos y paralelos.

`gdal_rasterize` Convierte archivos vectoriales a ráster.




(a) Sobrelapamiento desfasado de la imagen con proyección UTM 13N sobre el mapa en UTM 14N.




(b) Sobrelapamiento con trasformación de proyección a UTM 14N, la misma que el mapa de densidad poblacional.

Figura 3.10: Comparación del sobrelapamiento de la imagen *Landsat* del estado de Colima con dos proyecciones; en *(a)* UTM 13 N y en *(b)* UTM 14 N. El mapa base es el de *densidad poblacional* de la Comisión Nacional para el Conocimiento y Uso de la Biodiversidad (CONABIO) (2006) con proyección UTM 14 N. Ráster cortesía de Rocío Alanís Anaya, 2010.

### OGR

OGR es el *compadre* de GDAL y forma parte de su biblioteca. Se encarga de manipular datos vectoriales (Warmerdam, 1998-2010). Como su compañero, OGR soporta varios formatos diferentes; más de 30. En algunos casos OGR puede crear capas vectoriales nuevas y en otros sólo puede leerlos. Los archivos más populares de datos vectoriales pueden ser procesados por OGR. Algunos de éstos, según Sherman (2008), son:

- Arc/Info Binary Coverage
- Comma-Separated Value (.csv)
- DWG
- DXF
- ESRI Personal GeoDatabase
- ESRI ArcSDE
- ESRI Shapefile (.shp)
- GML
- Generic Mapping Tools; GMT
- GPX
- GRASS
- KML
- Mapinfo File
- MySQL
- ODBC
- Oracle Spatial
- PostgreSQL
- SDTS
- SQLite
- TIGER/Line
- VRT - Virtual Datasource
- Informix DataBlade



La lista de todos los formatos compatibles está disponible en la página del proyecto:
`http://www.gdal.org/`

**Ejemplos OGR | Reproyecciones vectoriales** En la figura 3.10 está empalmada la imagen satelital con un mapa vectorial UTM zona 14 N de *densidad poblacional*, el cual hemos venido trabajando. No hay que olvidar que este mapa lo bajamos del portal de información geográfica de CONABIO en coordenadas *WGS84* sin proyección. Si utilizamos la herramienta de medición de QGIS en este mapa nos daremos cuenta que sus unidades están en grados. Para los siguientes análisis será necesario tener todos nuestros datos con un modelo de proyección pues vamos a tener necesidad de hacer mediciones sobre la superficie en metros. Como ya convertimos la imagen ráster a UTM 14 N, convertamos también el mapa de CONABIO a UTM 14 N. Al igual que los *rásters* los datos vectoriales también tienen sus respectivos *metadatos*. La herramienta análoga a `gdalinfo` es, adivinaron, `ogrinfo`. Una diferencia es que por *default* da poca información del archivo. Si le pedimos que nos dé toda la información con el parámetro `-al` nos dará literalmente **TODA LA INFORMACIÓN** del mapa incluyendo todas las coordenadas de los puntos, arcos, líneas y polígonos.

Un comando que arroja menos información pero suficientemente útil será:

```
$ ogrinfo -so -al [ARCHIVO]
```

Vayamos al caso concreto del mapa de *densidad de población* de la CONABIO. La sintaxis del comando y su salida en la consola sería de la siguiente forma:

```
juan@consola:/home/GIS/tutorial$ ogrinfo -so -al
denedo2kgw.shp
1  INFO: Open of 'denedo2kgw.shp'
2        using driver 'ESRI Shapefile' successful.
3
4  Layer name: denedo2kgw
5  Geometry: Polygon
6  Feature Count: 382
7  Extent: (-118.366027, 14.534005)
8          - (-86.710744, 32.718767)
9  Layer SRS WKT:
10 GEOGCS["GCS_WGS_1984",
11     DATUM["WGS_1984",
12         SPHEROID["WGS_1984",6378137.0,298.257223563]],
13     PRIMEM["Greenwich",0.0],
14     UNIT["Degree",0.0174532925199433]]
15 AREA: Real (20.5)
16 PERIMETER: Real (20.5)
17 COV_: Real (11.0)
18 COV_ID: Real (11.0)
19 NUM_EDO: Real (11.0)
20 ENTIDAD: String (100.0)
21 CAPITAL: String (100.0)
22 PO_TO_2000: Real (20.0)
23 SUPER_KM2_: Real (20.0)
24 DE_PO_2000: Real (20.0)
```

Con esto vemos que el mapa no está proyectado, sólo tiene definido un modelo elipsoidal, por tanto sus unidades son en grados, como lo indica la línea 14. La transformación es de forma similar al modelo ráster, la diferencia está claramente en el algoritmo. Mientras que en el ráster se trata de hacer una interpolación para la transformación de los valores de las celdas, lo que conlleva pérdida de información, en el modelo vectorial sólo se hace una trasformación lineal adecuada, esto es, se multiplica cada vector por una matriz de transformación de 2x2. En este caso la transformación sí es invertible y por tanto no hay pérdida de información, i.e. se pueden hacer tantas transformaciones como se quieran. El comando que transforma proyecciones y además cambia de un formato vectorial a otro es `ogr2ogr` y la sintaxis es como se muestra acontinuación.

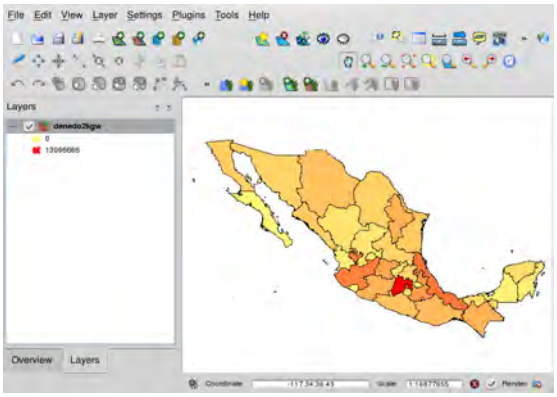
```
$ ogr2ogr -f ''Formato soportado'' -s_srs
''TIPO_COORD'' -t_srs ''TIPO_COORD''
Archivo_Salida.ext Archivo_Entrada.ext
```

Utilicemos este comando para reproyectar nuestro mapa vectorial. La ejecución de la transformación así como su correspondiente salida de información con `ogrinfo` sería:

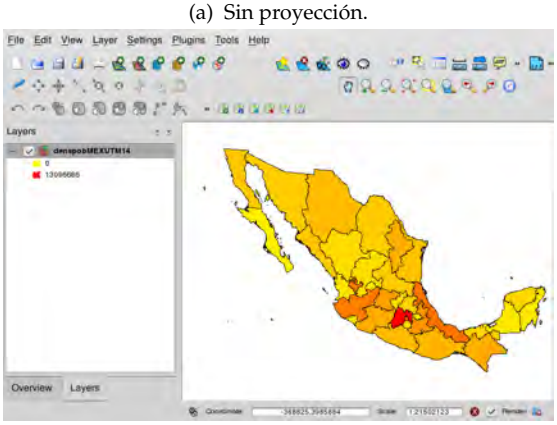
```
juan@consola:/home/GIS/tutorial$
$ogr2ogr -f ``ESRI Shapefile'' -t_srs EPSG:32614
denedo2kgw.shp denspobUTM14.shp
$ogrinfo -so -al denspobUTM14.shp
1   INFO: Open of `denedoUTM14.shp'
2       using driver `ESRI Shapefile' successful.
3
4   Layer name: denedoUTM14
5   Geometry: Polygon
6   Feature Count: 382
7   Extent: (-1403680.640250, 1617663.740535)
8            - (1786784.007451, 3748495.203533)
9   Layer SRS WKT:
10  PROJCS["WGS_1984_UTM_Zone_14N",
11      GEOGCS["GCS_WGS_1984",
12          DATUM["WGS_1984",
13              SPHEROID["WGS_1984",6378137
14                       ,298.257223563]],
15          PRIMEM["Greenwich",0],
16          UNIT["Degree",0.0174532925199433295]],
17      PROJECTION["Transverse_Mercator"],
18      PARAMETER["latitude_of_origin",0],
19      PARAMETER["central_meridian",-99],
20      PARAMETER["scale_factor",0.9996],
21      PARAMETER["false_easting",500000],
22      PARAMETER["false_northing",0],
23      UNIT["Meter",1]]
24  AREA: Real (20.5)
25  PERIMETER: Real (20.5)
26  COV_: Real (11.0)
27  COV_ID: Real (11.0)
28  NUM_EDO: Real (11.0)
29  ENTIDAD: String (100.0)
30  CAPITAL: String (100.0)
31  PO_TO_2000: Real (20.0)
32  SUPER_KM2_: Real (20.0)
33  DE_PO_2000: Real (20.0)
```

Observemos que las líneas de arriba que no están numeradas corresponde a la orden de reproyección, cada orden empieza con el signo $ que representa el cursor (*prompt*) de la consola[17]. La siguiente orden (`ogrinfo -so -al denspobUTM14.shp`) es para que nos muestre la información (metadatos) del nuevo mapa, producto de la reproyección a UTM 14 N. Como vemos, esta información sale en pantalla y en este cuadro se representa por todas las líneas numeradas. Vemos pues que de la línea 17 a la 23 este nuevo archivo ya tiene definidos parámetros de proyección y unidades en UTM 14 N y metros (ver línea 23) respectivamente. Para comparar geométricamente dicha transformación vease la figura 3.11

[17] Claramente no se tiene que escribir ese signo.




(a) Sin proyección.




(b) Proyección UTM zona 14 norte.

Figura 3.11: Comparación del mapa de densidad poblacional de la Comisión Nacional para el Conocimiento y Uso de la Biodiversidad (CONABIO) (2006) sin proyectar (en a) y proyectado en UTM 14N en (b). Nótese la deformación a medida que se aleja del Trópico de Cancer. En b el área es más angosta y deforme hacia el norte.

**Otro problema con INEGI** Este instituto tiene a disposición pública datos vectoriales que pueden ser descargados gratuitamente de su portal. Un ejemplo es el *Polígono de localidades urbanas geoestadísticas* (INEGI, 2009), disponible en:

`http://mapserver.inegi.org.mx/data/mgm/`

Al descargar el archivo y revisar sus metadatos con `ogrinfo` nos damos cuenta que no tiene asignada información de *datum*, coordenadas, modelo elipsoidal, unidades y proyección. es decir, carece completamente de referencia geográfica. Al regresar al sitio INEGI señala, de forma informal pues no esta incorporada en los datos, que el *datum* utilizado es el ITRF92 y tiene una proyección geográfica Cónica Conforme de Lambert. El *datum* no es problema pues ya hemos visto que para resoluciones mayores a 10 cm es equivalente al WGS84. Pero si es problema la proyección ya que aunque sabemos qué tipo de proyección es, carecemos de los paralelos estándar necesarios, además

no hay código EPSG con esa proyección para alguna región de México[18]. Tenemos dos opciones. *i)* determinar los paralelos nosotros mismos o *ii)* ir al sitio *SpatialReference.org* [19] que tiene este tipo de información. No sobra decir que es otro de tantos proyectos abiertos y colaborativos en los que gente de todo el mundo puede participar subiendo y bajando información de sistemas de referencia geográfica en varios formatos. En ese espíritu de colaboración alguien subió la información apropiada referente a la proyección Cónica Conforme de Lambert para México ; pero para *datum NAD83*. Como nosotros estamos trabajando con un datum WGS84 será necesario hacer algunas modificaciones a los parametros de proyección, de hecho es una modificación muy pequeña, donde diga NAD83 pondremos WGS84. Por ejemplo si utilizamos el estándar *ProJ4* las instrucciones que necesitamos serán las siguientes:

```
+proj=lcc +lat_1=17.5 +lat_2=29.5 +lat_0=12
+lon_0=-102 +x_0=2500000 +y_0=0 +ellps=WGS84
+units=m +no_defs
```

Con esta información ya podemos reproyectar el mapa de polígonos urbanos con el comando `ogr2ogr`, recordemos que si vamos a introducir los parámetros de proyección en formato *Proj.4* debemos encerrar todo ese texto entre comillas simples o dobles. Otra forma de hacerlo es mediante un archivo de texto en el cual guardemos sólo ese texto. Nombramos de alguna manera ese archivo y cuando corramos el comando `ogr2ogr` le decimos que tome los parámetros de proyección de ese archivo.

### Instalación de GDAL/OGR

Hay varias formas de instalar las bibliotecas GDAL/OGR. Estas dependerán de nuestras necesidades, nuestro equipo y nuestro sistema operativo. En el mejor de los casos (y el más probable, supongo) tenemos una computadora con procesador *Intel o AMD* y contamos con una distribución Linux popular Ubuntu, Fedora, RedHat, Debian,etc. Una forma fácil de instalar esta biblioteca es usando paquetes precompilados. El gestor de paquetes de nuestra distribución nos ahorrará mucho tiempo ( y probablemente muchas maldiciones). Si estamos usando Ubuntu o Debian bastará con buscar en los repositorios con el gestor *Synaptic* o vía línea de comandos con la herramienta `apt`. A continuación se muestra un ejemplo de busqueda del paquete GDAL en los repositorios de **Ubuntu** con la ayuda del comando `apt-cache search`. La lista resultante son todos los paquetes relacionados a GDAL que podemos instalar en nuestra PC con el comando `apt-get install [PAQUETE]` siendo super usuario o con el prefijo `sudo`.

[18] ¿Qué le pasa a esa gente de INEGI?
[19] `http://spatialreference.org/`

```
juan@consola:/home/GIS/tutorial$ sudo apt-cache search gdal
gdal-bin - Geospatial Data Abstraction Library - Utility programs
libgdal-doc - Documentation for the Geospatial Data Abstraction Library
libgdal-perl - Perl bindings to the Geospatial Data Abstraction Library
libgdal-ruby - Ruby bindings to the Geospatial Data Abstraction Library
libgdal-ruby1.8 - Ruby 1.8 bindings to the Geospatial Data Abstraction Library
libgdal1-1.4.0 - Geospatial Data Abstraction Library
libgdal1-1.4.0-grass - GRASS extension for the Geospatial Data Abstraction Library
libgdal1-dev - Geospatial Data Abstraction Library - Development files
python-gdal - Python bindings to the Geospatial Data Abstraction Library
```

Si se va a optar por esta forma de instalación, conviene instalar [20] los paquetes marcados con **negritas**. Otra opción es continuar la lectura o, por qué no, saltarse a la página 78 donde se explica una forma más fácil y rápida de instalar todos estos paquetes si y sólo si se tiene instalada una distribución basada en Ubuntu.

#### FWTools

Otra forma de instalar esta biblioteca es a través del proyecto **FWTools** [21]. Este proyecto es un conjunto de paquetes de código abierto que está diseñado especialmente para usuarios GIS del mundo *Open Source*. En palabras de su fundador *Frank Warmerdam* este *kit* esta pensado para el usuario final. Son paquetes precompilados, esto es que no tenemos que compilar de la fuente, crear dependencias y todo ese proceso engorroso. Sólo lo bajamos, instalamos y usamos. FWTools incluye las siguientes aplicaciones:

[20] e.g. `sudo apt-get install libgdal1-1.4.0-grass`
[21] `http://fwtools.maptools.org/`

### Paquetes incluidos en FWTools

- GDAL: Biblioteca para manipular datos vectoriales y ráster.
- OpenEV: Herramienta de visualización de datos ráster y vectoriales.
- PROJ.4: Biblioteca de proyecciones cartográficas con utilidades para linea de comandos.
- OGDI: Biblioteca multiformato para lectura de datos ráster y vectoriales con soporte para varios formatos militares como VPF (ie. VMAP, VITD), RPF (ie. CADRG, CIB), y ADRG.
- Mapserver: Paquete para desarrollo de GIS con interfaz web.
- Python: Lenguaje de programación tipo intérprete para generar scripts.

Puede ser el caso que algunas de las aplicaciones de FWTools ya esten instaladas en nuestra PC, de facto, Python. El proceso de instalación de FWTools no interfiere con paquetes ya instalados o por instalarse via `apt`. Esto es porque FWTools se instala de forma local en nuestro directorio de trabajo y por tal razón no se necesitan privilegios de super usuario. Si trabajamos en una computadora donde no somos los administradores, está es una buena opción.

#### Instalación de FWTools

De la página `http://fwtools.maptools.org/` bajamos el paquete correspondiente a nuestro sistema operativo. En el caso de usuarios Linux damos click en `FWTools 2.0.6 (Linux x86 32bit)`[22]. El número de versión (en este caso 2.0.6) puede cambiar debido a nuevas actualizaciones que hayan hecho los desarrolladores de FWTools.

Una vez bajado el paquete (aproximadamente 27MB) procederemos a su instalación. En virtud de recordar y fomentar el uso del *shell* efectuaremos la instalación en la línea de comandos[23]. Supongamos que el archivo bajado se encuentra en la carpeta *Descargas*. Es recomendable hacer una carpeta llamada fwtools (por ejemplo) en `home/usuario/` con el comando `mkdir fwtools`. Copiamos el archivo de la carpeta *Descargas* al directorio *fwtools* con el comando `cp Descargas/FWTools-linux-2.0.6.tar.gz fwtools/`. Lo descomprimimos con `tar -zxvf FWTools-linux-2.0.6.tar.gz` (dentro de la carpeta fwtools, obviamente). Inspeccionamos el contenido con el comando `ls` y buscamos que el archivo `install.sh` esté. Como es un script de shell lo ejecutamos con `sh install.sh`. Al terminar, el paquete estará instalado. Falta redefinir la ruta de los ejecutables para que podamos utilizarlos en cualquier directorio. Para esto creamos una variable de entorno con el comando: `PATH=$PATH:$HOME/FWTools.VERSION/bin_safe`.

[22] Esta versión también soporta arquitecturas de 64 bits

[23] De hecho no hay otra forma de instalar FWTools mas que ésta

Este último paso habrá que hacerse cada vez que se reinicie la computadora. Si se quiere dejarlo definido siempre, se tendrá que añadir este comando en el script de inicio ( /.bash_profile) este archivo se puede modificar con cualquier editor de texto.

A continuación se presenta un *facsímil* de la instalación vía línea de comandos.

Instalación de **FWTools**

```
 juan@Jaguar:~/Descargas$ ls
FWTools-linux-2.0.6.tar.gz

juan@Jaguar:~/Descargas$ cp FWTools-linux-2.0.6.tar.gz ../
juan@Jaguar:~/$ tar -zxvf FWTools-linux-2.0.6.tar.gz

juan@Jaguar:~/$ cd FWTools-linux-2.0.6

juan@Jaguar:~/FWTools-2.0.6$ ls
bin         html     install.sh  pics   README.1ST
conf        include  lib         pymod  sbin
demo-data   info     man         ramps  share
symbols    xmlconfig tools       wms
juan@Jaguar:~/FWTools-2.0.6$ sh install.sh
...
Compiling lib/python2.2/rfc822.py ...
Compiling lib/python2.2/rlcompleter.py ...
Compiling lib/python2.2/robotparser.py ...
Compiling lib/python2.2/sched.py ...
Compiling lib/python2.2/sgmllib.py ...
...

juan@Jaguar:~/FWTools-2.0.6$ PATH=$PATH:/home/juan/
FWTools-2.0.6/bin_safe
```

### 3.2.2. UbuntuGIS al rescate

Para usuarios de Debian o Ubuntu existe otra forma, sencilla y completa de instalar la mayoría de las herramientas del *GeoKit*, Usando la plataforma *launchpad.net* creada por los desarrolladores de Ubuntu. Los desarrolladores del proyecto *UbuntuGIS* se han dado a la tarea de compilar y empaquetar las versiones más nuevas de estos programas y están disponibles para nosotros en el sitio:`https://launchpad.net/ ubuntugis/+archive/ubuntugis-unstable` sólo tenemos que añadir los repositorios a nuestra base de datos y registrar la llave pública y podremos instalar via `apt` o *synaptic* los paquetes que necesitemos. Para esto copiamos las direcciones marcadas en el recuadro inferior en la lista de repositorios ubicada en */etc/apt/sources.list*. Podemos utilizar el editor de texto que queramos. Sólo hay que recordar que podremos modificar la lista de repositorios sólo si estamos en modo de super usuario es decir, si somos los administradores del equipo donde vamos a instalar el software (o si conseguimos la contraseña de algún modo).

- deb http://ppa.launchpad.net/ubuntugis/ubuntugis-unstable/ubuntu hardy main
- deb-src http://ppa.launchpad.net/ubuntugis/ubuntugis-unstable/ubuntu hardy main

Observaciones: al final de las dos direcciones vemos las frases `ubuntu hardy main` la palabra **hardy** hace referencia a la versión de ubuntu que estoy usando (8.04). Actualmente está disponible la versión (10.04) con soporte de tres años, esto significa que los desarrolladores de ubuntu se comprometen a mantener esta versión por tres años. El nombre se ésta se llama *lucid*. Es importante que se le cambie el nombre `hardy` por aquel que le corresponda a la versión de ubuntu donde se quiera instalar este software puede ser *intrepid, jaunty, karmic, lucid* o próximamente *maverick*. El archivo donde se deben agregar estas direcciones se llama *sources.list* y se encuentra en la carpeta `/etc/apt/`. Al guardar el archivo modificado habrá que actualizar la base de datos de los programas con el comando `sudo apt-get update`. Concluido este paso sólo falta registrar la llave pública de los nuevos repositorios. Para escribimos (en el *shell* como lo hemos hecho hasta ahora):

```
sudo apt-key adv --keyserver keyserver.ubuntu.com
                 --recv-keys 12345678
```

Tendremos que reemplazar la clave 12345678 por la escrita en la página de ubuntugis en *launchpad.net*[24]. Confiando en que no cambiará en el futuro sería:314DF160. Actualizando los repositorios, quedarán dados de alta y la instalación del *GeoKit* es pan comido, se puede usar *synaptic* o en el *shell* `apt-get install qgis grass libgdal libgdal-mrsid-src libgdal-ecw-src`.

De esta forma se tiene también la versión de QGIS y GRASS más recientes y las compilaciones de GDAL/OGR para fomatos restringidos como MrSID y ERDAS .

**Nueva forma de instalación para usuarios de Ubuntu 9.10 o superior**

Esta forma de instalación es más sencilla que la descrita anteriormente. De hecho sólo involucra abrir una terminal y escribir:

```
sudo add-apt-repository ppa:ubuntugis/ubuntugis-unstable
```

Una vez ejecutada esta orden, ubuntu bajará automáticamente los repositorios adecuados y la llave de auteticación. Bastará actualizar los repositorios con `sudo apt-get update` y listo, podremos instalar los programas del *GeoKit* más recientes con ayuda del *synaptic* o `apt-get install [PAQUETE]`.

[24] la llave son los ocho caracteres después de 1024R

# Capítulo 4

# Geoprocesamiento básico con GRASS

En este capítulo realizaremos dos actividades que servirán para ejemplificar la potencialidad de los SIG, en particular GRASS, para crear mapas temáticos y hacer análisis espacial a partir de los datos con los que hemos trabajado y otros nuevos. Este proceso, al tener información espacial, recibe el nombre de geoprocesamiento. La existencia, diversificación y éxito de los SIG está en la facultad de procesar, análizar y crear datos nuevos para comprender y responder todo tipo de preguntas relacionadas con el espacio. En la práctica, tomar mejores decisiones para problemas de cambio climático, conservación, uso de suelo, ordenamiento territorial, etcétera.

## 4.1. Ejercicio: *El mapa de Dificultad para conservar*

Esta sección muestra cómo hacer un mapa temático que nos mostrará con un gradiente de color la dificultad para conservar las *áreas prioritarias para la conservación* terrestre publicadas por la Comisión Nacional para el Conocimiento y Uso de la Biodiversidad (CONABIO) et al. (2008). La forma de proceder será la siguiente:

1. Crear un nuevo mapa vectorial en el cual se conviertan los puntos del censo en polígonos de área. A esta acción se le llama *buffering* .
2. *Fusionar* las áreas adecuadamente si hay traslapes.
3. Crear un mapa *ráster* producto de la transformación del mapa vectorial de *sitios prioritarios*.
4. Crear otro *ráster*, esta vez el transformado de las áreas del censo hecho en el paso 1.
5. Utilizar *álgebra de mapas* para calcular el gradiente de *dificultad para conservar*.

Esto ha sido un mero esbozo de lo que será la realización del mapa de *dificultad*, veremos a lo largo de esta sección que nos encontraremos con algunos detalles que iremos resolviendo poco a poco.

### 4.1.1. Materiales

Los datos requeridos para esta actividad son:

- Mapa de sitios terrestres prioritarios para la conservación con elipsoide WGS84. El mapa puede ser bajado del sitio:

  `http://www.conabio.gob.mx/informacion/gis/`

- Datos de localidades georreferenciadas del *XII Censo de Población y Vivienda, 2000; (INEGI, 2005)*, convertido en formato decimal, ver apéndice 3.

- Recomendable: *Geokit de herramientas* (pág 64) .
  Indispensable: GRASS y biblioteca GDAL/OGR.

### 4.1.2. Preparativos

Tenemos que organizar los datos antes para poder hacer análisis. Lo primero que haremos será preparar el *entorno de trabajo* de GRASS. Para esto, previa instalación de él (ver páginas 63 y 78), ejecutaremos el programa en la consola, dependiendo la versión instalada puede ser uno u otro.

```
juan@consola:/home/GIS/tutorial$  grass (grass64)
```

Al iniciar aparecerá un diálogo como el de la figura 4.1.

Habíamos comentado que GRASS necesita un espacio de trabajo definido previamente llamado **localidad** (*location*). Al definir esto, todos nuestros datos estarán desde el inicio bajo una proyección arbitraria. Como GRASS tiene su propio formato de datos será necesario importar los mapas desde él para que los convierta al formato nativo.

#### Crear localidades

GRASS tiene tres formas de crear localidades: *i)* Por medio de un archivo georreferenciado. *ii)* Definiendo las coordenadas extremas del norte, sur, este y oeste; con su correspondiente datum y parámetros de proyección. *iii)* Con el código EPSG de una zona en particular; como podría ser una zona UTM con datum WGS84 o NAD27. Una forma alternativa es mediante el plugin de GRASS, con QGIS. Este módulo incorpora muchas herramientas de GRASS dentro de QGIS, incluso para crear localidades; se define dibujando un cuadro en algún lugar de interés de un mapamundi con una herramienta sencilla.

El índice de *dificultad* lo calcularemos para todas las *zonas prioritarias para la conservación* del país, por tanto necesitamos definir una localidad que contenga todo el territorio nacional. Podemos utilizar el mapa *Shapefile* de densidad poblacional de CONABIO del año 2006 [1] para crear la localidad con la forma *(i)* descrita anteriormente. Para esto debemos dar click en el botón `Archivo georreferenciado` en el cuadro de inicio como el de la figura 4.1

[1] Recordemos que cuando bajamos este mapa lo hicimos con un datum WGS84 y sin proyección.


Figura 4.1: Menú de inicio de GRASS. En el cuadro rojo se muestran las tres formas de crear localidades y en el azul el botón para crear el *mapset* de usuarios.

Aparecerá un cuadro con dos campos: uno para darle nombre a la localidad y otro para darle la ruta del directorio donde se va a guardar la información. Al dar aceptar, GRASS creará automáticamente toda la estructura de directorios que vimos en el capítulo anterior (ver figura 3.9). Por omisión sólo tendrá el *mapset* `Permanent`, por tanto después de crear la localidad tenemos que crear un nuevo directorio de mapas para trabajar más ordenadamente. Esto se puede hacer con el botón: `Crear directorio de mapas`, como se puede ver en la figura 4.1 en el cuadro azul. Aquí podemos añadir directorios de trabajo asignados para cada usuario o proceso de elaboración de algún proyecto.

**Consideraciones:** Hemos comentado que GRASS utiliza su propio formato de datos, a diferencia de QGIS, el proceso para cargar información requiere previamente de un proceso de conversión al formato de GRASS. Este proceso, dependiendo de los datos, puede llegar a ser tardado. La ventaja principal en esto es que, al definir una localidad, si los datos para trabajar están proyectados en otro sistema o tienen otro datum distinto al definido en la localidad, GRASS automáticamente los transforma adecuadamente para que todos estén con las mismas coordenadas; obteniendo un empalme perfecto y contínuo. Estos nuevos datos, en formato GRASS, quedarán guardados en el *mapset* (directorio) donde se importaron previamente. Además todos los datos creados e importados se pueden exportar a otros formatos, como lo veremos más adelante.

**Interfaz gráfica con aprendizaje de comandos** En GRASS se puede trabajar de dos formas: con una interfaz gráfica o en la consola con línea de comandos. Esta última es la más rápida y eficiente pues se pueden crear prequeños *scripts* (programitas) para automatizar los análisis. La desventaja de esto es que tenemos que aprendernos los comandos y sus respectivas formas de utilización (sintaxis, parámetros, etc) y esto nos puede llevar más tiempo. Para minimizar esta curva de aprendizaje, los desarrolladores de GRASS han incluido esta información en la interfaz gráfica. De hecho, internamente, la interfaz gráfica (*gui*[2]) es un programa independiente del módulo a ejecutar, ésta sólo le pasa al intérprete de comandos la cadena de caracteres referente al comando y los parametros necesarios para hacer algún proceso.

**Funcionamiento de la interfaz gráfica (*gui*)** Dependiendo de la versión de GRASS es posible encontrar dos *versiones* de *gui*: la clásica (ver figura 4.2a) llamada `tclck` y la moderna llamada `wx`, basada en `wxpython`, una biblioteca de *python* para desarrollar interfaces gráficas como ventanas, cuadros de diálogos, etc. (ver figura 4.2b). Por lo general la instalación básica de GRASS tiene *defacto* la interfaz clásica. Para fines prácticos ambas sirven de igual forma, la versión `wx` tiene mayor trabajo estético y por tanto es más amigable. Particularmente, en este capítulo se utiliza esta interfaz para ejemplificar las actividades por lo que será común ver capturas de pantalla con esta interfaz.

Como podemos ver en las figuras (4.2) hay dos ventanas comunes en las dos *gui*. Estas son: la ventana de **Administración** y el **Monitor de mapas**.

- La ventana de Administración tiene cuatro secciones importante:

  **Barra de menús:** Nos servirá para buscar, configurar y ejecutar la mayoría ( o al menos las más útiles) herramientas de GRASS. Aquí podremos cargar mapas (ráster o vectoriales), importarlos, procesarlos, exportarlos, visualizarlos, conectar, administrar y crear bases de datos, hacer análisis estadísticos, etcétera. Hay casi tantas opciones como comandos y módulos se tengan instalados.

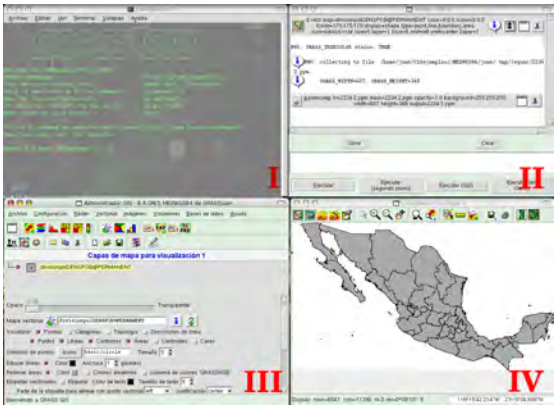
  **Barra de herramientas:** Aquí tendremos íconos representativos de los comandos más utilizados como abrir mapas ráster o vectoriales, guardar y borrar capas, visualizar gradillas, etcétera.

  **Menú de capas:** En esta sección se administran las capas que tenemos cargadas en el proyecto. Podemos visualizar u ocultar cualquiera que esté cargada así como editar sus parámetros de visualización.

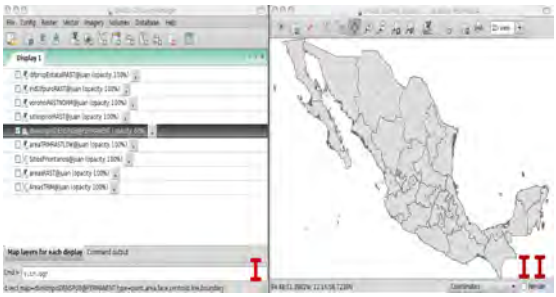
  **Menú de propiedades de capa:** Esta sección se comunica con el *monitor de mapas* mandándole los parámetros que seleccionemos.

- El *Monitor de mapas* nos servirá para visualizar los datos. Podemos hacer acercamientos, alejarnos, mover el mapa, buscar y mostrar atributos en la base de datos asociada, etc. La versión `wx` tiene más opciones que la *clásica*.

[2]Del acrónimo *graphical user interface*




(a) Interfaz gráfica clásica de GRASS. ( Invocación en consola: `g.gui gui=tclck`). I) Consola, II) Ventana de procesos, III) Administrador de capas y menús, IV)Monitor de mapas.




(b) Nueva interfaz gráfica de GRASS basada en *wxpython*. (Invocación en consola: `g.gui gui=wx`. I) Administrador multipropósito, II) Monitor de mapas.

Figura 4.2: Distintas *gui* de GRASS

**Ejecutando comandos** La parte central de GRASS es su capacidad de cómputo y análisis. Al ser modular, podemos ejecutar comandos en la terminal o con una *gui*. Si lo hacemos por línea de comandos e introducimos los parámetros adecuados, GRASS procesará todo y nos devolverá el informe de lo realizado en la consola. Por otro lado, estando en la consola, si escribimos el comando sin parámetros, se abrirá una ventana donde aparecerá toda la información referente al módulo (comando) llamado, a esta acción se le llama *invocación*. De esta forma podremos, sin necesidad de aprendernos la sintaxis del mismo, introducir toda la información necesaria para que el comando realice su tarea. Se ha añadido en la parte inferior de esta ventana, el nombre del comando así como los respectivos parámetros para la acción definida por nosotros en la ventana. De esta forma podemos ver como hacer la misma acción sin tener que abrir la ventana del comando correspondiente. La figura 4.3 muestra la ventana correspondiente al comando `v.in.ogr` que sirve para importar mapas vectoriales soportados por la biblioteca GDAL/OGR. En este caso se utiliza un archivo en formato *Shapefile*.

Si no utilizamos la consola para ejecutar tareas, si no que usamos únicamen-


(a) Ventana con *gui* clásica. (b) Ventana con *gui* `wx`.

Figura 4.3: Ventana del comando `v.in.ogr`. Obsérvese que en ambas, en la parte inferior, aparece una cadena de caracteres que corresponde al comando utilizado con los parámetros necesarios para cargar, procesar y crear los mapas sin necesidad de abrir la ventana.

te el *Administrador*, la información de los procesos realizados aparecerán en una *ventana de procesos* en el caso de *gui clásica* (fig: 4.2a) y en la ventana del *Administrador multipropósitos*, en la pestaña llamada: `Command output` (fig: 4.2b) en el caso *wx*. **A partir de este momento**, salvo caso explícito, utilizaremos el comando en su forma pura i.e. `comando + parámetros`. Considerando que el usuario puede también introducir los parámetros interactivamente por medio de las ventanas mencionadas anteriormente.

**Nombres y apellidos de los comandos.** Los comandos de GRASS tienen una letra al principio seguida de un punto y luego un nombre en particular. Esto es para tener clasificados los comandos y que sea más fácil utilizarlos. La primera letra hace referencia al nivel en el cual opera el comando. Por ejemplo si le vamos a aplicar el comando a un mapa vectorial, la primera letra del comando será `v` por vectorial; e.g. `v.in.ogr`. Si vamos a trabajar con ráster, entonces será `r`; e.g. `r.in.gdal`. Para bases de datos es `db`. Si vamos a mostrar la información en el monitor, todos los comandos relacionados a esta actividad empezarán con la letra `d` de *display*. Para los otros comandos que no involucran manipulación o visualización, pero sí configuraciones, ayuda e información general, empezarán con la letra `g` de *general*.

### Importar mapas

Los mapas vectoriales con los que hemos trabajado son de tipo *Shapefile* y *texto delimitado* para el caso del XII Censo de Población y Vivienda, 2000 (INEGI, 2005). El comando para importar los datos *.shp* como ya vimos es con `v.in.ogr`, pues esta biblioteca manipula datos en este formato[3] La figura 4.3 muestra la ventana de este comando.

Para los fines del ejercicio del *índice de dificultad* debemos incluir el *Shapefile* de densidad de población por estado (CONABIO, 2006), deberemos usar nuevamente este comando para importar ese mapa. La orden completa sería

[3]Otros formatos soportados por OGR en pág. (71).

la siguiente: [4]

```
v.in.ogr dsn=~/GISejemplos/denspob/denedo2kgw.shp
output=divisionpolDENSPOB min_area=0.0001
snap=-1
```

**Importar datos en texto simple.** Hemos visto como hacer esto en QGIS, en GRASS es algo distinto. Utilizaremos la herramienta: `v.in.ascii`, debemos definir manualmente los nombres de las columnas y debemos especificar también de qué tipo son es decir, *enteros (integer), flotantes (float), doble precisión (double precision) o cadena de caracteres (varchar(n), donde n es el tamaño máximo de la cadena)*. La orden en texto simple para los datos del censo de población de INEGI, 2005 corregidos previamente con el programa descrito en el *Apéndice 3* sería la siguiente:

```
1 v.in.ascii -z input="~/GISejemplos/Municipios
2 /salida2.csv"
3 output="localidadesPNT" format="point" fs="|" skip=1
4 columns= CVE_ENT int,NOM_ENT varchar(35),CVE_MUN int,
5 NOM_MUN varchar(50),CVE_LOC int,NOM_LOC varchar(80),
6 AMBITO varchar(10),LATITUD double precision,LONGITUD
7 double precision,ALTITUD int,CVE_CARTA varchar(10),
8 POB_TOT int,POB_MASC int,POB_FEM int,TOT_VIV_HAB int"
9 x=9 y=8 z=10 cat=0
```

**Descripción:** En los renglones 1 y 2 están definidos: la ruta del archivo (`input`), el nombre del archivo de salida (`output`), el formato de geometría (punto, línea, polígono), el delimitador (`fs`) y el salto entre cada delimitador para llegar a la siguiente variable (`skip`). Del renglón 4 al 8 se definen los nombres y tipos de las variables de los datos (`Columns` ) estos son, los nombres y tipos de los atributos. Notemos como los nombres en MAYÚSCULAS corresponden a los nombres de las variables asignadas por los autores del mapa. En el último renglón se especifica el número de la columna donde se encuentran los valores de *longitud (*`x`*), latitud (*`y`*) y altura (*`z`*)*.

Una vez finalizado el comando tendremos disponible el mapa de puntos en nuestro *mapset* podremos cargarlo en el *Administrador* desde la barra de herramientas o con el comando `d.vect`.

### 4.1.3. Cómo crear mapas a partir de subconjuntos de datos

Dado que los puntos en el mapa del censo poblacional por localidades son muchísmos, aproximadamente 250,000 datos, el tiempo de cómputo para hacer las tareas más sencillas se prolonga demasiado. Para evitar esto, lo primero que vamos a hacer es tomar aquellas localidades cuya población total sea mayor a 1000 habitantes. Con esto reduciremos significativamente los puntos, ya que hay muchas localidades con menos de 100 habitantes. Estamos suponiendo que

[4] Téngase en cuenta que el campo `dsn=` es donde debemos poner la ruta en la que se localiza el archivo. Esta ruta es personal y el símbolo ~ es una abreviatura del directorio de trabajo de la sesión en Linux. En mi caso ~ `:=/juan/GISejemplos/denspob/`

las localidades con menos de 1000 personas tienen un impacto no significativo en la conservación de áreas prioritarias para la conservación terrestre. Dada la escala que estamos trabajando, esta suposición parece razonable.

**v.extract:**

Esta herramienta nos servirá para este propósito, su funcionamiento, esencialmente, es buscar por medio del estándar SQL aquellos objetos (puntos, polígonos, líneas) que cumplan con lo especificado en el campo (con formato SQL) llamado `WHERE`. Una vez determinados estos objetos, GRASS creará un nuevo mapa (en formato GRASS) con los datos que cumplieron con la fórmula de la búsqueda, obteniendo así un mapa de un subconjuto del original con la característica de que estos nuevos datos cumplen con la propiedad arbitariamente especificada. Esta operación, junto con las de unión, intersección, resta y complemento de conjuntos son básicas para crear mapas nuevos[5]. Más aún, con éstas básta para crear una variedad muy grande de datos. Volviendo a nuestro problema, la busqueda es verdaderamente sencilla pués sólo queremos quedarnos con los datos cuyos valores en el campo `POB_TOT` sean mayores a 1000. La busqueda en el lenguaje SQL se incorpora al comando `v.extract` en la entrada `where=" "` la forma correcta de ejecutar toda esta acción es la siguiente:

```
 v.extract input="localidadesPNT@mapset"
output="muestraLOC" type="point,line,
boundary,centroid,area,face"
layer=1 where="POB_TOT>1000" new=-1
```

La figura 4.4 muestra una captura de la ventana correspondiente a este comando con los valores de busqueda requeridos. Para ver el nombre de las variables así como otra información importante se puede utilizar el comando:

```
v.info -c map=NombreMapa@mapset
```

El mapa resultante (`muestraLOC`) tiene muchos menos puntos que el original. De aquí en adelante trabajaremos con éste pues el procesamiento es más rápido.

### 4.1.4. Conversión de puntos y líneas a polígonos

La geometría de los datos de localidades del censo de INEGI, 2005 son puntos. Para poder hacer el *índice de dificultad* debemos transformar adecuadamente estos puntos en áreas y luego a éstas asignarles los valores poblacionales de los puntos de manera unívoca. Por tanto, lo que queremos hacer es subir de dimensión la geometría de estos puntos. Dicho de otra manera, A cada punto hacerle corresponder uno y sólo un polígono. Hay muchos algoritmos que pueden hacer esta transformación. Utilizaremos dos metodologias para contrastarlas y tomar una mejor decisión.

[5]Los matemáticos *teóricos-conjuntistas* han demostrado que con las operaciones elementales de conjuntos y fórmulas lógicas de primer orden como las que podemos hacer con SQL se puede recrear la matemática que conocemos actualmente.

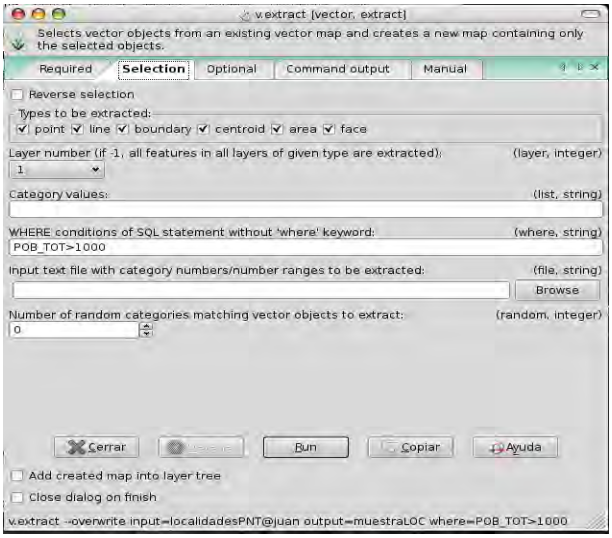



Figura 4.4: Cuadro de diálogo del comando `v.extract` utilizado para crear mapas vectoriales cuyos objetos sean subconjuntos de otro.

**Crear *Buffers*. O cómo *inflar* líneas y puntos.**

Uno de los algoritmos más usados para estos fines es el *buffering*. Éste crea un disco de radio $r$ tomando como centro cada punto a convertir. Si es una curva creará una región a lo largo de ésta tomando como frontera dos curvas paralelas a la curva dada y alejados de ésta a una distancia $r$. Este proceso es útil para modelar, simular o analizar regiones aledañas a ríos, carreteras, fronteras de propiedad, etcétera. El parámetro $r$ no tiene que ser constante, puede ser un atributo, o proporción de atributo, del mismo mapa. Otra cosa a considerar sobre el parámetro de radio $r$, también llamado *distancia*, es la unidad métrica sobre la que está definido. Si los mapas no tienen proyección, estas unidades estarán en grados. Si se requiere una distancia real como metros, pies, etc., se debe proyectar el mapa como mejor convenga. En este ejemplo, como el mapa no está proyectado utilizaremos unidades en grados, lo que equivale a la longitud de arco.

**Determinar el parámetro $r$** Al pensar en el problema surgen algunas cosas a considerar antes de hacer el *buffering*. El primero que se presenta es que, para que sea realista, *a fortiori*, el disco a crear para cada punto debe estar en función del tamaño poblacional de la localidad. Es decir deberemos usar como radio el campo `POB_TOT`. Sin embargo, el rango de este campo varía de 0 a más de 1.8 millones. Si usamos sólo este atributo como valores del radio, habría áreas que no sólo cubrirían a otras si no que cubrirían al mismo globo varias veces. Para evitar esto debemos darle un factor de escalamiento adecuado. Este factor podría ser asignarle un área al punto que concentre mayor población similar a su área verdadera. Hagamos una busqueda de esto, podemos hacer un **sencillo análisis estadístico** univariado a la variable `POB_TOT` y ver cuales son sus valores extremos. Esto lo podemos hacer con el comando:

```
v.univar map=muestraLOC@mapset column=POB_TOT
```

El resultado obtenido fue el siguiente:

```
v.univar map=muestraLOC@mapset column=POB_TOT
Incompatible vector type(s) specified,
only number of features, minimum,
maximum and range can be calculated
number of features with non NULL attribute: 8371
number of missing attributes: 0
number of NULL attributes: 0
minimum: 1001
maximum: 1.82089e+06
range: 1.81989e+06
```

Aquí podemos ver que el valor máximo es $1{,}82089e+06$ y buscando este valor en la tabla de atributos, nos damos cuenta que corresponde a la delegación *Iztapalapa* en la Ciudad de México. Con la heramienta de medición del monitor podemos medir el radio aproximado del polígono de la región. También podemos dar click con la herramienta de atributos al mapa de localidades de INEGI, 2009 en el polígono correspondiente a la delegación *Iztapalapa*. El mapa puede ser cargado a GRASS como lo hemos visto o para un vistaso rápido lo podemos hacer en QGIS. El área del polígono de *Iztapalapa* es 0.010 grados$^2$. Para sacar el radio de un disco con área equivalente deberemos de utilizar la fórmula:

$$radio = \sqrt{\frac{área}{\pi}}.$$

Haciendo las correspondientes susticiones obtenemos que el radio es aproximadamente: 0.056419 . Este será el factor de escalamiento para el valor máximo, pero para obtener el verdadero factor de escalamiento tendremos que normalizar[6] los valores de `Pob_Tot` para que el máximo sea igual a uno y el mínimo mayor que cero y por último multiplicarlos por el radio obtenido arriba. Haciendo las operaciones necesarias concluimos que el verdadero factor de escalamiento es:

$$r = 3{,}0984e-08.$$

Podemos entonces hacer el *buffering* con el siguiente comando y parámetros:

```
   v.buffer --overwrite input=muestraLOC@mapset
output=localidadesAREA@mapset
type=point bufcolumn=POB_TOT
scale=3.0984e-08
```

El parámetro `type` es donde asignamos que tipo de primitiva geométrica queremos convertir en área, en este caso es punto. La figura 4.5 muestra la ventana del comando `v.buffer` y el mapa resultante de esa aplicación. Este comando no incorpora la tabla de atributos del mapa base, por tanto tendremos que *conectar* la base de datos del mapa base. El comando `db.connect` realiza dicha tarea, la orden es la siguiente:

[6] dividirlos entre el máximo.

```
v.db.connect -o map=localidadesAREA@mapset
             table=muestraLOC
```

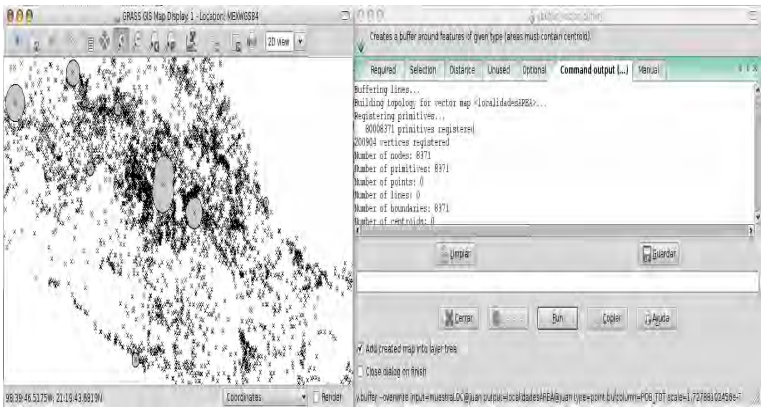


Figura 4.5: Cuadro de diálogo del comando `v.buffer` (derecho) y mapa resultante en el monitor del lado izquierdo. Los discos grises son las áreas producidas con `v.buffer`, los "puntos" restantes también son áreas pero son muy pequeñas para que se puedan apreciar como discos.

Después de este proceso hemos creado un mapa vectorial de polígonos (áreas) que representa el tamaño poblacional de una localidad con un área de forma lineal. Pensando en el problema original, parece que no satisface nuestras necesidades pues hay dos problemas principales: *i)* Hay localidades con muy poca gente que tienen un área muy pequeña. Si quisieramos tomar en cuenta estas localidades tendríamos que aumentar significativamente la resolución de nuestra región de estudio, lo que llevaría a incrementar exponencialmente el tiempo de cómputo. *ii)* Hay sobrelapamiento en las áreas, esto implica que a cada área le pueden corresponder más de un valor poblacional. ¿Cómo podemos determinar cuál tomar? Para resolver esto tendríamos que hacer algunos ajustes y cálculos más. Parece ser que para calcular el *indice de dificultad* no nos sirve mucho este método para crear áreas.

**Teselaciones**

Pensando en evitar los problemas mencionados arriba, debemos buscar una forma de definir áreas, asignarle valores de población y evitar sobrelapamientos. La técnica ideal para esto, en especial para no tener sobrelapamientos, es la de particionar adecuadamente el mapa en regiones ajenas es decir, que no se intersequen. Este proceso lleva el nombre de *teselación* y no es más que una partición del plano mediante polígonos que convenientemente agrupados recubren enteramente el plano dicho de otra forma, partir una región a manera de un mosáico. Por definición, una teselación debe tener dos requisitos fundamentales:

- No pueden haber huecos entre los polígonos.
- No puede haber superposición o sobrelapamiento de los mismos.

No podíamos buscar un modelo matemático más adecuado. Para esto, en el mundo de los SIG existe una implementación muy socorrida de un algoritmo llamado *teselación de Voronoi* o *polígonos de Thiesen*. Este algoritmo crea una partición del plano al unir los puntos entre sí trazando las mediatrices de los segmentos de unión. Las intersecciones de estas mediatrices determinan una serie de polígonos en un espacio bidimensional alrededor de un conjunto de puntos de control, de manera que la frontera de los polígonos generados sea equidistante a los puntos vecinos, designando de esta manera una área de influencia.

Como vemos, parece que este método para generar áreas es ideal para nuestro problema ya que se evitan los dos problemas principales, sobrelapamiento y huecos. Además, al definir el área de influencia, no por su densidad poblacional si no por su proximidad con otras localidades vecinas, hace posible una mejor resolución del modelo, dándole mayor área de influencia a localidades aisladas y por ende con menos impacto antropogénico e inversamente un área de influencia pequeña a localidades densamente pobladas. Así podemos relacionar directamente el área de influencia como región potencial para promover la conservación y con esto un zona más adecuada para calcular directamente el índice de *dificultad para la conservación* que queremos.

**v.voronoi** Tomaremos como mapa base el de localidades con más de 1000 habitantes i.e.(`muestraLOC`). El comando para realizar la teselación mencionada se puede utilizar de la siguiente forma:

```
v.voronoi input=muestraLOC@mapset output=voronoiM@mapset
```

Se generará un nuevo mapa de áreas que cubre todo el cuadro de la localidad, incluyendo el mar. Para resolver esto haremos uso de las operaciones de conjuntos, en este caso una intersección. Si utilizamos el mapa base de densidad poblacional estatal, sólo para tomar el perímetro del país y lo intersectamos con el mapa teselado de localidades, obtendremos un mapa en el cual tendremos los polígonos que están en los dos mapas, dejando de lado las teselaciones del mar. El comando para esto es:

```
 v.overlay -t  ainput=voronoiM@mapset
binput=divisionpolDENSPOB@PERMANENT
output=AreasTRIM@mapset operator=and
```

El mapa obtenido puede verse en la figura 4.6, los atributos de ambos mapas se han incorporado a los de este nuevo mapa.

### 4.1.5. Transformar mapas vectoriales a ráster

En el modelo vectorial las figuras geométricas no tienen problemas con la escala pues un punto siempre es un punto sin importar a que distancia estemos de él. Lo mismo se aplica para las líneas y polígonos. Esto es muy diferente en los modelos ráster pues la unidad mínima de este modelo es la celda, que es un área de dos dimensiones. Si queremos transformar un mapa vectorial a uno ráster tenemos que tener en cuenta esto. Puede ser que si queremos representar una carretera vectorial en un ráster esto se convierta en una cadena irregular de

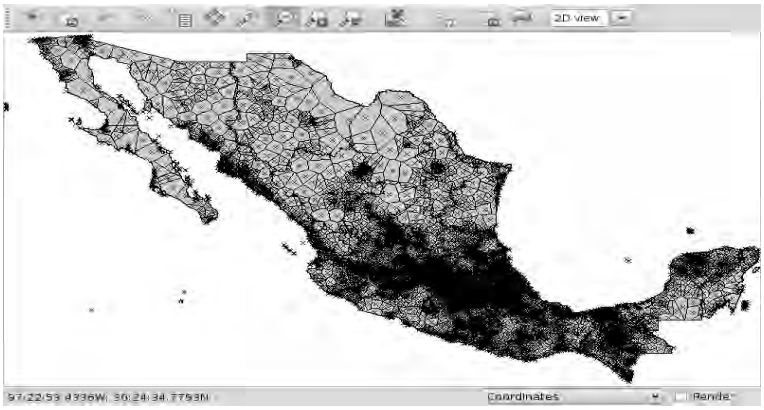



Figura 4.6: Monitor con mapa desplegado de áreas de localidades creado con la teselación de *Voronoi / Thiessen* utilizando el comando `v.voronoi` .

rectángulos gigantes. Se debe definir entonces un tamaño de elemento de área mínima antes de transformar o procesar datos ráster. Este elemento de área se llama *resolución* y se define en *la región* de trabajo.

**Configurando la región de trabajo**

En GRASS una región hace referencia a un área geográfica delimitada basada en un sistema de coordenadas y una proyección de en particular. Cada región también tiene asociado valores de resolución espacial para latitud y longitud, necesarios para los datos ráster. Esta resolución define el tamaño de las unidades mínimas que se van a representar, llevan el nombre de *celdas*.

Los valores de la frontera se definen con los valores extremos de la región de estudio deseada. Es decir, para definir una región se necesitan los puntos máximos y mínimos para las coordenadas $x$ (longitud) y $y$ (latitud). Las fronteras norte y sur en GRASS se llaman *northings* y las del este y oeste se les llama *eastings*. Al definir la resolución, GRASS calcula el número de celdas necesarias para cubrir toda la región, las resoluciones *norte-sur* y *este-oeste* no necesariamente tienen que ser las mismas, permitiendo la existencia de celdas rectangulares y no únicamente cuadradas. Generalmente todos los módulos ráster y de visualización son afectados por los parámetros de la región, a excepción de la mayoría de los módulos vectoriales. Al crearse una localidad, GRASS asigna de forma automática ciertos parámetros de región para todos los *mapsets*. Sin embargo, al ser multiusuario, la región de cada *mapset* puede modificarse para satisfacer necesidades particulares. El comando para manipular los parámetros es `g.region`.

**g.region** Este comando tiene muchas opciones para configurar la región de trabajo. Puede ser por medio de entrada manual o copiando los parámetros de región de algún mapa existente; sea ráster o vectorial, incluso 3-D. Sugiero se revise todas estas opciones directamente en la ventana del comando. Para que nuestra transformación vectorial-ráster este bien definida, en una escala razonablemente grande, será necesario cambiar los parámetros de resolución

de la región. Una buena resolución para el índice puede ser: 0:00:1 , i.e. un segundo de grado. Para definir esto en la consola se debe utilizar lo siguiente:

```
g.region nsres=0:00:1 ewres=0:00:1
```

Corroboremos el cambio viendo los parámetros de la resolución con:

```
g.region -p
```

Si la localidad está bajo algúna proyección, la resolución, no será en grados si no en las unidades cartesianas definidas en la proyección; metro, Kilómetro, pié, milla, etc.

Para calcular el *índice de dificultad* utilizaremos los mapas de teselación (`AreasTRIM`), densidad de población (`divisionpolDENSPOB`) y el de sitios prioritarios (`SitiosPrioritarios` ) y punto a punto calcularemos el índice. Antes de eso será necesario convertir los mapas mencionados en ráster y luego hacer el cálculo celda a celda. Por esto fue necesario modificar la resolución de nuestra región. Si no se satisface la resolución del mapa resultante de la transformación vectorial a ráster se puede modificar nuevamente y volver a transformar el mapa vectorial.

#### Transformación vectorial-ráster

**v.to.rast** Este comando nos servirá para transformar mapas vectoriales a ráster. El algoritmo toma la sección del mapa vectorial correspondiente a la región asignada en la localidad (en nuestro caso será todo el mapa). Después divide en una gradilla de tamaño tal que cada celda tendrá la longitud asignada como *resolución* de la región de trabajo (ver tal valor con: `g.region -p`). Una vez determinada esta *cuadrícula* se realiza la conversión, que asigna un valor numérico a cada celda. Este valor depende del atributo que hayamos seleccionado de entre todos los posibles de la base de datos del mapa vectorial. Este *campo* tiene que ser de tipo numérico, de lo contrario no se podrá realizar la conversión. Para los mapas que hemos trabajado utilizaremos como campos los atributos:

- `POB_TOT`[7] para el mapa de áreas teseladas (AreasTRIM).
- `PO_TO_2000` Para el mapa de densidad poblacional.
- `PRIORIDAD` Para el mapa de Sitios Prioritarios.

La forma en la que transformaremos estos mapas será con las órdenes:

```
v.to.rast input=AreasTRIM@mapset
          output=AreaTRIMRAST column=a_POB_TOT
v.to.rast input=divisionpolDENSPOB@PERMANENT
           output=despobEstatalRAST column=PO_TO_2000
```

[7] Al realizar la *intersección* de mapas, los atributos de los dos mapas se agregaron a la base de datos del resultante. Por tanto, el nombre del atributo `POB_TOT` se cambió por `a_POB_TOT`. Para ver los nombres de los campos (columnas) de los atributos úsese: `v.info -c map=NombreDelMapa@NombreMapset`

Tenemos listos dos mapas, para el tercero habrá que hacerle algunos tratamientos más porque el campo que nos interesa, i.e. `PRIORIDAD` es una cadena de caracteres; los valores que toma son: *extrema, alta, media y HUECO*. ¿Cómo podremos convertir estas categorias en números?

**Convertir categorias cualitativas en cuantitativas**

Para hacer esto deberemos modificar la base de datos del mapa de *Sitios Prioritarios*. Los pasos para lograrlo serán los siguientes:

1. **Agregar un nuevo campo (columna) a la base de datos del mapa de SitiosPrioritarios**.
   Este nuevo campo será del tipo entero (*integer*) y lo llamaremos `CATPRIO`.
   El comando a utilizar será:
   `v.db.addcol map=SitiosPrioritarios@mapset columns=CATPRIO integer`

2. **Relacionar unívocamente cada una de las categorías con un valor entero.**
   Por ejemplo para la prioridad `extrema` se le puede asociar el 3, para `alta` el 2 y para `media` el 1, los valores `HUECO` le podremos asignar el 0 porque representan sitios vacios alrededor de otros categorizados. Estos valores se agregaran en el nuevo campo que creamos en el paso anterior llamado `CATPRIO`. Para hacer esto usaremos el comando `v.db.update` junto con parámetros de busqueda SQL. La figura 4.7 muestra la ventana correspondiente a este comando. Las órdenes para ejecutar el paso 2 serán las siguientes:

```
v.db.update --verbose map=SitiosPrioritarios@mapset
   column=CATPRIO value=3 where=PRIORIDAD='extrema'

v.db.update --verbose map=SitiosPrioritarios@mapset
   column=CATPRIO value=2 where=PRIORIDAD='alta'

v.db.update --verbose map=SitiosPrioritarios@mapset
   column=CATPRIO value=1 where=PRIORIDAD='media'

v.db.update --verbose map=SitiosPrioritarios@mapset
   column=CATPRIO value=0 where=PRIORIDAD='HUECO'
```

Hemos convertido las categorias cualitativas en valores numéricos enteros. Ahora ya es posible convertir a ráster este mapa seleccionando como campo la columna `CATPRIO`; el comado será:

```
v.to.rast input=SitiosPrioritarios@mapset
output=sitiospriorRAST@mapset column=CATPRIO
```

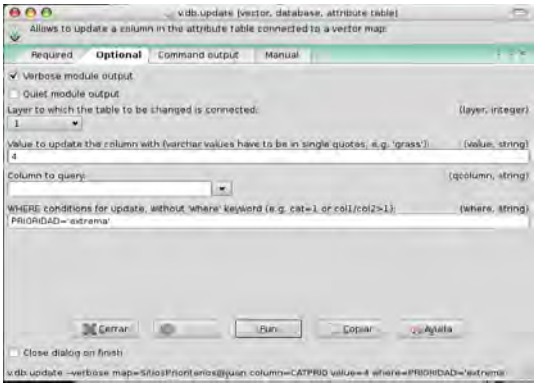



Figura 4.7: Cuadro de dialogo del comando `v.db.update` mostrando las opciones disponibles así como la utilización de los parámetros de busqueda con SQL.

### 4.1.6. Álgebra de mapas

Esta es una de las herramientas más poderosas de análisis para datos ráster. Con ella podemos modelar un sin fin de fenómenos porque tiene la capacidad de integrar información y procesarla celda a celda en todos los mapas ráster de entrada. Además tiene incorporadas funciones matemáticas de muchos tipos como trigonométricas, lógicas, exponencial, logaritmos, variables aleatorias, por mencionar algunas. GRASS esta pensado para ser un entorno de programación por tanto es fácil utilizar esta herramienta en *scripts* para hacer modelación de sistemas dinámicos. La implementación de autómatas celulares o modelos basados en agentes se puede hacer de forma sencilla. Incorpora un módulo llamado `r.mpeg` con el que se pueden crear animaciones a partir de imágenes estáticas. Si se quiere ver un ejemplo de esto ver Bowman (2008) que ejemplifica cómo programar autómatas celulares en dos dimensiones usando *scripts* y álgebra de mapas. Para nuestro ejemplo del *índice de dificultad* utilizaremos una fórmula muy sencilla que muestra en un rango del 0 al 1 la proporción de gente viviendo en una localidad con respecto a la población estatal multiplicada por un peso (también entre 0 -1) que representa la prioridad de conservación. Para esto tenemos que normalizar los tres mapas. Esto se consigue sabiendo cuál es el valor máximo para cada uno de ellos y haciendo álgebra de mapas, dividir el valor de cada una de las celdas por este. Para saber los rangos del mapa se puede usar el comando `r.info -r NombreMapaRaster@NombreMapset` en este caso el comando junto con la salida que necesitamos será:

```
1 r.info -r map=areaTRIMRASTLOW@mapset
2           min=1001
3           max=1820888
4
5 r.info -r map=denspobEstatalRAST@mapset
6           min=0
7           max=13096686
```

```
8
```

Es obvio que las líneas 2, 3, 6, 7 son la salida de los comandos de las líneas 1 y 5.

Con esta información podemos crear nuevos mapas ráster normalizados (con valores 0 - 1) y luego utilizar estos mapas para hacer el cálculo del índice. Si se quiere ahorrar tiempo y memoria se puede hacer esto en la misma operación sin tener que crear los mapas normalizados. Denotemos con:

$$\begin{aligned} A &:= \texttt{despobEstatalRAST@mapset}, \\ B &:= \texttt{areaTRIMRASTLOW@mapset}, \\ C &:= \texttt{sitiospriorRAST@mapset}. \end{aligned}$$

Conociendo los valores máximos de cada uno podemos obtener el *índice de dificultad* con la siguiente fórmula:

$$I_{dif} = \frac{\frac{B}{máx(B)} \times \frac{C}{máx(C)}}{\frac{A}{máx(A)}}$$

Donde claramente $I_{dif}$ representa el *Índice de dificultad*. Esta formulilla nos *modela* la dificultad para conservar una zona prioritaria para la conservación dependiendo de la cantidad de gente viviendo en el área de influencia de una población multiplicada por la cantidad de poblaciones vecinas. Aunque las letras $A$ $B$ $C$ son mapas, la fórmula está dada así porque las operaciones del álgebra de mapas son entrada a entrada, formalmente debería ser $A_{i,j}$ $B_{i,j}$ $C_{i,j}$ y $I_{dif}(i,j)$ correspondiente a la entrada del renglón $i$ y columna $j$. Como los mapas tienen la misma resolución son de la misma región, son matrices con el mismo número de renglones y columnas por tanto la fórmula está bien definida.

El comando en GRASS para realizar este cálculo se llama `r.mapcalculator`. Utiliza como base otro llamado `r.mapcalc` que tiene más flexibilidad por que es de más bajo nivel, a cambio de esto se necesitan conocimientos básicos de programación, por ahora limitémonos a éste. La figura 4.8 muestra la ventana de este comando. La orden para implementar el cálculo del índice sera:

```
r.mapcalculator amap=despobEstatalRAST@mapset
        bmap=areaTRIMRASTLOW@mapset
        cmap=sitiospriorRAST@mapset
        formula=(bmap/1820888.0) / (amap/13096686.0)
        * cmap/4 outfile=difpropEstatalRAST
```

### 4.1.7. Exportar resultados a otros formatos

Como GRASS utiliza la biblioteca GDAL/OGR, en particular puede escribir archivos en los formatos soportados por ésta. Esto quiere decir que podemos exportar los resultados en una variedad muy grande de formatos posibles. Además de éstos puede escribir en otros más. Basta con ver la cantidad de comandos relacionados con `v.out.*` y `r.out.*`. Por ahora exportemos

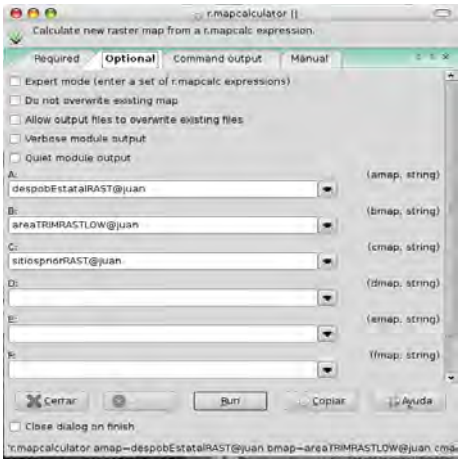



Figura 4.8: Cuadro de dialogo del comando `r.mapcalculator` mostrando los campos para asignar mapas.

nuestro mapa de *índice de dificultad* en formato GeoTIFF (soportado por GDAL) y guardémoslo en otro directorio. Este formato se podrá abrir después en cualquier SIG que lo soporte, que es la gran mayoría, incluso la familia *ESRI ARC*. El comando que usaremos para exportar nuestro mapa será:

```
r.out.gdal input=difpropEstatalRAST@mapset
format=GTiff output=~/GISejemplos/indicediffGRASSEXP
```

Terminado el proceso podremos abrirlo con QGIS, por ejemplo.

### 4.1.8. Usar GRASS en QGIS

Como vimos a lo largo del capítulo tres QGIS amplía sustancialmente sus capacidades con los *plugins*. Hasta ahora hemos visto geoprocesamiento básico con GRASS que desgraciadamente no es posible hacerlo con QGIS con sus herramientas básicas. La buena noticia es que QGIS tiene incorporado un *plugin* que actúa como cliente de GRASS. Con este *plugin* podemos crear localidades, abrir mapas y en general ejecutar los comandos de GRASS en él. Los comandos de GRASS se conectan directamente con la interfaz de QGIS por lo que podemos usar sus ventanas y demás herramientas. El administrador de GRASS queda gestionado por el *plugin* y los mapas en formato de GRASS se pueden abrir y visualizar en el *Map Canvas* de QGIS como si fuera cualquier archivo de SIG. Se pueden agregar y empalmar datos en otros formatos sin necesidad de hacer la conversión de formatos (`r.in.* o v.in.*`). No obstante si se quieren hacer análisis más complejos con estos mapas usando GRASS sí se tendrán que importar. Como ejemplo visualicemos el mapa que generamos del *índice de dificultad* por medio del *plugin* en QGIS. Para esto debemos ejecutar QGIS en una terminal y una vez abierto ir al menú `Plugins / Manage Plugins`. Buscamos y seleccionamos el apartado de GRASS y damos aceptar. Apareceran unos nuevos íconos en la barra de herramientas.

Estos íconos hacen referencia a todas las posibles actividades que podemos hacer con GRASS usando el *plugin* de QGIS. Estas son:

- Abrir *mapset*.
- Crear *mapset* nuevo.
- Cerrar *mapset* actual.
- Añadir mapa ráster de GRASS.
- Añadir mapa vectorial de GRASS.
- Crear mapa vectorial de GRASS.
- Editar mapa vectorial de GRASS.
- Herramientas de GRASS.
- Mostrar información de la región.
- Editar parámetros de región.

Como ya tenemos creada la localidad podemos buscarla y abrirla. La figura 4.9 muestra las ventanas para abrir localidades y cargar mapas.


(a) Selección de localidad y *mapset*. (b) Selección de capa a cargar en QGIS.

Figura 4.9: Configuración del *plugin* de QGIS para importar capas de GRASS

Al abrirla podemos cargar el mapa del *índice de dificultad* que creamos (`difpropEstatalRAST`). La figura 4.10 muestra cargadas las capas de este mapa y el de *áreas de localidades*.

## 4.2. Geoprocesamiento básico con datos ráster

En esta actividad vamos a trabajar con las imágenes Landsat prestadas por la bióloga Rocío Alanís Anaya, 2010. Además utilizaremos un dato ráster llamado *modelo digital de elevación* (DEM) indispensable para tareas de geoprocesamiento. Este modelo representa el valor de elevación de una región en la Tierra celda a celda. Con estas imágenes podremos realizar otras actividades más relacionadas a geosimulación y prospección.

### 4.2.1. Importar ráster en GRASS

El paquete de imágenes Landsat consta de siete rásters. Los nombres son bastante largos y están proyectados en el sistema UTM zona 14. La localidad con la que estamos trabajando es sin proyección y con datum *WGS84*.

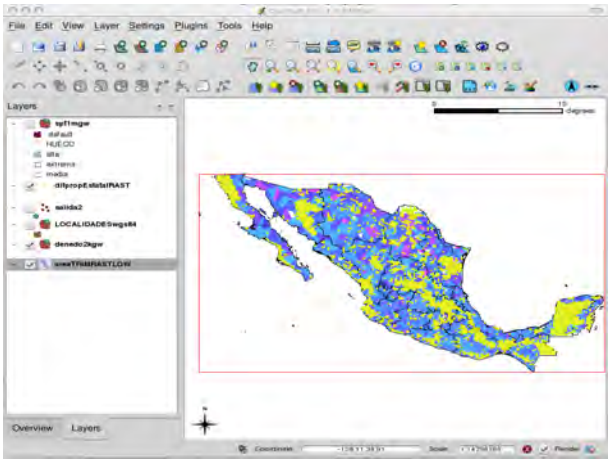


Figura 4.10: QGIS puede cargar datos del formato de GRASS por medio del *plugin* `GRASS Layer`. Aquí abiertos y sobrelapados dos capas de GRASS en el *Map Canvas* de QGIS.

GRASS puede *sobrecargar* ciertos tipos de proyección e importar los mapas con la transformación adecuada, sin embargo, por alguna razón desconocida, no es posible *sobrecargar* directamente estas imágenes *Landsat*. Esto no es impedimento porque podemos transformar estos datos con las herramientas GDAL/OGR que vimos en el capítulo anterior (pág. 68). El código EPSG para la localidad en GRASS con la que estamos trabajando es: 4326. Recomiendo crear una carpeta nueva con el nombre `wgs84` y ahí guardar los mapas transformados. Estos comandos, aunque no forman parte de GRASS se pueden ejecutar en la consola donde está activo GRASS. [8] El cómando para transformar estos mapas será con la órden:

```
gdalwarp -t_srs EPSG:4326 NombreMapaUTM14.tif
    wgs84/NombreMapaSalidaWGS84.tif
```

También se pueden importar datos desde servicios *wms* esto se logra con el comando`r.wms` para ráster y `v.wfs` para vectoriales.

**Transformación automática de archivos *vía Shell*.**

Podemos optar por hacer esta transformación siete veces o utilizar las funciones de programación sencillas como el bucle `for`. La forma de utilización de éste es igual a la que se usa en el *shell*. Suponiendo que todos los ráster están en la misma carpeta, la órden para tal conversión sería:

[8] Con `mkdir wgs84` si se usa Linux

```
juan@consola:/home/GIS/tutorial$
1 j=1
2 for i in `ls *.tif`
3 do
4 gdalwarp -t_srs EPSG:4326 $i wgs84/colimaWGS84$j.tif
5 j=$[$j + 1]
6 done
```

**Explicación:** En la línea 1 se inicializa la variable que servirá para numerar los mapas nuevos. En la línea 2 se dan los parámetros del ciclo `for`. En este caso *i* funciona como una variable que toma los valores de salida del comando `ls *.tif`. Este comando devuelve todos los archivos con extención `.tif` presentes en el directorio donde se ejecute. La variable *i* tomará entonces el nombre de cada archivo `.tif`; que es la extensión de los mapas ráster con formato *GeoTif*. **Es importante utilizar las comillas inversas cuando se define la ejecución de cualquier comando que esté dentro de otro comando.** En este caso el comando `ls` se ejecuta dentro del comando `for`. Si no se usan estas comillas el *shell* no tomará la orden siguiente como algo a ejecutar; si no como una variable o una cadena de caracteres, en el mejor de los casos. Las palabras reservadas `do` y `done` delimitan las órdenes que se ejecutarán dentro del cíclo las veces que hayan sido definidas en la cabecera del `for` (línea 2). Dentro del `do` y el `done` está definida la órden principal. i.e. `gdalwarp`. Cuando anteponemos el signo $ a una variable estamos solicitando el valor de esa variable, como *i* va tomando un nombre distinto por cada uno de los archivos `tif`, cuando damos la órden de la línea 4, lo que hacemos es asignarle el íesimo archivo `tif` a `gdalwarp` para que transforme éste en uno nuevo, a saber, ubicado en la carpeta `wgs84` con el nombre formado por el sufijo: `colimaWGS84` seguido de el número dado por la variable *j*. En la línea 5 sólo se actualiza el valor de la variable *j* para que cada archivo transformado tenga un número propio. El uso de ciclos fácilita en gran medida el trabajo porque lo automatiza. En una primera aproximación, estos nuevos comandos pueden parecer complicados pero vale la pena trascender el prejuicio porque su uso nos podrá ahorrar mucho tiempo. Si se quiere profundizar en estas herramientas recomiendo revisar Arena (2007), que trae muchos consejos y herramientas, y Figgins y Love (2000) que trae una descripción detallada de la mayoría de comandos disponibles del *Shell*.

Una vez completada la transformación podemos proceder a importar los datos sin necesidad de *sobrecargar* la proyección de los mismos. La forma de cargarlos es muy similar a los datos vectoriales; el comando que se utiliza es `r.in.gdal` porque son datos tiff soportados por la biblioteca GDAL. Aprovechando que ya sabemos utilizar bucles para repetir procesos usemos nuevamente el ciclo `for` para importar los mapas. La órden sería la siguiente:

```
juan@consola:/home/GIS/tutorial$
j=1
for i in `ls *.tif`
do
   `r.in.gdal input= $i output=colimarast$j`
    j=$[$j + 1]
done
```

La consola devolverá un mensaje indicándonos el término del proceso. Concluido éste podremos visualizar los mapas usando las herramientas del administrador y del monitor.

### 4.2.2. Obtención de mapas de México por medio de la clave de cartas

.

INEGI tiene a disposición pública y gratuita una cantidad considerable de mapas digitales en formatos ráster y vectorial. Estos datos pueden ser descargados del sitio de descargas gratuitas de INEGI.

`http://www.inegi.org.mx/inegi/default.aspx?s=geo&c=911`

Para descargar estos datos, a nivel regional se puede hacer especificando las coordenadas extremas requeridas (*northings* y *eastings*), útil si vamos a descargar sólo pocos mapas. Otra opción es por medio de la clave de cartas. INEGI dividió a México en una gradilla regular que distribuye homogéneamente todas las cartas geográficas del territorio nacional. Existen dos gradillas distintas caracterizadas por la escala que es usada para definir tal partición (teselación). Las dos escalas disponibles son: $1:250,000$ y $1:50,000$. Cada carta tiene una clave que representa la región que representa. Utilizando esta clave podremos bajar directamente los mapas necesarios, pero ¿ cómo saber que claves usar ? Para responder esta pregunta utilizaremos QGIS [9]. Vayamos al sitio de mapas de CONABIO donde bajamos los mapas de *Densidad Poblacional* y *Sitios prioritarios para la conservación terrestre*.

`http://www.conabio.gob.mx/informacion/gis/`

En la sección `Topografía / Información general` se encuentran los mapas vectoriales de los *índices de cartas* para las escalas: $1:250,000$ y $1:50,000$. Bajemos la escala más grande ($1:50,000$) sin proyección y con datum *WGS84*.

Si abrimos este mapa en QGIS podremos cargar cualquiera de las imágenes *Landsat* que hayamos convertimos al datum *WGS84* sin proyección. De esta forma podremos utilizar la *Herramienta de Selección de Atributos* para tomar la región de interés y buscar a qué claves de cartas corresponde toda la región que queremos analizar. La figura 4.11 muestra la selección de los cuadros correspondientes a la zona de interés. Una vez seleccionada la región podemos ver los atributos de estos objetos seleccionados accediendo a la tabla de atributos del mapa de cartas. En esa ventana de atributos podemos discriminar

[9] Ya que no necesitamos geoprocesamiento avanzado

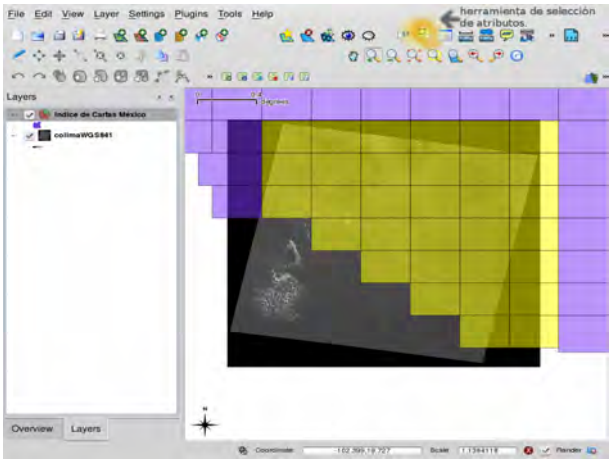



Figura 4.11: Cuadros seleccionados con la *herramienta de selección de atributos* ( resaltada en la barra de herramientas). Estos cuadros representan las claves de las cartas topográficas necesarias de la región a trabajar, en este caso el estado de Colima.

sólo estos datos mediante las herramientas incluidas en la parte inferior de la ventana. Hay una que nos permitirá copiar los atributos de los cuadros seleccionados. Al hacer click en este ícono (o con las teclas `Ctrl + C`) podremos pegar esta tabla en cualquier editor de texto u hoja de cálculo. Como sólo queremos las claves de cartas podemos utilizar una hoja de cálculo como *OpenOffice Spreadsheet* o *Excell*. Al importarlas adecuadamente (con el símbolo de espacio como delimitador) tendremos divididos estos atributos por columnas, la última columna muestra las claves de las cartas correspondientes. Para la zona de Colima éstas serán:

- E13B26
- E13B25
- E13B37
- E13B24
- E13B36
- E13B87
- E13B23
- E13B35
- E13B86
- E13B47
- E13B22
- E13B34
- E13B46
- E13B33
- E13B32
- E13B45
- E13B44
- E13B43
- E13B57
- E13B42
- E13B56
- E13B55
- E13B54
- E13B53
- E13B67
- E13B66
- E13B65
- E13B64
- E13B77
- E13B76
- E13B75
- E13B27

**El Continuo de Elevaciones Mexicano (CEM)**

Este modelo integra información de varios modelos digitales de elevación del territorio nacional basados en la cartografía topográfica a escala 1: 50 000 producida por el INEGI. La última versión de estos datos fue generada en

2003. El CEM es un modelo ráster y proporciona elevaciones celda a celda del territorio en un continuo nacional con datums, unidades y elipsoides consistentes. Actualmente se cuentan con proyecciones UTM (zonas 11 a 16 norte) y Cónica Conformal de Lambert. En palabras del INEGI, 2010: *"se busca que el CEM proporcione el mejor conjunto de datos disponibles en el tema, de modo que sirva como base para diversas aplicaciones (ortorrectificación, hidrología, predicción de zonas de saturación, evaluación de zonas de riesgo, simulación de procesos dinámicos, etc.)"* Las cartas del CEM están disponibles en el mismo sitio: `http://mapserver.inegi.org.mx/DescargaMDEWeb/`. Podemos proporcionar ahí la clave de la carta correspondiente para bajarlas. La interfaz para descargar los datos no es eficiente ni amigable, por tanto tendremos que bajar todos los datos de forma manual.

Antes de descargar los datos sugiero crear una carpeta donde guardarlos, al finalizar todas las descargas sugiero también crear una carpeta nueva dentro de la primera con el nombre `zips`, aquí será donde guardaremos una copia de respaldo en caso que perdamos o modifiquemos las originales. En el capítulo 1 vimos que el *shell* es también un intérprete de comandos y que podemos definir ciclos (For, While), condicionales (If, Switch) y formas sentenciales (funciones) dentro de él. En este caso, para ahorrar trabajo, podemos descomprimir todos los archivos de una vez con un ciclo For. El cuadro siguiente muestra los comandos para hacerlo.

**juan@consola:/home/GIS/tutorial$**

```
1 mkdir zips
2 cp *.zip zips/
3 for i in `ls *.zip`
4 do
5   unzip $i
6 done
7 rm *.zip
```

**Explicación** En la línea 1 creamos un directorio con nombre `zips` con el comando `mkdir`. En la línea 2 copiamos todos los archivos con extensión `.zip` (i.e. los comprimidos que bajamos del sitio) y los guardamos en la carpeta `zips` que acabamos de crear. Observemos que usamos el símbolo universal `*` para especificar TODOS los archivos dentro de la carpeta que tengan terminación `.zip`. En la línea 3 es donde comienza el bucle. Aquí la variable *i* va a actuar como un archivo con extensión `.zip`, por eso dentro del `for` utilizamos el comando `unzip $i` para descomprimir a *i*. El signo de pesos es para que el *shell* actué sobre la variable *i* que es un archivo comprimido. Este proceso se hace tantas veces como archivos `.zip` se tengan en la carpeta. Los procesos ejecutados dentro del `for` están definidos entre las palabras `do` y `done` (líneas 4 a 6). En la última línea se borran todos los archivos `.zip` dejando intacta la copia de respaldo que dejamos en la carpeta `zips`. Es importante hacer notar que para que ejecutemos un comando en el *shell* dentro de otro comando, como en el caso de la línea 3, este comando *anidado* debe estar definido entre comillas simples invertidas. Al terminar todo este proceso, si vemos lo que hay dentro de la carpeta, nos encontraremos con archivos con la extensión `.bil, .blw,`

`.hdr`. El primero es el ráster y los otros dos los metadatos correspondientes. Al pedir un listado de éstos tendremos algo como esto:

```
GRASS 6.4.0RC5 (MEXWGS84):~/colimaDEM > ls
MDE-n191500s190000e1032000o1034000.bil
MDE-n181500s180000e1024000o1030000.bil
MDE-n181500s180000e1024000o1030000.blw
MDE-n181500s180000e1024000o1030000.hdr
                    ...
MDE-n181500s180000e1030000o1032000.bil
MDE-n181500s180000e1030000o1032000.blw
MDE-n181500s180000e1030000o1032000.hdr
MDE-n183000s181500e1024000o1030000.bil
```

Tenemos listos los *MDE* para importarlos a GRASS. La forma de hacerlo será nuevamente con un bucle, porque son muchos. El comando para importar en GRASS es `r.in.gdal`, utilizaremos este mismo sólo que esta vez le daremos el parámetro de *sobrecarga* `-o` es decir que transforme los datos con diferentes datums o proyecciones al mismo definido en la localidad y región en la que estamos trabajando. Con este parámetro GRASS los convierte de forma automática. Las órdenes para importar los mapas serán:

```
1 j=1
2 for i in `ls *.bil`
3 do
4 r.in.gdal -o input=$i output=colDEM$j
5 j=$[$j + 1]
6 done
```

En la línea 1 declaramos una variable *j* que nos va a servir para numerar y nombrar los mapas transformados, de esta manera además, los nombres de los mapas son más cortos y fáciles de manejar.

### 4.2.3. Cómo empalmar mapas ráster

En muchas ocasiones necesitaremos crear un mapa ráster a partir de mapas pequeños. Es decir, quisieramos una herramienta que pegara mapas contiguos haciendo un mosaico sin bordes. En el ejemplo que estamos usando, los *MDE* (también llamados *DEM* por sus siglas en inglés) que bajamos son muchos y dificulta el análisis, pues cada que queramos hacer algo en toda la región habría que hacerlo en cada uno de los 32. La herramienta que estamos buscando se llama `r.patch` y acepta una lista grande de mapas, devolviendo la unión cartográfica de todos éstos. Para utilizarlo podemos crear una variable temporal `MAPA` y luego con un ciclo *for* concatenar los nombres de los mapas `colDEM1`, `colDEM2`, ... ,`colDEM32` separados por comas. También se puede usar la ventana del comando (simplemente ejecutando) `r.patch` y seleccionar todos los mapas requeridos oprimiendo la tecla `Ctrl`. Para los 32 mapas la orden para el pegado será:

```
r.patch input=colDEM1@mapset,colDEM3@mapset,
colDEM2@mapset,colDEM7@mapset,colDEM4@mapset,
colDEM5@mapset,colDEM6@mapset,colDEM8@mapset,
colDEM9@mapset,colDEM10@mapset,colDEM11@mapset,
colDEM12@mapset,colDEM13@mapset,colDEM14@mapset,
colDEM15@mapset,colDEM17@mapset,colDEM16@mapset,
colDEM19@mapset,colDEM18@mapset,colDEM20@mapset,
colDEM21@mapset,colDEM22@mapset,colDEM23@mapset,
colDEM24@mapset,colDEM25@mapset,colDEM26@mapset,
colDEM27@mapset,colDEM28@mapset,colDEM29@mapset,
colDEM30@mapset,colDEM31@mapset,colDEM32@mapset
output=colDEMTOT
```

El mapa resultante `colDEMTOT` será el producto de este pegado. En la figura 4.12 se puede ver el resultado de este módulo.

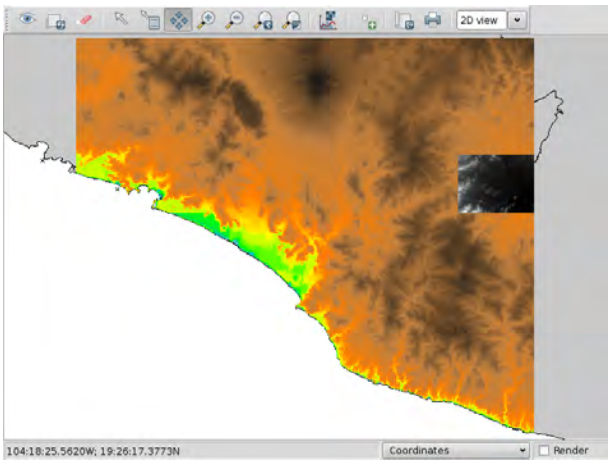



Figura 4.12: Mosaico generado con el pegado de 32 mapas ráster de elevación. El color está dado con la regla `elevation` en escala logarítmica. El cuadro obscuro es uno de los mapas que se usaron para generar éste. Se incluyó sólo para comparar tamaños. El comando utilizado es comando (`r.patch`).

### 4.2.4. Relieve topográfico

Una vez creado el mapa *global* de elevación podemos crear uno con una apariencia tridimencional que muestre las elevaciones. El comando para esto se llama `r.shaded.relief` y utiliza la altitud del sol y el azimut de Norte a Este, ambas magnitudes en grados. Con esto, el comando calcula la sombra que debiera tener el terreno dada sus diferencias de altura. Se le puede dar un parámetro de escalamiento en la elevación para darle un mayor realce. Para crear este mapa basta con usar el comando:

```
r.shaded.relief map=colDEMTOT@mapset
shadedmap=colSHADED@mapset
```

```
3 zmult=2 units=meters
```

El parámetro `zmult=2` (línea 3) es el valor de escalamiento de altura, también llamado *exageración*. Es importante asignar la unidad en la que está descrita la *altitud*, en este caso es *metro*. La figura 4.13 muestra el mapa resultante después de aplicar este comando al mapa de elevaciones.

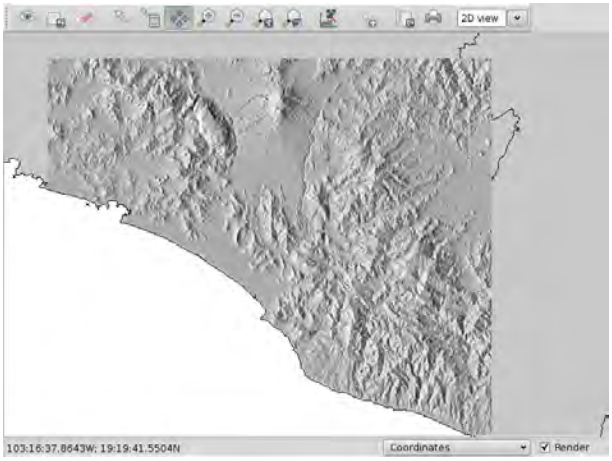



Figura 4.13: Mapa del relieve topográfico generado con el mosaico de elevación y el comando `r.shaded.relief` . El valor de azimut es de 270 grados y la altura del Sol de 30. Tiene un factor de exageración de 2.

Una vez creado este mapa se le puede asignar un *color de cobertura* para visualizarlo mejor. Por ejemplo: `colDEMTOT` con los colores de altitud.

### 4.2.5. Modelos Hidrológicos

GRASS tiene incorporados varios módulos de análisis hidrológicos como:

- Canales de corrientes de escorrentía
- Relleno de lagos
- Acumulación de flujos
- Líneas de dirección de flujos
- Simulación de aguas subterraneas
- Entre muchos otros

Usaremos el *rellenado de lagos* para ejemplificar uno de estos modelos.

**¿ Qué pasaría en la costa de Colima si aumentara 100 metros el nivel del mar?**

Esta pregunta, así como otras relacionadas a inundaciones, se puede responder simulando un aumento progresivo y discreto del nivel del mar en un punto particular de la superficie. Para hacer esta simulación se necesita un

modelo de elevación digital (que ya tenemos) para tener los datos de altitud en cada celda, un valor arbitrario en metros [10], que será la profundidad (en metros sobre el nivel del mar) a la cual va a llegar la simulación de inundación y una lista de puntos (semillas) donde comenzará ésta. Pueden ser varios puntos e incluso un mapa ráster.

Tomemos entonces un punto en la costa de Colima, i.e. en la orilla del mapa de elevación digital `colDEMTOT`. Este punto se puede tomar con la herramienta de atributos del monitor si tenemos seleccionado el mapa de elevación. Seleccionemos un punto cercano a la ciudad de Tecomán, las coordenadas de este punto son:

`-103.888015993,18.8358457164`

En la ventana de *salida* se puede ver que el valor en $z$ de este punto es $8$ metros, también se puede saber con el comando:

```
r.what -f input=colDEMTOT@mapset
east_north=-103.888015993,18.8358457164
```

**Rellenando lago** Utilizando el comando `r.lake` podemos obtener un mapa ráster con el área de la masa de agua aumentada hasta cubrir la profundidad deseada. La orden del comando es la siguiente:

```
r.lake dem=colDEMTOT@mapset wl=100 lake=colimaInundada
xy=-103.888015993,18.8358457164
```

Donde `wl` es el parámetro de profundidad (*water level*) y `lake` el nombre del mapa resultante. La figura 4.14 muestra el mapa resultante de esta simulación, como se puede ver, varias comunidades quedaron cubiertas por el aumento del nivel del mar, por lo que puede ser una herramienta muy importante para determinar zonas de riesgo y con esto optimizar planes de evacuación y prevenir tragedias.

### 4.2.6. Coloreando mapas

Es posible modificar los valores de color para mapas ráster, sobre todo aquellos monocromáticos. Los valores del rango de color se llaman *colormaps* y estos se definen por medio de un archivo de texto simple llamado *rules*. Este archivo puede ser uno creado por nosotros mismos o puede ser producto de un cálculo de optimización y equilibrio de color, especialmente útil para en imágenes *Landsat*. El comando para modificar los valores de color se llama: `r.colors`.

#### Colorear con valores arbitrarios

Lo primero que necesitamos es un pequeño archivo de texto llamado *rules* para definir los colores por el porcentaje del rango de elevación en el ráster.

[10] Pueden ser otras unidades, depende de las unidades de la región

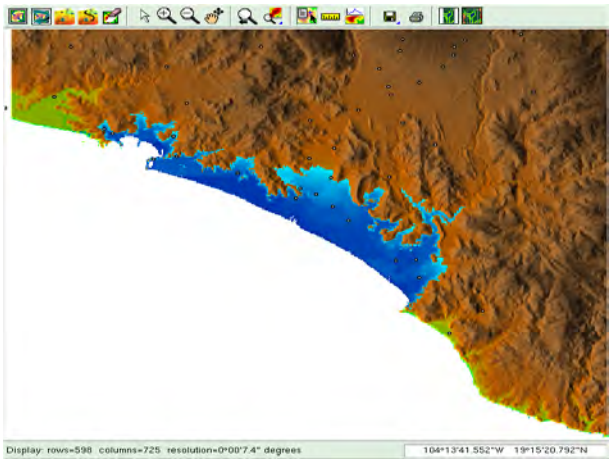



Figura 4.14: Mapa resultante de la simulación de inundación a 100 metros sobre el nivel del mar en las costas de Colima. Éste fue generado con `r.lake` y sobrelapado con el mapa vectorial de *localidades* con más de 1000 habitantes y con el modelo digital de elevación total de la región `colDEMTOT`.

El archivo consta de cuatro columnas. La primera indica el porcentaje de elevación y las otras tres el color asignado en notación RGB. Esta notación es un vector de tamaño tres cuyos valores en cada componente es un número entero entre 0 y 255. Este número es la saturación del canal correspondiente a la entrada, la primera es el valor en rojo (*Red*), la segunda en verde (*Green*) y la tercera en azul (*Blue*). La combinación de estos tres valores en el intervalo [0,255] crea cualquier color siendo el negro = [0,0,0] y el blanco = [255,255,255] los valores extremos. En el cuadro siguiente se muestra un ejemplo de como debe estar construido el archivo *rules*. Este archivo puede ser escrito con cualquier editor de textos (gedit, kate, emacs, vi, nano, etc).

*MisColores.rules*

```
0% 0 230 0
20% 0 160 0
35% 50 130 0
55% 120 100 30
75% 120 130 40
90% 170 160 50
100% 255 255 100
```

Guardamos este archivo con el nombre *MisColores.rules*. Asignemos esta regla de colores al archivo de relieve topográfico `ColSHADED`. Para hacer esto utilizamos el comando `r.colors` que con el argumento `color=rules` recibe en pantalla los valores de las reglas de color. Como ya tenemos un archivo guardado que contiene esta información, lo que podemos hacer es un truquillo del *shell* llamado *piping* o *entubamiento* de datos, que no es otra cosa más que direccionar las salidas de los comandos para que en vez de salir en pantalla los reciba otro comando. En este caso si utilizamos el comando `cat MisColores.rules` nos mostrará en pantalla todo el contenido del archivo. La idea es que esta salida la reciba el comando `r.color`. Como hemos visto en el capítulo 1, el *entubamiento* se logra con el símbolo `|` . El cuadro siguiente

muestra como crear nuevamente el mapa de relieve topográfico a partir del mapa DEM global como en la sección 4.2.4, asignarle al DEM los valores nuevos de color y crear un nuevo map que combine el relieve con estos colores.

```
1 r.shaded.relief map=colDEMTOT
2 shadedmap=colSHADED
3 zmult=2 units=meters --overwrite
4 cat MisColores.rules | r.colors map=colDEMTOT color=rules
5 r.his -n h_map=colDEMTOT i_map=colSHADED
6 r_map=colRojo g_map=colVerde b_map=colAzul --overwrite
7 r.compsite -d red=colRojo blue=colAzul green=colVerde
8 output=colCOMBINADO --overwrite
```

**Explicación** Las líneas 1 a 3 son las misma que en la sección 4.2.4. En la línea 4 se le asignan los valores de coloración mediante el *entubamiento* descrito anteriormente. Hay dos comandos que no hemos visto en las líneas 5 y 7 estos son: `r.his` que genera capas de mapa ráster roja, verde y azul combinando valores de tonalidad (*hue*), intensidad y saturación (HIS) a partir de capas de mapas ráster de entrada especificadas por el usuario. En este caso el mapa de tonalidad (`h_map`) será el DEM de toda la región (ver sección 4.2.3, pág 105) `colDEMTOT` y el de intensidad (`i_map`) será el del relieve topográfico `colSHADED`. Este comando crea tres nuevos mapas uno con valores en rojo (`colRojo`), otro en azul (`colAzul`) y otro en verde (`colVerde`). El argumento `--overwrite` sólo es para que, si ya existen mapas con los mismos nombres los reemplace por éstos nuevos, este argumento funciona de igual forma en cualquier comando de GRASS. El otro comando, `r.composite`, (línea 7) es para combinar los tres mapas de color en uno sólo a todo color (RGB). Le indicamos qué mapas tome para qué colores y cual será el nombre del mapa de salida, en este caso `colCOMBINADO`. La figura 4.15 muestra el resultado de estos procesos.

#### Asignar colores en imágenes *Landsat*

Los módulos para manipular imágenes satellitales, a pesar de ser datos ráster tienen su propio sufijo es decir, todos los comandos de estos módulos empiezan con la letra `i` de *image*. En particular hay un módulo que optimiza el contraste, saturación y coloración en imagenes *Landsat* y se llama: `i.landsat`. Podemos utilizarlo con las imágenes *Landsat* que hemos trabajado (Alanís-Anaya, 2010) necesitaremos usar tres imágenes, el comando sería el siguiente:

```
1 i.landsat.rgb -f -p -r red=colimarast1@mapset
2   green=colimarast2@mapset blue=colimarast3@mapset
3   strength=98
```

Una vez realizados los cálculos podemos proceder a crear un mapa *compuesto* que toma las tres distintas capas, con colores *optimizados*, y les asigna un color primario fotográfico (rojo, azul, verde) a cada una, i.e. color *RGB*. El comando para componer el nuevo ráster es similar al de la sección anterior, es decir:

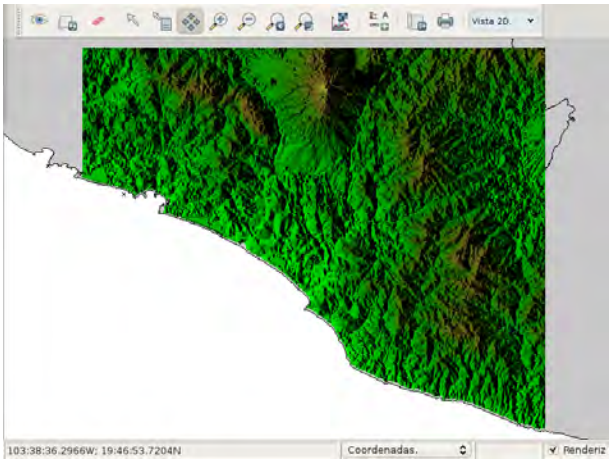



Figura 4.15: Mapa de color compuesto (RGB) por tres mapas, uno rojo, uno azul y uno verde; éstos tres generados con el comando `r.his` a partir de las reglas de coloración dadas por el archivo *MisColores.rules* asignadas al mapa digital de elevación que sirvió para dar los valores de tonalidad. Con el mapa de relieve topográfico `colSHADED` se asignaron los valores de intensidad.

```
r.composite red=colimarast1@mapset
green=colimarast2@mapset
blue=colimarast3@mapset levels=32
output=ColimaCombinado
```

Se pueden visualizar las imágenes en tres bandas de color con dos modelos diferentes; el *RGB* y el *HIS*. Éste último es el acrónimo de *Tonalidad (Hue), Intensidad (Instensity) y Saturación (Saturation)*, dependiendo las bandas a disposición se puede optar por uno u otro modelo. Esta forma de visualización se logra empalmando tres imágenes y asignándolas como una capa de color fijo. El comando forma parte de los módulos de visualización y se llama (`d.rgb`). En la figura 4.16 se puede ver una coloración falsa en bandas *Landsat* 4, 5, 7 utilizando el modelo *RGB*.

### 4.2.7. Contornos y objetos de las superficies

Otra de las herramientas más usadas para análisis de terreno es el *mapa de pendientes*. El módulo genera capas de mapa ráster de pendiente, orientación, curvaturas y derivadas parciales a partir de una capa de mapa ráster de valores reales de elevación. La orientación se calcula en sentido horario desde el Este.

Como sabemos, la pendiente se puede representar por una función contínua llamada *derivada*. Este módulo calcula la derivada de cada celda con respecto sus vecinos mediante una aproximación lineal, en otras palabras, aproxima un plano tangente al valor de la celda en cuestión. Dicho cálculo incorpora información sobre las derivadas parciales de la superficie, es decir, las derivadas respecto al eje $x$ y $y$, generando mapas de éstas e incluso de las segundas derivadas parciales y mixtas.

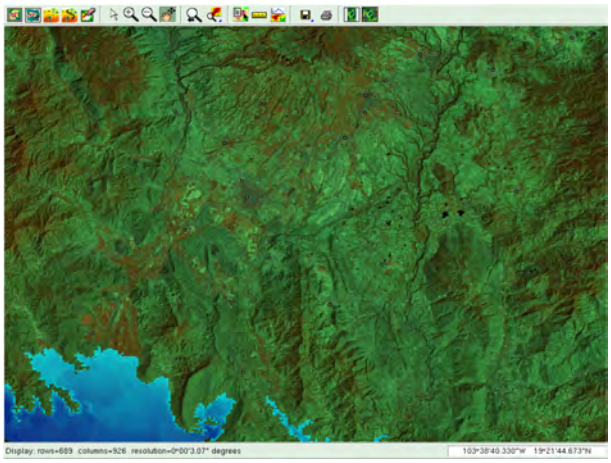


Figura 4.16: Imagen satelital que combina tres bandas *Landsat* (4, 5, 7) dándole una coloración falsa que resalta parcelas y localidades humanas en rojo.

Esta información es muy importante para análisis intrínsecos del terreno, busqueda de geodésicas, trayectorias cortas o líneas de flujo. Gracias a estos módulos se pueden hacer análisis complejos utilizando los elementos de la geometría diferencial . Estos mapas también son necesarios para otros módulos, como el módulo de radiación solar (`r.sun`) y de sombras (`r.sunmask`). El comando del módulo de pendientes se llama: `r.slope.aspect` y la forma de utilización, con la opción de generar los mapas con derivadas parciales $x$ e $y$, es:

```
r.slope.aspect elevation=colDEMTOT@maset
slope=dDEM@juan aspect=dDEmaspect@mapset
dx=DEMdx@mapset dy=DEMmapset
```

La figura 4.17 muestra la imagen resultante del cálculo de pendientes.

### Curvas de nivel

A veces será necesario tener referencia de los cambios en la altitud, para esto podemos usar el módulo de *curvas de nivel*. El comando `r.contour` crea un mapa vectorial de dichas curvas a partir de un mapa digital de elevación. Acepta distintos tipos de parámetros para delimitar el rango en el cúal va a crear las curvas, así como la distancia (en las unidades de la región) entre un nivel y otro, a esta distancia se la llama *paso (step)*. Si aplicamos este comando al mapa de elevación, la orden sería la siguiente:

```
 r.contour input=colDEMTOT@mapset
 output=colimaCurvasNivel step=50
```

Por omisión los valores mínimos y máximos que toma para hacer las curvas son los mínimos y máximos del mapa de elevación. Aquí utilicé una distancia (tamaño de paso) de 50 metros entre curva y curva. La figura 4.18 muestra el resultado de este mapa vectorial sobre el mapa de elevación digital.

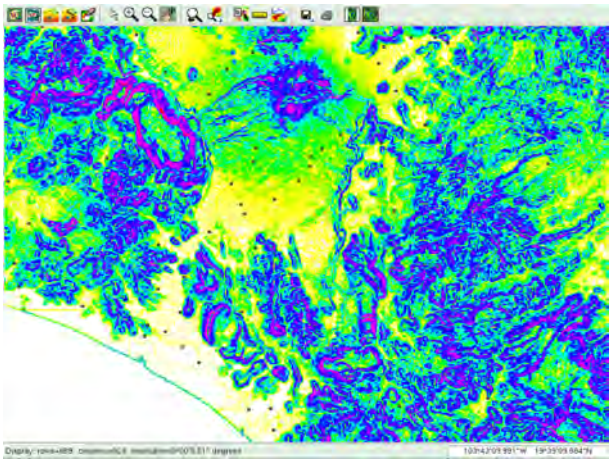


Figura 4.17: Mapa resultante del cálculo de pendientes realizado con `r.slope.aspect`. Los colores oscuros muestran como las pendientes son más abruptas y los colores claros muestran pendientes ligeras.

### 4.2.8. Instalación de otros módulos fuera de la instalación base

Hemos revisado sólo algunas herramientas de todas las que tiene GRASS disponible en la instalación estándar. Cabe señalar que además de estos módulos base se tiene una cantidad grandísima de módulos extra disponibles en el sitio de GRASS. Para ver la lista de todos estos módulos se puede visitar el sitio:

```
http://grass.osgeo.org/wiki/GRASS_AddOns
```

La forma de instalación de cada módulo puede ser diferente y puede depender de bibliotecas no instaladas en el sistema. Pero básicamente el proceso de instalación es así:

1. Bajar el código fuente del módulo de interés en el sitio de GRASS.
2. Revisar si se tienen las bibliotecas necesarias para usar el *plugin* con:

```
./configure [parámetros opcionales] [11].
make libs
```

3. Cada módulo descargado debe venir con un archivo *Makefile* de compilación. Para compilarlo sólo ejecútese:

```
make MODULE_TOPDIR=/ruta/a/grass64/
```

4. La instalación, en ocasiones puede solicitar la cuenta del administrador, corre con el comando:

```
make MODULE_TOPDIR=/ruta/a/grass64/ install
```

[11] ver archivo README dentro del archivo descargado

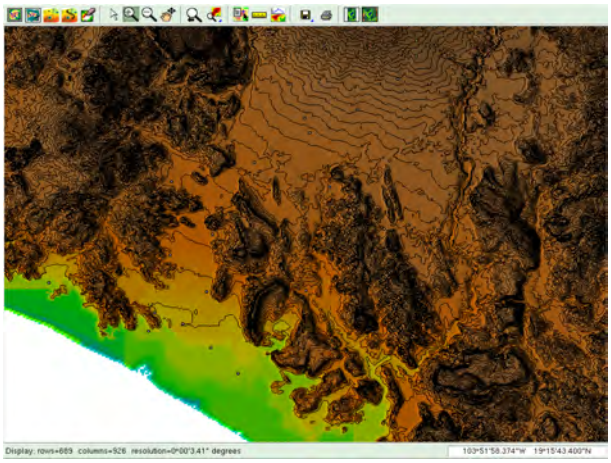


Figura 4.18: Mapa resultante del cálculo de curvas de nivel con el comando `r.contour`, entre cada nivel hay 50 metros de diferencia en valores de altitud.

Esta fue una instalación básica estándar. Cada módulo puede ser un poco diferente. La forma de instalación de cada módulo debe estar documentada en el archivo README del mismo. Desde ahora podemos realizar bastantes tipos de análisis sólo con las herramientas que hemos visto. La tarea siguiente será la resolución de problemas reales y concretos.

¡Libre viene de Libertad¡

# Bibliografía


Rocío M. Alanís-Anaya. *Impacto volcánico en la vegetación en Barranca Monte Grande en el Volcán de Colima*. 2010. En proceso.

Tom M. Apostol. *Calculus*, volumen II. Editorial Reverté, Barcelona, segunda edición, 2001.

Héctor Facundo Arena. *202 Secretos de Linux*. Manuales USERS, primera edición, 2007. ISBN 9789871347414.

Itzhak Benenson y Paul M. Torrens. *Geosimulation: automata-based modeling for urban phenomena*. John Wiley & Sons Ltd, 2006. ISBN 9780470843499.

Barry Boehm, Bradford Clark, Ellis Horowitz, Ray Madachy, Richard Shelby, y Chris Westland. Cost models for future software life cycle processes: Cocomo 2.0. *Annals of Software Engineering*, 1995.

Hamish Bowman. Conway's way of life implentation on grass [En línea]. 2008 [Última consulta: 14 febrero 2010]. URL `http://grass.osgeo.org/wiki/GRASS_AddOns#r.game_of_life`. Es un *script* para el shell que implementa el Juego de la Vida clásico de John Conway. Su objetivo es demostrar lo fácil que es programar automatas celulares en GRASS incluyendo ráster en tres dimensiones y visualización con técnicas de series de tiempo.

P.A. Burrough y R.A. McDonnell. *Principles of Geographical Information Systems*. Oxford University Press, 1998.

Gérard Cliquet. *Geomarketing: Methods and Strategies in Spatial Marketing (Geographical Information Systems Series (ISTE-GIS))*. Wiley-ISTE, 2006.

Comisión Nacional para el Conocimiento y Uso de la Biodiversidad (CONABIO). *Densidad de población por entidad federativa, 2000*. Datos tomados del XII Censo de Población y Vivienda 2000 del Instituto Nacional de Estadística Geografía e Informática (INEGI), México, 2006. Formato de representación geográfica: Shapefile. Formato vectorial compuesto por 4 archivos (shp, shx, dbf,prj)
Escala: 1:1000000 Modelo Elipsoidal: WGS84. Sin proyectar.

Comisión Nacional para el Conocimiento y Uso de la Biodiversidad (CONABIO), Comisión Nacional de Áreas Naturales Protegidas (CONANP), y The Nature Conservancy Program México. Sitios prioritarios terrestres para la

conservación de la biodiversidad. 2008. Formato de representación geográfica: Shapefile. Formato vectorial compuesto por 4 archivos (shp, shx, dbf,prj) Escala 1:1000000 Proyección Cónica Conforme de Lambert WGS84. (EPSG:48402).

Peter H. Dana. Geodetic datums overview, 1995 [Última consulta: 28 agosto 2009]. URL `http://www.colorado.edu/geography/gcraft/notes/datum/datum_f.html`.

Instituto Nacional de Estadistica y Geografía. Servicio de ortofotos con resolución de 3 metros [En línea]. 2009 [Última consulta: 13 de enero de 2010]. URL `http://mapserver.inegi.gob.mx/geografia/espanol/prodyserv/ortofotos/ortofotos.cfm?c=718`.

Michael N. DeMers. *Gis for Dummies*. Wiley Publishing, Inc, primera edición, 2009.

Gerald Evenden, Comunidad del Software Libre, y USGS. Proj.4 - cartographic projections library [En línea, Última consulta: 29 de septiembre, 2009]. URL `http://trac.osgeo.org/proj/`.

Stephen Figgins y Robert Love. *Linux in a Nutshell*. Oreilly, July 2000. ISBN 0596009305.

Free Software Foundation. General public license [En línea, Última consulta: 8 de octubre, 2009]. URL `http://www.gnu.org/licenses/licenses.html`.

Seeber G. *Satelite Geodesy*. Walter de Gruyter, 1993.

Jesús M. González-Barahona, Miguel A. Ortuño Pérez, Pedro de las Heras Quirós, José Centeno González, y Vicente Matellán Olivera. Counting potatoes: the size of debian 2.2. *Upgrade Magazine*, 2001. http://pascal.case.unibz.it/retrieve/3246/counting-potatoes.html.

GRASS Development Team. *Geographic Resources Analysis Support System (GRASS GIS) Software*. Open Source Geospatial Foundation, USA, 2009. URL `http://grass.osgeo.org`.

Jonathan Iliffe. *Datums and Map Projections.For Remote sensing, GIS,and survey*. Whittles Publishing Services, primera edición, 2003.

(INEGI) y Instituto Nacional de Estadística y Geografía. *Polígono de localidades urbanas geoestadísticas*. Datos tomados del XII Censo de Población y Vivienda 2000, (INEGI), México, 2009. Formato de representación geográfica: Formato vectorial Shapefile.

Instituto de Biología. Unidad de Informática para la Biodiversidad (UNIBIO) [En línea]. 2008 [Última consulta: 26 enero 2010]. URL `http://unibio.unam.mx/`.

Instituto de Geografía. Unidad de Informática Geoespacial (UNIGEO) [En línea]. 2008 [Última consulta: 26 enero 2010]. URL `http://www.unigeo.igeograf.unam.mx/unigeo/index.php/Ultimas/Unidad-de-Informatica-Geoespacial.html`.

Estadística e Informática (INEGI) Instituto Nacional de Geografía. Sistema de Descarga del Continuo de Elevaciones Mexicano [En línea]. 2010 [Última consulta: 16 Febrero 2010]. URL `http://mapserver.inegi.org.mx/DescargaMDEWeb/?s=geo&c=977`.

Paul A. Longley, Michael F. Goodchild, David J. Maguiere, y David W. Rhind. *Geographic Information Systems and Science*. John Wiley & Sons,Ltd, segunda edición, 2005.

Martin Lutz y David Ascher. *Learning Python*. Oŕeilly, CA, first edición, 1999.

Hector Resendiz López. *Los sistemas de información geográfica y su aplicación en la ingenieria civil*, 2004. Tesis Licenciatura (Ingeniero Civil) -UNAM Facultad de Estudios Superiores Acatlán.

Silvia Madrid, Arturo Olvera, y Jorge Viveros. Mapas y proyecciones, 1998 [Última consulta: 30 agosto, 2009]. URL `http://www.fenomec.unam.mx/publicaciones/notas/002.pdf`.

M.Neteler y H. Mitasova. *Open Source GIS: A GRASS GIS Approach*. Número 773 in SECS. Kluwer Academic Publishers / Springer, Boston, third edición, June 2008.

Rachel Muheim, Susanne Akesson, y Thomas Alerstam. Compass orientation and possible migration routes of passerine birds at high arctic latitudes. *Oikos*, 103(2):341–349, 2003.

Antonio Hernandez Navarro. Georreferencia de la información [Última consulta: 1 febrero, 2009]. URL `http://infoteca.semarnat.gob.mx/reunion_nal/Dwnl/Georreferencia%20de%20la%20Informaci%C3%B3n.doc`.

Open Geospacial Consortium. Opengis web map service (wms) implementation specification [En línea]. 2009 [Última consulta: 13 de enero de 2010]. URL `http://www.opengeospatial.org/standards/wms`.

Quantum GIS Development Team. *Quantum GIS Geographic Information System*. Open Source Geospatial Foundation, 2009. URL `http://qgis.osgeo.org`.

Gary E. Sherman. *Desktop GIS. Mapping the Planet with Open Source Tools*. Pragmatic Bookshelf, first edición, October 2008.

Gary E. Sherman, Tim Sutton, Radim Blazek, Stephan Holl, Otto Dassau, Tyler Mitchell, Brendan Morely, Lars Luthman, Godofredo Contreras, Magnus Homann, y Martin Dobias. *Quantum GIS. User and installation guide*. USA, 2009. URL `http://download.osgeo.org/qgis/doc/manual/qgis-0.9.1_user_guide_es.pdf`.

Richard Stallman. Free unix ! [En línea, Última consulta: 8 de octubre, 2009]. URL `http://groups.google.com/group/net.unix-wizards/msg/4dadd63a976019d7`.

Richard Stallman. Anuncio inicial. *Usenet:!mit-eddie!RMSOZ , !mit-vax!RMSOZ*, 1983. Anuncio original del Proyecto GNU.

Reto Stockli, Erick Vermone, Nazmi Saleous, Robert Simmon, y David Herring. The blue marble next generation - a true color earth dataset including seasonal dynamics from modis [En línea]. 2005 [Última consulta: 15 de enero de 2010]. URL `http://earthobservatory.nasa.gov/Features/BlueMarble/`.

Linus Torvalds. Free minix-like kernel sources for 386-at [En línea, Última consulta: 8 de octubre, 2009]. URL `http://groups.google.com/group/comp.os.minix/msg/2194d253268b0a1b`.

UNAM. Portal de la Reserva Ecológica Pedregal de San Ángel. [En línea]. 2007 [Última consulta: 17 enero 2010]. URL `http://www.cic-ctic.unam.mx:31101/reserva_ecologica`.

USGS Mapping Applications Center. Map projections [En línea]. 2000 [Última consulta: 23 agosto, 2009]. URL `http://egsc.usgs.gov/isb/pubs/MapProjections/projections.html`.

Frank Warmerdam. GDAL - Geospatial Data Abstraction library [En línea]. 1998-2010 [Última consulta: 28 enero 2010]. URL `http://www.gdal.org/`. Licencia: *Open Source*, X11/MIT.

David A. Wheeler. More than a gigabuck: Estimating gnu/linux's size. *En linea: http://www.dwheeler.com/sloc/redhat71-v1/redhat71sloc.html*, 2001.

Sam Williams. *Free as in Freedom: Richard Stallman's crusade for Free Software*. OŔeilly & Associates, March 2002.

2000 XII Censo de Población y Vivienda. Catálogo de entidades federativas, municipios y localidades [En línea]. 2005 [Última consulta: 24 enero 2010]. URL `http://mapserver.inegi.org.mx/mgn2k/?s=geo&c=954`.

## .1. Apéndice A

# Comandos básicos de la consola Bash

- **adduser** Se utiliza para añadir un usuario. En ese momento, no sólo se creará la cuenta del usuario, también su directorio de trabajo, un nuevo grupo de trabajo que se llamará igual que el usuario y añadirá una serie de ficheros de configuración al directorio de trabajo del nuevo usuario.
  Sintaxis: `adduser [NOMBRE_ USUARIO]`
- **apt-cache search (texto)** Busca en los repositorios paquetes que lleven el nombre o estén relacionados con texto que se introduce como parámetro y da una breve descripción de cada uno.
  Sintaxis: `apt-cache search [PAQUETE]`
- **apt-get dist-upgrade** Actualiza todos los paquetes instalados en el sistema a la versión más nueva que haya en los repositorios.
  Sintaxis: `apt-get dist-upgrade`
- **apt-get install (paquetes)** Instala paquetes y sus respectivas dependencias.
  Sintaxis: `apt-get install [PAQUETE]`
- **apt-get remove (paquete)** Borra paquetes. Con la opción `{purge` borramos tambien la configuración de los paquetes instalados.
  Sintaxis: `apt-get remove [PAQUETE]`
- **apt-get update** Actualiza la lista de paquetes disponibles para instalar.
  Sintaxis: `apt-get update`
- **at** Realiza un tarea programada una sola vez.
  `Sintaxis: at [-lr] hora [fecha].`
- **bash, sh** Abre una terminal tipo bash, sh, csh o ksh.
  `Sintaxis: bash sh ksh csh.`
- **cal** Muestra un calendario.
  `Sintaxis: cal [[mes] año].`
- **cat** Muestra el contenido del archivo en pantalla en forma continua, el cursor regresará una vez mostrado el contenido de todo el archivo. Permite concatenar uno o mas archivos de texto.
  `Sintaxis: cat nom_archivo.`
- **cd** Cambia de directorio.
  `Sintaxis: cd nom_directorio.`
- **chmod** Cambia los permisos de lectura, escritura o ejecución de los archivos. r:lectura w:escritura x:ejecución +: añade permisos -:quita permisos u:usuario g:grupo del usuario o:otros
  `Sintaxis: chmod -c ugo +r-wx permisos nom_archivo`
- **clear** Limpia la pantalla y coloca el cursor al principio de la misma.
  `Sintaxis: clear.`

- **cmp** Compara dos archivos, línea por línea.
  `Sintaxis: cmp nom_archivo1 nom_archivo2.`

- **cp** Copia archivos en el directorio especificado.
  `Sintaxis: cp nom_archivo nom_directorio.`

- **date** Da el año, mes, día, hora, minutos y segundos.
  `Sintaxis: date.`

- **deluser** Elimina la cuenta de algún usuario. Hace referencia a distintos comandos para que también lo borre de grupos de trabajo, directorios personales, permisos, etcétera.
  `Sintaxis: deluser nom_usuario.`

- **df** Reporta el uso del sistema de archivos en disco duro. Muestra los dispositivos montados.
  `Sintaxis:df`

- **dmesg** Muestra los mensajes del núcleo durante el inicio del sistema.
  `Sintaxis: dmesg.`

- **dpkg-reconfigure (paquete)** Volver a reconfigurar un paquete ya instalado.
  `Sintaxis: sudo dpkg-reconfigure [PAQUETE]`

- **du** Estima la cantidad de espacio en disco utilizado en algún directorio.
  `Sintaxis: du`

- **echo** Muestra un mensaje por pantalla.
  `Sintaxis: echo "Cadena de caracteres".`

- **eject** Si el dispositivo no está en uso permite expulsar un disco óptico.
  `Sintaxis: eject.`

- **exit** Cierra las ventanas, conexiones remotas establecidas o consolas abiertas. Antes de salir es recomendable eliminar todos los trabajos o procesos que se estén ejecutando en la computadora.
  `Sintaxis: exit.`

- **file** Determina el tipo del o los archivo(s) indicado(s).
  `Sintaxis: file nom_archivo.`

- **find** Busca los archivos que satisfagan la condición dada en la carpeta indicada.
  `Sintaxis: find nom_directorio o nom_archivo condición.`

- **finger** Da información relacionada con las sesiones de algún usuario.
  `Sintaxis: finger usuario.`

- **free** Muestra información sobre el estado de la memoria del sistema.
  `Sintaxis: free.`

- **ftp** Protocolo de Transferencia de Archivos, permite transferir archivos entre computadoras.
  `Sintaxis: ftp maquina_remota`
  `directorio local`

- **grep** Busca patrones en archivos. Escribe en salida estándar aquellas líneas que concuerden con una cadena de caracteres.
  `Sintaxis: grep [-cilnv] expr nom_archivos.`

- **gzip** Comprime un archivo utilizando la extensión .gz .
  `Sintaxis: gzip nom_archivo.`

- **head** Muestra las líneas de un archivo de texto arriba para abajo.
  `Sintaxis: head -count nom_archivo.`

- **history** Lista los 300 comandos más recientes que se hayan ejecutado en la consola.
  `Sintaxis: history`

- **ifconfig** Administra la configuración de las interfaces de red.
  `Sintaxis: sudo ifconfig`

- **kill** Termina un proceso en ejecución. Se le debe introducir el *identificador de proceso* (PID). Este se obtiene con el comando *top* (ver más adelante). Kill (pid) termina un proceso y Kill -9 (pid) forza a terminar un proceso en caso de que la opción anterior falle.
  `Sintaxis: kill [opciones] PID..`

- **ln** Sirve para crear enlaces a archivos, es decir, crear un fichero que apunta a otro. En Windows se llaman "enlaces directos".
  `Sintaxis: ln [-s] nom_archivo nom_acceso.`

- **logout** Cierra la sesión en la consola.
  `Sintaxis: logout.`

- **ls** Da una lista de los archivos y directorios dentro del directorio de trabajo en el cual se ejecute.
  `Sintaxis: ls.`

- **make** Sirve para compilar e instalar programas a partir del código fuente.
  `Sintaxis: make.`

- **man** Ofrece información acerca de los comandos o tópicos de los sistemas UNIX, así como de los programas y librerías existentes.
  `Sintaxis: man [comando].`

- **mkdir** Crea un directorio nuevo.
  `Sintaxis: mkdir nom_directorio.`

- **more** Muestra el contenido de un archivo en pantalla línea por línea. Para salir, presionar q.
  `Sintaxis: more nom_archivo.`

- **mount** En Linux no existen las unidades A: ni C: sino que todos los dispositivos "cuelgan" del directorio raíz. Para acceder a un disco es necesario **montarlo** primero, esto es, asignarle un lugar dentro del árbol de directorios del sistema.
  `Sintaxis: mount -t sistema_de_archivo dispositivo nom_directorio.`
- **mv** Mueve archivos o subdirectorios de un directorio a otro. También cambia el nombre del archivo o directorio.
  `Sintaxis: mv nom_archivo1 ...nom_archivoN nom_directorio.`
- **netstat** Muestra las conexiones y puertos abiertos por los que se establecen las comunicaciones.
  `Sintaxis: netstat.`
- **nice** Permite cambiar la prioridad de un proceso en nuestro sistema.
  `Sintaxis: nice -n prioridad PID.`
- **passwd** Se utiliza para cambiar la contraseña de algún usuario.
  `Sintaxis: passwd nom_usuario.`
- **pstree** Muestra un árbol de procesos.
  `Sintaxis: pstree.`
- **pwd** Muestra el directorio actual de trabajo.
  `Sintaxis: pwd.`
- **reset** Reestablece la consola.
  `Sintaxis: reset.`
- **rm** Remueve o elimina un archivo. Utilizar el sufijo -r para eliminar recursivamente todos los archivos de ese directorio.
  `Sintaxis: rm nom_archivo.`
- **rmdir** Elimina el directorio indicado, el cual debe estar vacío. Utilizar el sufijo -r para eliminar recursivamente todos los archivos de ese directorio.
  `Sintaxis: rmdir nom_directoriofinger .`
- **scp** Copia Segura, se utiliza para enviar archivos de una computadora a otra. Es necesario ser usuario en las dos máquinas. La información viaja encriptada.
  `Sintaxis: scp usuario@servidor:directorio_servidor directorio_local.`
- **sftp** Protocolo de Transferencia Segura de Archivos. La información viaja encriptada.
  `Sintaxis: sftp maquina_remota.`
- **sort** Muestra el contenido de un fichero, pero mostrando sus líneas en orden alfabético.
  `Sintaxis: Sort [opciones] nom_archivo.`
- **ssh (Secure Shell Client)** Interfaz de consola remota. La comunicación entre el usuario y la máquina remota está encriptada para evitar que terceras personas lean la información. Se debe introducir la dirección del

servidor o *host* y el nombre de usuario. Para iniciar sesión es necesario que se introduzca la contraseña del usuario.
`Sintaxis: ssh usuario@maquina_remota.`

- **startx** Inicia el entorno gráfico (servidor X).
  `Sintaxis: startx.`
- **su** Inicio de sesión como super usuario (administrador).
  `Sintaxis: su.`
- **sudo** Ejecuta un comando dado a nombre del super usuario. Es frecuente en Ubuntu.
  `Sintaxis sudo [comando]`
- **tail** Despliega el contenido de un archivo desde la última línea hasta el principio.
  `Sintaxis: tail –count nom_archivo.`
- **tar** Comprime o descomprime archivos y directorios con la extensión .tar
  `Sintaxis: tar –[arg] nom_archivo.tar nom_archivo.`
- **top** Muestra los procesos que se ejecutan en ese momento. Da información de memoria, recursos del CPU, etc.
  `Sintaxis: top.`
- **touch** Cambia la fecha y hora de creación de un archivo o fichero.
  `Sintaxis: touch nom_archivo –d [AñoMesDiaHoraMinutoSegundo].`
- **traceroute** Da la ruta que toma un paquete enviado por la red desde donde sale hasta su destino.
  `Sintaxis: traceroute [opciones] host [tamaño del paquete].`
- **vi** Permite editar un archivo en el directorio actual de trabajo. Es uno de los editores de texto más usado en UNIX. Para mas información referirse al manual del comando.
  `Sintaxis: vi nom_archivo.`
- **view** Es similar al vi, solo que no permite guardar modificaciones en el archivo, es sólo lectura.
  `Sintaxis: view nom_archivo.`
- **wc** Cuenta los caráteres, palabras o líneas de algún archivo de texto.
  `Sintaxis: wc nom_archivo.`
- **whereis** Devuelve la ubicación del archivo especificado, si existe.
  `Sintaxis: whereis nomb_archivo.`
- **who, w** Lista quienes están conectados al servidor. Incluye , nombre de usuario, tiempo de conexión y el equipo desde donde se conecta.
  `Sintaxis: who  w.`
- **whoami** Escribe el nombre de usuario en pantalla.
  `Sintaxis: whoami.`

- **xmessage** Enviar un mensaje al display de otro usuario o al nuestro propio.
  `Sintaxis: xmessage (mensaje)  export DISPLAY=157.92.49.211:0`
  `xmessage Hola!!.`
- & Añadiendo un & al final del comando haremos que ese comando se ejecute en segundo plano dejando líbre la consola para ejecutar otros procesos.
  `Sintaxis: nom_comando&.`

## .2. Apéndice B

# Glosario

**API** Una interfaz de programación de aplicaciones o API (del inglés Application Programming Interface) es el conjunto de funciones y procedimientos (o métodos, si se refiere a programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

**Applet** Es un componente de una aplicación (programa) que se ejecuta en el contexto de otro programa, por ejemplo un navegador web. El applet debe ejecutarse en una aplicación, que le proporciona un programa anfitrión, mediante un plugin. A diferencia de un programa, un applet no puede ejecutarse de manera independiente, ofrece información gráfica y a veces interactúa con el usuario, típicamente carece de sesión y tiene privilegios de seguridad restringidos. Un applet normalmente lleva a cabo una función muy específica que carece de uso independiente. El término fue introducido en AppleScript en 1993.

**Archivo Torrent** BitTorrent es un protocolo diseñado para el intercambio de archivos punto a punto (peer to peer o P2P). El protocolo Bit Torrent fue desarrollado originalmente por el programador Bram Cohen y está basado en software libre. Mediante un cliente BitTorrent se permite a un usuario establecer una conexión tipo P2P para descargar ficheros que otros usuarios (de la misma red de archivos) poseen y que están dispuestos a compartir basándose en la filosofía del mismo BitTorrent (compartir por igual para todos) para facilitar el intercambio de los mismos. Estos programas clientes están disponibles para diversos sistemas operativos para Linux *Ktorrent, Transmission*.

**Áreas iguales** Una proyección es de *áreas iguales* si cada parte, así como el todo, tiene la misma área correspondiente a la parte de la Tierra,en la misma escala proyectada. Ningún mapa puede ser conformal y de áreas iguales.

**Círculos mayores** A los círculos producidos por la intersección de la esfera con un plano cuyo origen sea el origen de la esfera misma se les denomina *círculos mayores*. Para todo puntos $a$, $b$ en la esfera existe un único *círculo mayor* $C$. Tal que $a$ y $b$ estan en $C$. A este segmento de arco le corresponde la mínima distancia entre estos dos puntos. A esta curva se le llama *geodésica*.

**Conformalidad** Un mapa es *conformal* si para todo punto en la proyección, la escala es la misma en cualquier dirección. Esto implica que los meridianos y paralelos intersecan en ángulos rectos. También las formas de áreas pequeñas se preservan.

**CPU** La unidad central de procesamiento o CPU (por el acrónimo en inglés de *Central Processing Unit*), es el componente en una computadora digital que interpreta las instrucciones y procesa los datos contenidos en los

programas de la computadora. Las CPU proporcionan la característica fundamental de la computadora digital (la programabilidad) y son uno de los componentes necesarios encontrados en las computadoras de cualquier tiempo, junto con el almacenamiento primario y los dispositivos de entrada/salida.

**Firmware** Es el intermediario (interfaz) entre las órdenes externas que recibe el dispositivo y su electrónica, ya que es el encargado de controlar a ésta última para ejecutar correctamente dichas órdenes externas.

**Gratícula** La gratícula es el sistema de coordenadas esféricas basado en líneas de latitud y longitud.

**HTTP** Protocolo de transferencia de hipertexto (HTTP, *HyperText Transfer Protocol* ). Es un protocolo orientado a transacciones y sigue el esquema petición-respuesta entre un cliente y un servidor. Al cliente que efectúa la petición (un navegador o un spider) se lo conoce como üser agent"(agente del usuario). A la información transmitida se la llama recurso y se la identifica mediante un URL. Los recursos pueden ser archivos, el resultado de la ejecución de un programa, una consulta a una base de datos, la traducción automática de un documento, etc.

**Mapa Temático** Mapas temáticos son mapas que, basados en los mapas topográficos, representan cualquier fenómeno de la superficie terrestre que sea cartografiable. Persiguen objetivos bien definidos. Hacen referencia a la representación de ciertas características de distribución, relación, densidad o regionalización de objetos reales (vegetación, suelos, geología, etc.), o de conceptos abstractos (indicadores de violencia, de desarrollo económico o de calidad de vida, etc.).

**Memoria Ram** La memoria de acceso aleatorio (en inglés: *random-access memory cuyo acrónimo es RAM*) es la memoria desde donde el procesador recibe las instrucciones y guarda los resultados. Es el área de trabajo para la mayor parte del software de un computador. Muchos de los firmwares están protegidos por Derechos de Autor.

**Ortofotografía** (del griego *Orthós*: correcto, exacto) Es una presentación fotográfica de una zona de la superficie terrestre en la que todos los elementos presentan la misma escala y está libre de errores y deformaciones. Esta fotografías han sido corregidas digitalmente para representar una proyección ortogonal sin efectos de perspectiva, y en la es posible realizar mediciones exactas. A este proceso de corrección digital se le llama ortorectificación. Una ortofotografía (u ortofoto) combina las características de detalle de una fotografía aérea con las propiedades geométricas de un plano.

**Plug-in** Un complemento (o *plug-in* en inglés) es una aplicación que se relaciona con otra para aportarle una función nueva y generalmente muy especifica. Esta aplicación adicional es ejecutada por la aplicación principal e interactúan por medio de la API.

**Repositorios** Depósito o archivo es un sitio centralizado donde se almacena y mantiene información digital, habitualmente bases de datos o archivos informáticos.

**SQL** El Lenguaje de consulta estructurado (SQL Structured Query Language) es un lenguaje declarativo de acceso a bases de datos relacionales que permite especificar diversos tipos de operaciones en éstas. Una de sus características es el manejo de álgebra y cálculo relacional permitiendo efectuar consultas de forma sencilla para recuperar información de interés de una base de datos; así como también hacer cambios sobre ella.

**Tarjeta gráfica** Una placa o tarjeta gráfica, tarjeta de vídeo, tarjeta aceleradora de gráficos o adaptador de pantalla, es una tarjeta de expansión para una computadora, encargada de procesar los datos provenientes de la CPU y transformarlos en información comprensible y representable en un dispositivo de salida, como un monitor o televisor.

**Wiki** Un wiki, o una wiki, es un sitio web cuyas páginas web pueden ser editadas por múltiples voluntarios a través del navegador web. Los usuarios pueden crear, modificar o borrar un mismo texto que comparten.

# .3. Apéndice C

Este código recibe como parámetros (ver líneas 164, 165 y 166): *i)* el nombre del archivo en texto simple de los datos del *Catálogo de entidades federativas, municipios y localidades* publicado por INEGI en 2005 en su página electrónica . *ii)* los índices de las columnas de localización i.e. (*latitud y longitud*) . Convierte éstos puntos a base decimal y regresa un archivo llamado `salida.csv` (ver línea 161 y función `escribe()` definida de 143 - 154 ) listo para ser procesado por cualquier SIG, utilizando como delimitador el símbolo | . Las líneas 4 - 50 son estándar en la documentación , uso y licencia de cualquier programa con intenciones *libres*. Toda línea que empiece con el símbolo # así como toda cadena de texto entre tres comillas dobles (`"""`) son tomadas como comentarios en `Python`.

*Código de* grad2dec.py

```
1  #!/usr/bin/python
2  # -*- coding: utf-8 -*-
3  """
4  ################################################################################
5  NAME:
6          grad2dec,py  -  Convierte grados sexagesimales a decimales en un archivo
7                          de texto separado por tabulaciones.
8
9
10 SYNOPSIS:
11         python grad2dec.py [NOMBRE_DEL_ARCHIVO.EXTENSION] [num_col_LATITUD]
12                            [num_col_LONGITUD]
13
14
15
16
17 DESCRIPTION:
18         Este programa convierte los datos del:
19                 Catálogo de entidades federativas, municipios y localidades
20                 basado en el XII Censo de Población y Vivienda, 2000 publicado
21                 por el Instituto Nacional de Estadística y Geografía, INEGI.
22                 (México).
23
24
25                 num_col:LATITUD    El número de columna correspondiente al
26                                    atributo de LATITUD (Entero).
27                 num_col_LONGITUD   El número de columna correspondiente al
28                                    atributo de LONGITUD (Entero).
29
30
31 AUTHOR:
32         Escrito por Juan Escamilla Mólgora.
33
34 REPORTING BUGS:
35         <molgor@gmail.com>
36
37 COPYRIGHT:
38
39         This program is free software: you can redistribute it and/or modify
40     it under the terms of the GNU General Public License as published by
41     the Free Software Foundation, either version 3 of the License, or
42     (at your option) any later version.
43
44     This program is distributed in the hope that it will be useful,
45     but WITHOUT ANY WARRANTY; without even the implied warranty of
46     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
47     GNU General Public License for more details.
48
49     You should have received a copy of the GNU General Public License
50     along with this program.  If not, see <http://www.gnu.org/licenses/>
51
52 ################################################################################
53
```

```
"""




from sys import argv


ARCHSAL="salida.csv"

DELIMITADOR="|"


def HELP():
        print "Uso: $python grad2dec [ARCHIVO.ext] [entero:NumCol_LATITUD]
                [entero:NumCol_LONGITUD] "
        print  "Verifica archivo de datos para ver los números de
                columnas correspondientes."
        return


def corta(file):
        #Elimina el encabezado
        f=file.readlines()
        TEMP=[]
        ARCHIVO=[]
        for j in f:
                x=j.split("\r")
                TEMP.append(x)
                for k in x:
                        y=k.split("\t")
                        ARCHIVO.append(y)
        hdr=ARCHIVO[0]
        ARCHIVO=ARCHIVO[1:]
        temp=""
        for c in hdr:
                c+=DELIMITADOR
                temp+=c
        temp+="\n"
        #CAMBIEMOS LOS NOMBRES SIN SENTIDO POR ALGO COMPRENSIBLE.
        temp1=temp.replace("Z1","POB_TOT",1)
        temp2=temp1.replace("Z2","POB_MASC")
        temp3=temp2.replace("Z3","POB_FEM")
        tempFIN=temp3.replace("Z119","TOT_VIV_HAB")
        tempFIN.replace("19"," ")
        return ARCHIVO,tempFIN


def convierte(String,bool):
        """La longitud y latitud esta en el lugar 5 y 6
           respctivamente en cada elemento(lista) de Lista"""
        tam=len(String)
        Ssec=String[-3:-1]
        Smin=String[-5:-3]
        #Porque esta contando las comillas
        #print "Smin"
        Sgrad=String[1-tam:-5]
        #print "Sgrad"
        #Arma conversión
        #Dgrad=int(Sgrad)
        Dmin=float(int(Smin)/60.0)
        Dsec=float(int(Ssec)/3600.0)
        DEC=float(Sgrad)+Dmin+Dsec
        if bool==True:
                return DEC
        else:
                DEC*=(-1)
                return DEC

def convTOT(lista):
        for l in range(len(lista)):
                latitud=convierte(lista[l][LATITUD],True)
                longitud=convierte(lista[l][LONGITUD],False)
                 #Falso para hacerlos negativos
```

```
128                 Salida[l][LATITUD]=str(latitud)
129                 Salida[l][LONGITUD]=str(longitud)
130
131 def quitaSimbolos(lista):
132         ListaSalida=[]
133         for l in lista:
134                 temp=[]
135                 for k in l:
136                         z=k.replace("\"","")
137                         x=z.replace("-","0")
138                         w=x.replace("*","-1")
139                         temp.append(w)
140                         temp.append(z)
141                 ListaSalida.append(temp)
142         return ListaSalida
143
144
145
146 def Escribe():
147         SALIDA.write(hdr)
148         for s in Salida:
149                 cadena=""
150                 for c in s:
151                         c+=DELIMITADOR
152                         cadena+=c
153                 #print s
154                 #le quita el último caracter
155                 CAD=cadena[:-1]
156                 #CAD+="\n"
157                 SALIDA.write(CAD)
158
159
160 SALIDA=open(ARCHSAL,"w")
161
162 Salida=[]
163
164
165
166 if len(argv) > 3:
167         ARCH=argv[1]
168         LATITUD=argv[2]
169         LONGITUD=argv[3]
170         try:
171                 LATITUD=int(LATITUD)-1
172                 LONGITUD=int(LONGITUD)-1
173         except:
174                 HELP()
175         else:
176                 try:
177                         file=open(ARCH,"r")
178                 except IOError:
179                         print "No existe el archivo"
180                 else:
181
182                         LISTACORT,hdr=corta(file)
183                         file.close()
184                         Salida=quitaSimbolos(LISTACORT)
185                         convTOT(LISTACORT)
186                         #Salida=quitaSimbolos(Salida)
187                         Escribe()
188                         SALIDA.close()
189 else:
190         HELP()
191
```

# Índice alfabético