

---

# **OSM + IMPOSM + TILEMILL**

*Publicación 2.0 cfp 2014*

**Geoinquietos Valencia**

09 de March de 2015



<b>1. OpenStreetMap, el mapa colaborativo</b>	<b>1</b>
1.1. Introducción a OpenStreetMap . . . . .	1
1.2. Historia de OSM . . . . .	2
1.3. HOT . . . . .	2
1.4. Procedimiento de creación de mapas . . . . .	3
1.5. Obteniendo los datos de OpenStreetMap . . . . .	6
1.6. Referencias y enlaces de interés . . . . .	10
<b>2. Taller de JOSM</b>	<b>11</b>
2.1. Instalación y arranque . . . . .	11
2.2. Cargar datos de referencia . . . . .	12
2.3. Descarga de datos . . . . .	15
2.4. Filtrando la información . . . . .	16
2.5. Digitalizando . . . . .	18
2.6. Añadiendo etiquetas . . . . .	23
2.7. Consejos generales sobre digitalización y etiquetado . . . . .	29
2.8. <i>Plugins</i> de JOSM . . . . .	29
2.9. Guardando el archivo . . . . .	29
2.10. Subir al servidor . . . . .	29
2.11. Ejercicio . . . . .	31
<b>3. Importando datos a PostGIS</b>	<b>33</b>
3.1. Qué es Imposm . . . . .	33
3.2. Características . . . . .	33
3.3. Limitaciones . . . . .	34
3.4. Ejercicio . . . . .	34
3.5. Instalación . . . . .	34
3.6. Obtener el juego de datos . . . . .	35
3.7. Preparando la base de datos . . . . .	35
3.8. Primera importación . . . . .	36
3.9. Flujo de trabajo . . . . .	38
3.10. Modificando el <i>mapping</i> . . . . .	39
3.11. Referencias y enlaces . . . . .	45
<b>4. TileMill, el estudio cartográfico</b>	<b>47</b>
4.1. Iniciando TileMill . . . . .	47
4.2. Secciones . . . . .	47

4.3.	Creando el proyecto . . . . .	47
4.4.	Añadiendo datos . . . . .	48
4.5.	Introducción al lenguaje CartoCSS . . . . .	49
4.6.	Taller . . . . .	54
4.7.	Más sobre el lenguaje CartoCSS . . . . .	61
4.8.	Extra: OSM-Bright . . . . .	63
4.9.	Exportando los mapas . . . . .	66
4.10.	Otras alimañas . . . . .	68
4.11.	Ejercicio . . . . .	70
4.12.	Referencias y enlaces . . . . .	73
<b>5.</b>	<b>¿Qué es Geoinquietos Valencia?</b>	<b>75</b>
<b>6.</b>	<b>Facilitadores</b>	<b>77</b>
<b>7.</b>	<b>Autores</b>	<b>79</b>
<b>8.</b>	<b>Licencia</b>	<b>81</b>

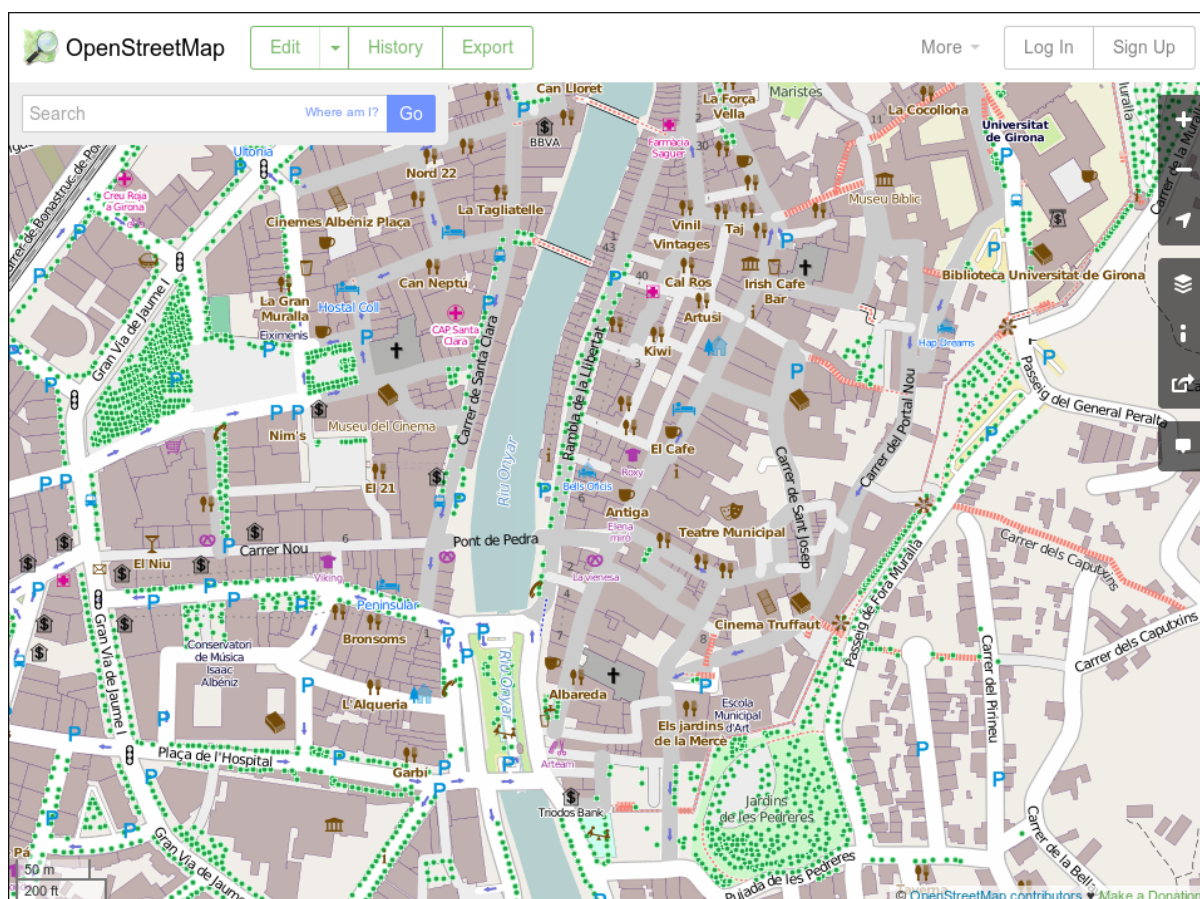


## OpenStreetMap, el mapa colaborativo

### 1.1 Introducción a OpenStreetMap

OpenStreetMap (en adelante OSM) es un proyecto colaborativo para crear mapas libres y editables. Se dice que OSM es a lo mapas, como la Wikipedia a las enciclopedias. Actualmente hay más de 1.500.000 usuarios registrados (estadísticas).

La mejor definición de OSM es que se trata de una **Comunidad de usuarios** que crea una **Base de datos colaborativa** con datos geográficos.



La página principal de OSM es <http://www.openstreetmap.org/> donde puede verse el mapa que se genera con los datos aportados por los usuarios. La comunidad se organiza a través de una wiki cuya dirección es <http://wiki.openstreetmap.org/>

El proyecto es propiedad de la Fundación OpenStreetMap, cuyo objetivo es «*fomentar el crecimiento, desarrollo y distribución de datos geoespaciales libres y a proveer datos geoespaciales a cualquiera para usar y compartir*». Los datos tienen una licencia conocida como [Open Database License 1.0](#) especialmente ideada para publicar bases de datos.

## 1.2 Historia de OSM

El proyecto nace de la mano de [Steve Coast](#) en 2004 que por discrepancias personales con la gestión cartográfica y los precios del organismo británico *Ordnance Survey* decide crear una base de datos cartográfica accesible a todos los públicos.

En 2006 el proyecto toma forma de fundación sin ánimo de lucro y en ese mismo año Yahoo autoriza a la fundación a utilizar su capa de imágenes aéreas de todo el mundo para que los usuarios puedan digitalizar información sobre ellas.

En 2007 la empresa Automotive Navigation Data (AND) dona sus datos de los Países Bajos y de las principales carreteras de la India y China a la fundación y además se incorpora la información de la base de datos [TIGER](#) (Censo de EEUU).

En 2008 la aparece la empresa CloudMade con el objetivo de explotar comercialmente la información del proyecto y que dona a la fundación 2,4 Millones; en ese mismo año la iniciativa pública canadiense GeoBase.ca dona sus datos de Canadá al proyecto.

En 2009 se libera la versión 0.6 de la API y se incrementan en casi 100.000 el número de usuarios duplicando los existentes en solo un años.

En 2010 tiene lugar en Girona la conferencia [State of the Map](#), Bing Maps (Microsost) permite el uso de sus imágenes para digitalizar información y el Ordnance Survey decide liberar sus datos. Terremoto de Haití.

En 2011 se superan los 500.000 usuarios y los 1.000.000.000 nodos.

En 2012 Foursquare abandona el uso de Google Maps y pasa a usar datos de OSM renderizados por MapBox. Se cambia la licencia a ODbL. Apple hace un uso sin atribución de los datos de OSM para sus aplicaciones.

En 2013 se supera el millón de usuarios y tiene lugar el Tifón Yolanda

En 2014 se han superado los 1,5 millones de usuarios.

## 1.3 HOT

### Humanitarian OpenStreetMap Team

Se organiza a partir del terremoto de Haití con el objetivo de proporcionar a los equipos de emergencias mejores mapas de la zona. Su función es servir de puente entre los Actores tradicionales de respuesta humanitaria y la comunidad de OpenStreetMap.

Su labor consiste en recopilar datos, aunque también se realizan trabajos de formación en zonas necesitadas. Aún así, la mayor parte del trabajo es remoto y llevado a cabo por voluntarios.

Actualmente trabajando en Haití, Indonesia, Somalia, Costa de Marfil...

## 1.4 Procedimiento de creación de mapas

Los mapas se realizan siguiendo 3 pasos:

- Toma de datos
- Subida de datos a los servidores de OSM:
  - Edición gráfica de los datos
  - Edición alfanumérica de los datos
- Renderizado de los mapas

### 1.4.1 Toma de datos

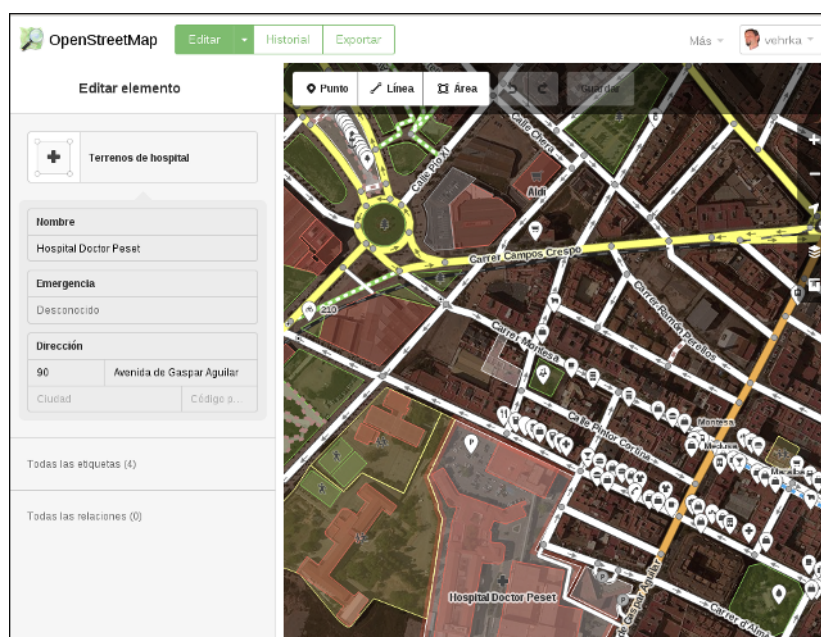
Los datos se recopilan por observación directa, preferentemente empleando GPS, aunque pueden emplearse otros medios como fotografía aérea si los derechos de la imagen lo permite. Aún así el proyecto recomienda conocer y recorrer la zona personalmente para garantizar la máxima calidad del resultado.

Los orígenes más comunes de datos son:

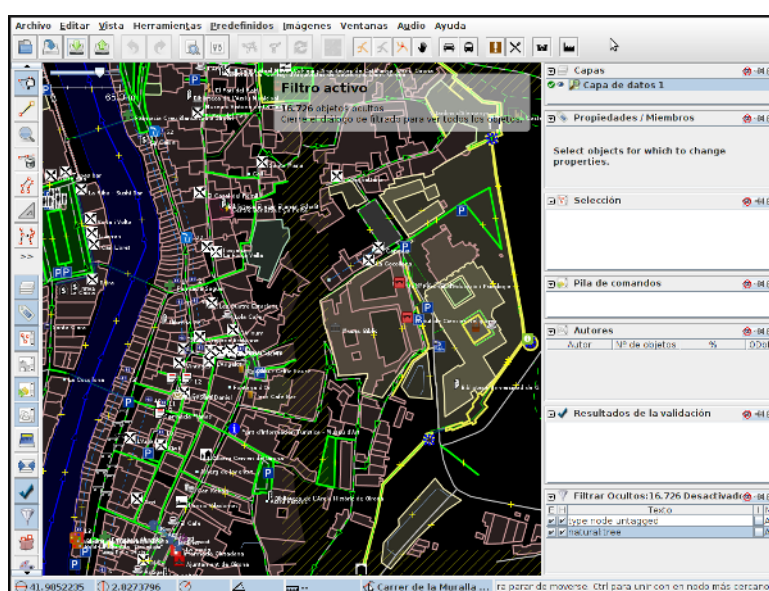
- Trazas GPS, resultado de recorrer la zona usando un dispositivo GPS que almacene dicha información.
  - También suelen usarse *waypoints*, fotos geolocalizadas y archivos de audio geolocalizados
- Imágenes de Yahoo, Bing Maps, el PNOA en España, Landsat y en general cualquier imagen cuyos derechos de autor hayan sido expresamente cedidos, se hayan extinguido o estén en el dominio público.
- Mapas e información de los usuarios. Siempre que se trate de información en el dominio público o cuyos derechos de autor hayan sido expresamente cedidos.
- Información previa existente que requiera ser incluida en un mapa.

### 1.4.2 Subida de datos a los servidores de OpenStreetMap

Una vez recopilada la información, esta debe ser incorporada a la base de datos de OSM. Para ello existen diversos medios, aunque principalmente se emplean clientes web como iD:



y el cliente de escritorio JOSM:



En cualquier caso lo más frecuente es convertir los datos GPS tomados al formato estándar GPX y subirlos posteriormente al repositorio de trazas GPS de OSM de forma que cualquier usuario pueda acceder a dicha información.

## Edición gráfica de los datos

Empleando alguna de las aplicaciones que lo permiten; como iD, Potlach2, JOSM o Merkaartor por ejemplo; se descarga del servidor la porción de información que se quiere editar, para que esta se ajuste a los estándares acordados en el proyecto.

OpenStreetMap solo reconoce 2 tipos de datos gráficos:

- **Nodos:** Son elementos puntuales
- **Vías:** Conexiones lineales entre nodos.



- **Vías abiertas:** Vías que tienen entre 2 y 2000 nodos
- **Vías cerradas:** Vías que empiezan y acaban en el mismo nodo y definen una forma poligonal.
  - **Áreas:** Zonas contenidas dentro de *Vías cerradas*

## Edición alfanumérica de los datos

OpenStreetMap reconoce 2 tipos de datos alfanuméricos:

- **Relación:** Lista ordenada de nodos con un rol, como por ejemplo una restricción de giro.
- **Etiqueta:** Par clave/valor que permite definir atributos.

El modelo de datos alfanuméricos de OSM se basa en el uso de etiquetas *tags* consensuadas por los usuarios a través de la wiki del proyecto.

Las etiquetas se definen por un par clave/valor. Actualmente hay casi 1000 claves “oficialmente” reconocidas y varios centenares propuestos.

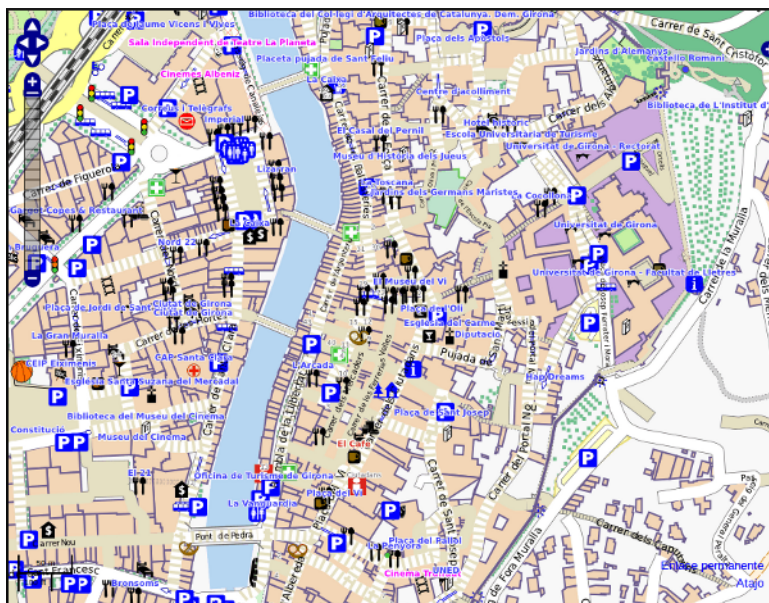
Esta información adicional alfanumérica permite clasificar los datos para que el proceso de renderizado los muestre correctamente representados.

### 1.4.3 Renderizado de los mapas

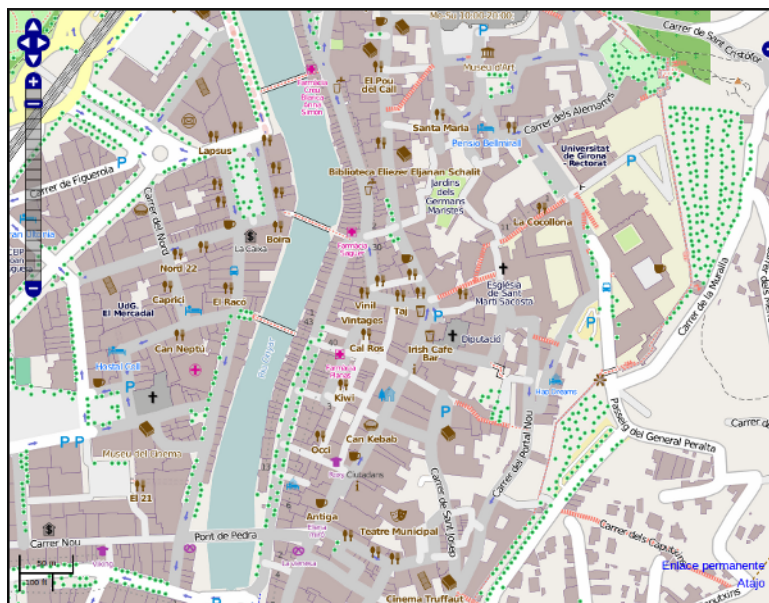
El proyecto OSM tiene varios motores de renderizado tanto en 2D como en 3D que permiten obtener una imagen de la información de la base de datos.

Los principales motores de renderizado son:

- Osmarender En realidad se trata más bien de un conjunto de reglas XLST que genera SVG.



- Mapnik Toma los datos y los carga en un PostGIS para posteriormente renderizar tiles de 256x256. Es el motor de render más utilizado actualmente.



## 1.5 Obteniendo los datos de OpenStreetMap

Daremos un rápido vistazo a la API de OSM y al formato XML de OSM.

### 1.5.1 La API de OSM

La [API](#) de OSM es el único medio de modificar datos de la base de datos. Todas las aplicaciones que quieran obtener datos y subir datos a la base de datos de OSM lo tienen que hacer usando dicha API.

La versión actual de la API es la v0.6 y su uso es obligatorio desde 2009.

La API es una API [RESTful](#) de edición, esto quiere decir que utiliza directamente el HTTP para manipular la información y que recibe los mensajes y resultados en formato XML.

Todas las consultas se realizan de forma anónima, pero las actualizaciones se realizan usando [OAuth](#) (son necesarios un usuario y una contraseña válidos)

La API da soporte de versionado directamente, de forma que todas las actualizaciones quedan registradas con un número de versión de forma que permite detectar errores y conflictos de manera eficiente.

Las descargas están limitadas a cuadrados de 15' de arco y además existe una limitación de ancho de banda, de forma que si se excede la primera limitación el sistema responde un mensaje de error y si se excede la segunda se bloquearán los accesos de manera temporal.

La API no está enfocada a consulta, sino a edición, para consultar la base de datos es más eficiente emplear otros métodos que básicamente consisten en obtener uno de los archivos [Planet](#), convertirlo a una base de datos local y consultar sobre ésta.

### Actualización de datos

Ejemplos de actualización de datos:

```
PUT /api/0.6/changeset/create
PUT /api/0.6/changeset/#id/close
PUT /api/0.6/[N|W|R]/create
DELETE /api/0.6/[N|W|R]/#id
```

Ejemplo de respuesta:

```
<osm>
  <changeset>
    <tag k="created_by" v="JOSM 1.61"/>
    <tag k="comment" v="Just adding some streetnames"/>
    ...
  </changeset>
  ...
</osm>
```

## Otras consultas

Ejemplos de consultas:

```
GET /api/0.6/[N|W|R]/#id/relations
GET /api/0.6/node/#id/ways
GET /api/0.6/[W|R]/#id/full
```

Ejemplo de respuesta:

```
<?xml version="1.0" encoding="UTF-8"?>
<osm version="0.6" generator="OpenStreetMap server">
  <gpx_file id="836619" name="track.gpx" lat="52.0194" lon="8.51807"
    user="Hartmut Holzgraefe" visibility="public" pending="false"
    timestamp="2010-10-09T09:24:19Z">
    <description>PHP upload test</description>
    <tag>test</tag>
    <tag>php</tag>
  </gpx_file>
</osm>
```

## 1.5.2 OSM XML Data: el formato OpenStreetMap

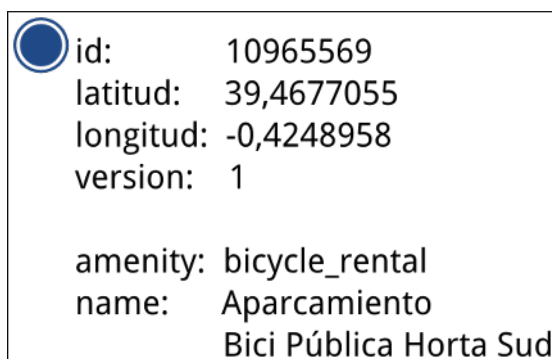
El formato de intercambio estándar de la API es un XML compuesto por combinaciones de los cuatro elementos principales.

### Nodos (Node)

Los Nodos tienen, entre otras informaciones, las siguientes características:

- **id:** el identificador
- **lat y lon:** la posición geográfica en EPSG4326
- **visible:** boolean que determina la visibilidad
- **user:** usuario que creó la versión del nodo
- **timestamp:** marca de tiempo de creación

- **version:** incremental para cada objeto.



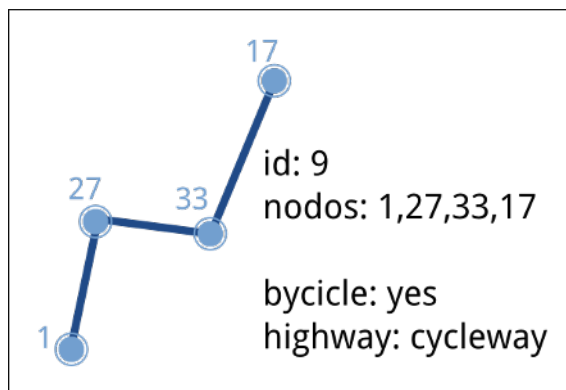
Además el Nodo puede contener información asociada al estilo OSM a través de pares key/value

```
<node id="25496583" lat="51.5173639" lon="-0.140043" version="1" changeset="203496" user="...">
  <tag k="highway" v="traffic_signals"/>
</node>
```

## Vías (Way)

Las Vías son listas ordenadas de nodos que tienen información como:

- **id:** el identificador
- **visible:** boolean que determina la visibilidad
- **user:** usuario que creó el nodo
- **timestamp:** marca de tiempo de creación
- **version:** incremental para cada objeto.



Debe tener una lista de nodos agrupados cada uno con su etiqueta XML *nd* con la referencia id de los nodos que agrupa. Además la Vía puede contener información asociada al estilo OSM a través de pares key/value

```
<way id="5090250" visible="true" timestamp="2009-01-19T19:07:25Z" version="8" changeset="...">
  <nd ref="822403"/>
  <nd ref="21533912"/>
  <nd ref="821601"/>
  <nd ref="21533910"/>
  <nd ref="135791608"/>
  <nd ref="333725784"/>
  <nd ref="333725781"/>
</way>
```



```

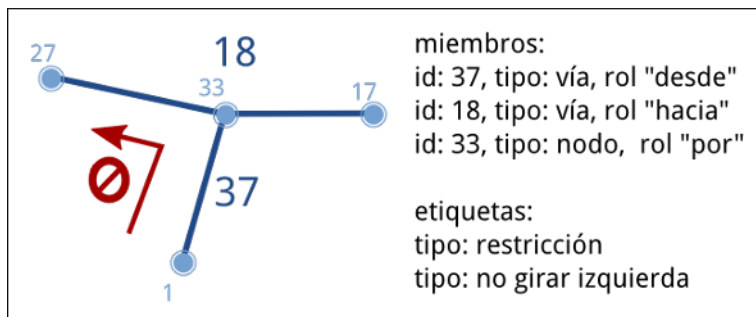
<nd ref="333725774"/>
<nd ref="333725776"/>
<nd ref="823771"/>
<tag k="highway" v="unclassified"/>
<tag k="name" v="Clipstone Street"/>
<tag k="oneway" v="yes"/>
</way>

```

## Relaciones (Relation)

Las Relaciones son listas ordenadas de objetos, son objetos en si mismas y sirven para definir relaciones entre cualquier tipo de objeto. También tienen información como:

- **id:** el identificador
- **visible:** boolean que determina la visibilidad
- **user:** usuario que creó el nodo
- **timestamp:** marca de tiempo de creación



Y además en una etiqueta XML member definir atributos *type*, *id* y *role* que permiten configurar la relación y unas etiquetas tag para describir el tipo de relación.
























```

<relation id="77" visible="true" timestamp="2006-03-14T10:07:23+00:00" user="fred">
  <member type="way" id="343" role="from" />
  <member type="node" id="911" role="via" />
  <member type="way" id="227" role="to" />
  <tag k="type" v="restriction"/>
  <tag k="type" v="no_left_turn"/>
</relation>

```

## Etiqueta (Tag)

Pese a ser una primitiva reconocida por la API de OSM en realidad está integrada dentro de las otras primitivas y nos permite definir los atributos de las mismas.

amenity	school		school and grounds		
amenity	university		a University campus or buildings		
Transportation					
amenity	bicycle_parking		Parking for bicycles		
amenity	bicycle_rental		Rent a bicycle		
amenity	bus_station		Has been replaced by <a href="#">public_transport=station</a>		
amenity	car_rental		Rent a car		
amenity	car_sharing		Share a car		
amenity	car_wash		Wash a car		
amenity	ev_charging		Electric Vehicle Charging Facility		

## 1.6 Referencias y enlaces de interés

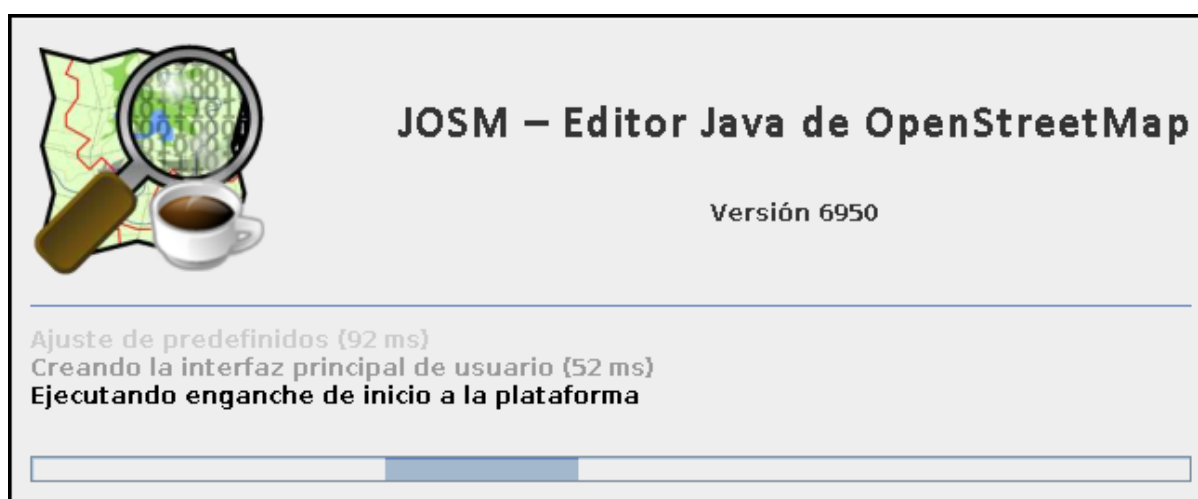
- [Página principal de OpenStreetMap](#)
- [Wiki de OpenStreetMap](#)
- [Información sobre Potlach](#)
- [Información sobre JOSM](#)
- [Información sobre Merkaartor](#)
- [Etiquetas aceptadas por la comunidad OSM:](#)
- [Exportación vía web de OSM](#)
- [API de OSM versión 0.6](#)

---

### Taller de JOSM

---

JOSM es el acrónimo de Java OpenStreetMap Editor, se trata de una aplicación multiplataforma desarrollada por Immanuel Scholz y Frederik Ramm. Es el editor preferido por la comunidad OSM, ya que tiene muchas funcionalidades implementadas y permite editar gran cantidad de datos, aunque su curva de aprendizaje puede resultar un poco pronunciada al inicio.



Una sesión de edición en JOSM suele incluir los siguientes pasos:

1. Carga de datos GPS o el uso de imágenes satélite u ortofotografías
2. Digitalización de información
3. Etiquetado de la información
4. Validación y subida de datos al servidor de OSM

---

**Importante:** Durante la explicación del funcionamiento de JOSM el alumno puede trabajar con cualquier zona y experimentar en zonas sin datos para habituarse a la mecánica de trabajo de JOSM. Una vez revisado el funcionamiento de JOSM se propondrá un ejercicio práctico.

---

### 2.1 Instalación y arranque

La instalación de JOSM es muy sencilla, si tenemos una versión reciente de Java probablemente sea tan fácil como ejecutar el enlace `josm.jnlp` de la [web de JOSM](#). Este método utiliza una característica

llamada *Java Web Start*, y tiene la ventaja de que comprueba al inicio si existe una versión más reciente de JOSM y se actualiza automáticamente.

Si este método no funciona puedes descargar de la misma web la versión actual del enlace `josm-tested.jar` y hacer doble clic en el archivo descargado.

---

**Nota:** Puedes consultar más documentación sobre la instalación y ejecución de JOSM en la sección [Running](#) de las notas de instalación de JOSM.

---

### 2.1.1 Crear un *script* de actualización y arranque en Linux

Si se utiliza un sistema Linux u OSX y no se dispone de soporte para *Java Web Start* se puede crear un *script* como el siguiente para automatizar la descarga de la versión más reciente de JOSM antes de ejecutarlo. Esto resulta interesante porque JOSM se actualiza con mucha frecuencia.

El *script* sería el siguiente:

```
#!/bin/bash

cd /tmp
wget -N http://josm.openstreetmap.de/josm-tested.jar
java -jar josm-tested.jar
```

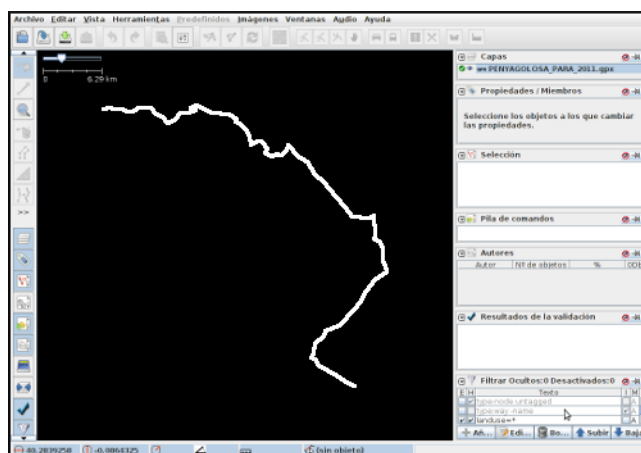
A este *script* bastaría con darle permisos de ejecución y ubicarlo en algún lugar accesible. Al ejecutarlo comprobará si la versión de JOSM es la más reciente y de no ser así la descargará. Después lanzará automáticamente el programa.



## 2.2 Cargar datos de referencia

### 2.2.1 Carga de datos GNSS

JOSM permite cargar información obtenida a través de un receptor GNSS usando para ello el formato de intercambio estandar GPX.

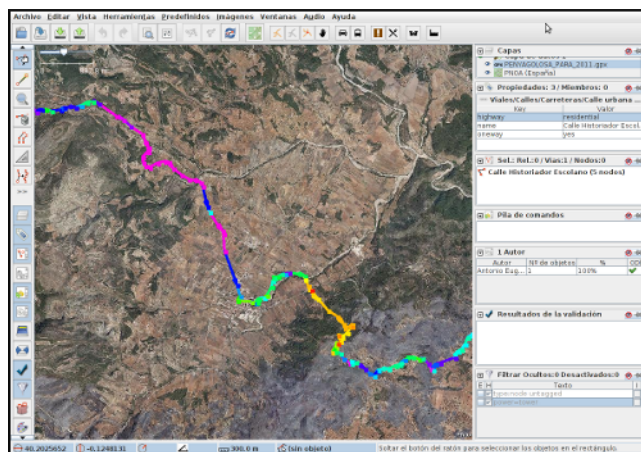


**Importante:** Se recomienda encarecidamente no subir esta información directamente sin depurar o sin tratar, es preferible siempre usarla como base para digitalizar sobre ella y añadir los atributos correspondientes.

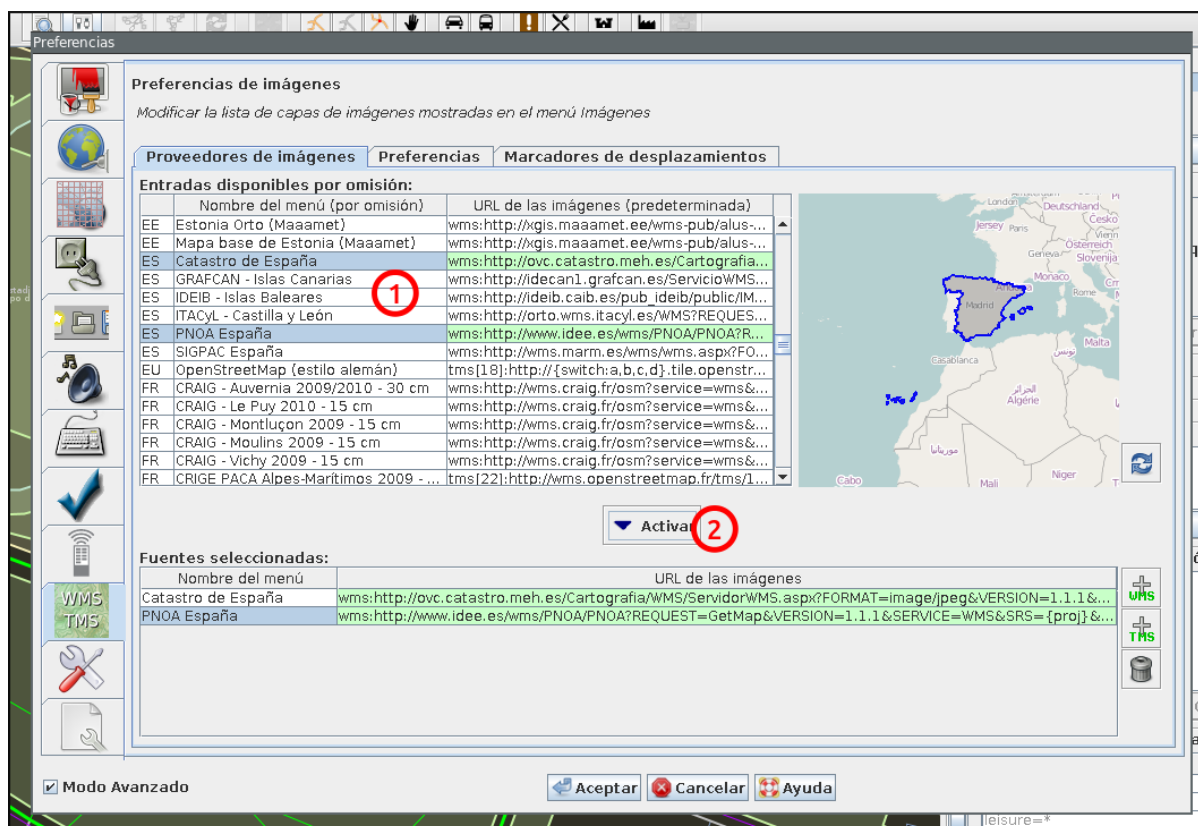
## 2.2.2 Cargar servicios de imágenes



Además de utilizar los datos recogidos en campo con GPS, notas, etc. se pueden también usar imágenes en distintos formatos para usarlas como cartografía de referencia y poder digitalizar sobre ellas.

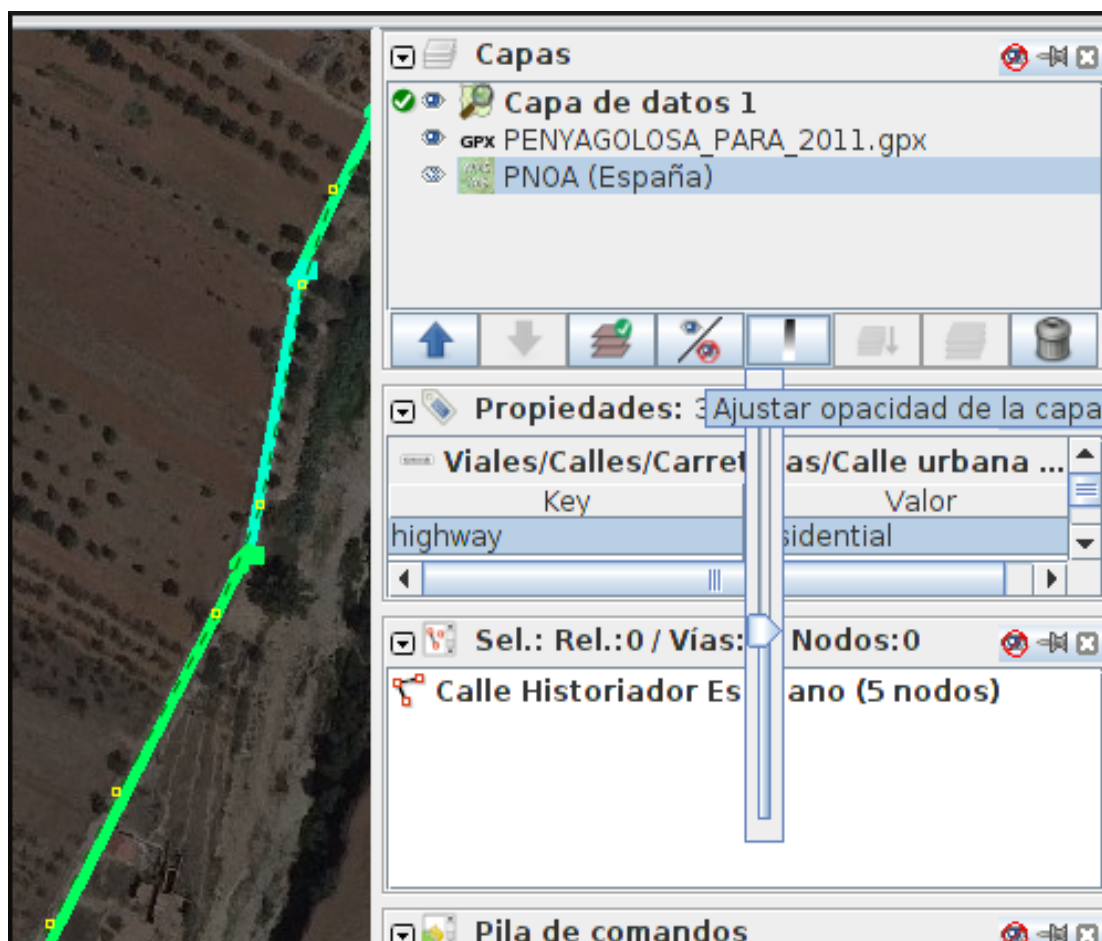
En especial tienen significativa importancia dentro de JOSM la posibilidad de cargar imágenes base provenientes de diversos proveedores a través de Internet cuya información ya viene integrada en el propio JOSM o incluso se pueden agregar nuevos como por ejemplo orígenes de datos WMS o TMS. En España está autorizado el uso del PNOA y del Catastro para digitalizar sobre las ortofotos siempre que se identifiquen el origen y la resolución temporal con las etiquetas `source` y `sourcedate`.



Se puede acceder a la configuración de los proveedores a través del menú *Editar* → *Preferencias* → *WMS/TMS*. Primero se busca el proveedor a partir del código de país y una vez seleccionado se puede hacer clic en el botón *Activar* y confirmar el diálogo.



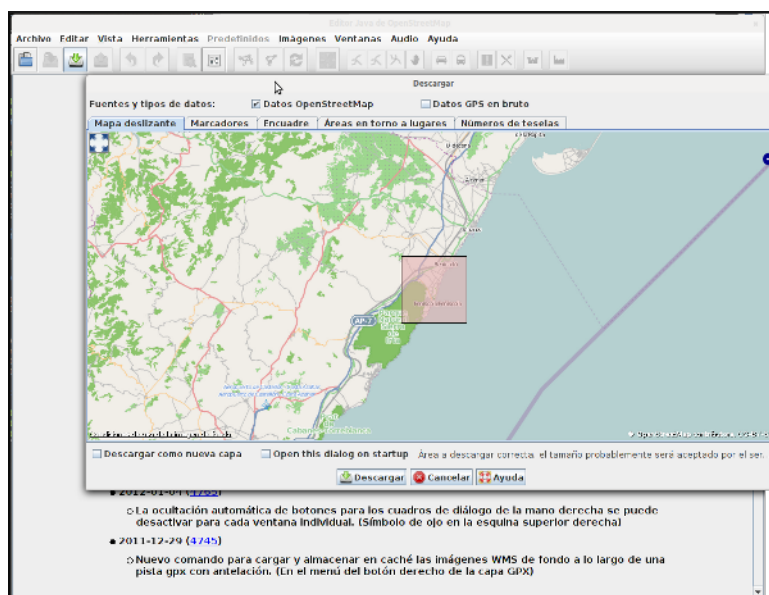
Tras los cambios aparentemente nada habrá cambiado, pero ahora hay una nueva entrada en el menú *Imágenes* y al pulsarla se cargará una capa, debajo de la capa de datos actual, con la ortofotografía de la zona. Es una capa que se puede activar o desactivar , o cambiar la transparencia .



## 2.3 Descarga de datos

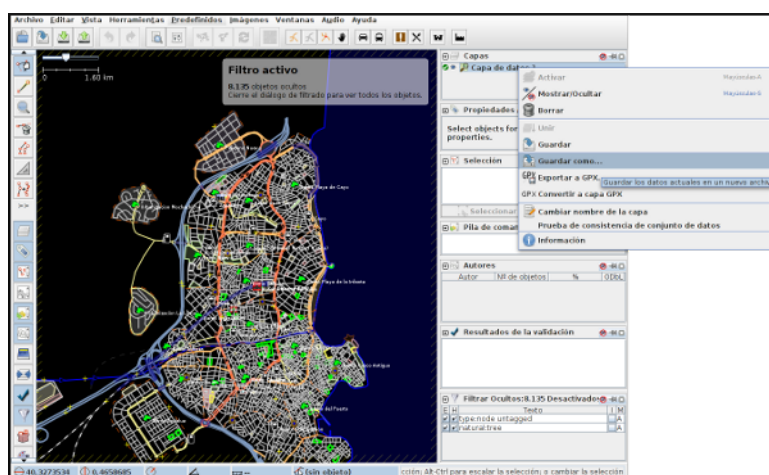
JOSM trabaja por defecto con archivos de formato XML de OSM (archivos .osm). Para obtener un archivo de la zona con la que se quiere trabajar hay que pulsar el botón de *Descarga* de datos del servidor o desde el menú *Archivo* → *Descargar desde OSM*. Al pulsar el botón se muestra una interfaz donde se puede seleccionar la porción de datos que quiere obtenerse.

**Nota:** En este mapa se usan la rueda y los dos botones del ratón: el izquierdo para seleccionar la zona y la rueda y el derecho para desplazarse por el mapa.



El servidor limita las peticiones que cubran gran extensión para no colapsar el servicio, pero si se requiere gran cantidad de datos se pueden realizar diversas peticiones que acabarán almacenándose en un solo fichero.

Una vez seleccionada la zona y aceptada la petición por el servidor, JOSM creará una capa que aparecerá en el panel superior del lado derecho.



## 2.4 Filtrando la información

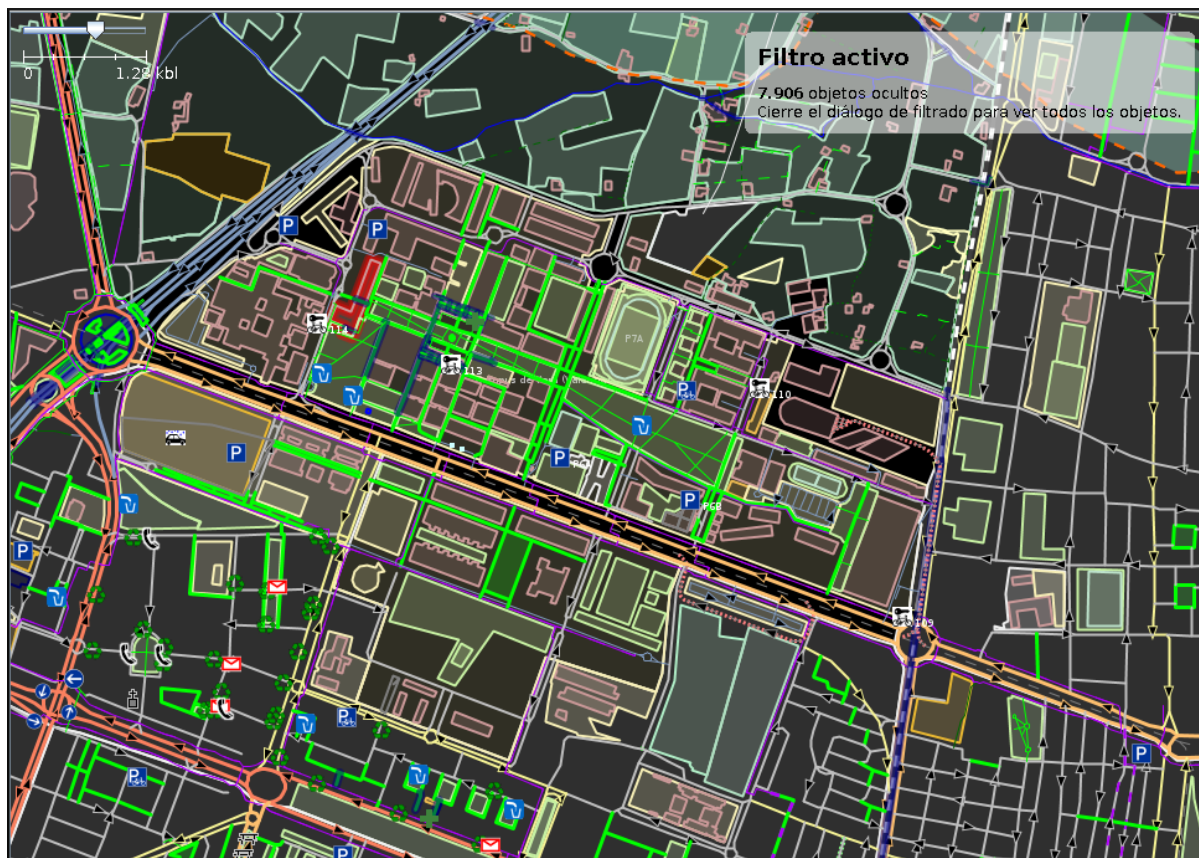
Los filtros son una característica de JOSM que permite ocultar temporalmente elementos cargados en pantalla para tener una mejor visibilidad del área de trabajo descartando aquellos elementos que no nos interesen.

Antes de aplicar un filtro:





Tras aplicar el filtro:



Para definir nuevos filtros se utiliza el panel *Filtrar* que suele encontrarse en la parte inferior del panel

del lado derecho.




Figura 2.1: Filtros en JOSM

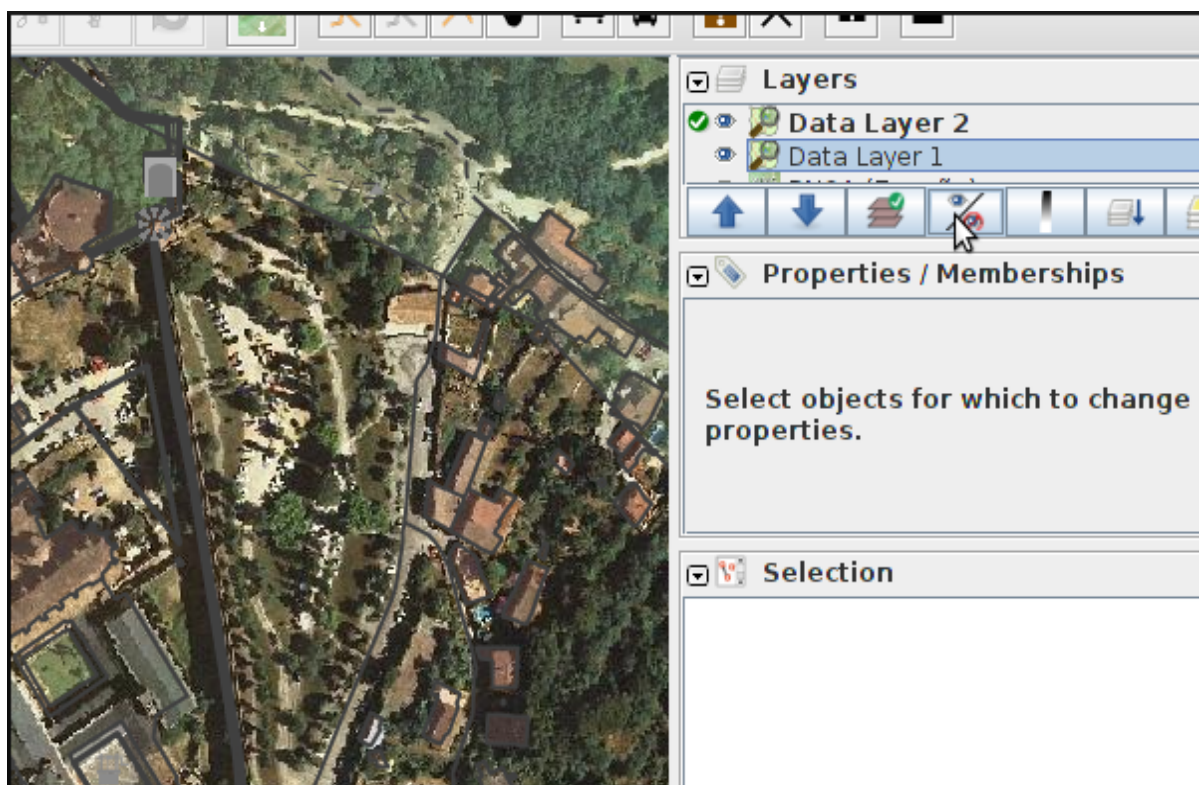
La sintaxis de los filtros es bastante sencilla y al *Añadir* uno nuevo se nos muestra una pequeña guía con ejemplos. Los filtros que se muestran en la figura *Filtros en JOSM* realizan lo siguiente:

- Filtrar todos los nodos que no tengan etiqueta
- Filtrar todos los nodos que tengan la etiqueta *name* sea cual sea el valor de esta
- Filtrar todos los nodos que tengan la etiqueta *amenity* (otra forma de filtrar sin que importe el valor de la etiqueta)
- Filtrar todas las vías que **no** estén cerradas (usando el *check* para invertir la búsqueda)


## 2.5 Digitalizando

Para probar la digitalización crearemos una nueva capa en la que poder trabajar sin modificar los datos que se han descargado, para crear la capa usaremos el menú *File* → *New Layer* o el atajo de teclado `Ctrl + N`.

**Nota:** Al crear la nueva capa, la capa de datos anterior deja de ser la capa de datos activa y aparecerá como líneas de color negro. Es conveniente desactivar la capa para poder ver la ortofotografía, para ello seleccionaremos la capa y pulsaremos en botón de cambiar la visibilidad .



**Nota:** También es recomendable desactivar los filtros pulsando en la casilla *E*.

Para digitalizar un punto, haremos *zoom* sobre una zona con árboles, el *zoom* se controla con la barra que hay arriba a la izquierda, pero también con la rueda del ratón. A continuación pulsamos sobre el botón agregar  o pulsamos la tecla *A* para entrar en el modo de edición.

### 2.5.1 Nodos

Digitalizamos los árboles poniendo un punto, haciendo un solo clic, sobre cada copa de la ortofotografía. JOSM está pensado para añadir elementos lineales por lo que por defecto espera tener que añadir líneas, para añadir tan solo puntos deberemos pulsar la tecla *ESC* después de hacer clic sobre cada árbol.





---

**Nota:** JOSM tiene **muchos atajos**. Si mantienes pulsada la tecla *Shift* no es necesario pulsar la tecla *Esc*, JOSM insertará nodos individuales.

---

De momento en realidad estamos simplemente poniendo los *Nodos*. Para que OSM los reconozca como árboles deberíamos añadir también las *Etiquetas*, como veremos más adelante.

### 2.5.2 Vías

Para digitalizar una vía, buscaremos un nivel de *zoom* que nos permita ver la vía en su totalidad por lo menos una parte muy significativa de ella. Puede que tengamos que desplazarnos por la imagen, pero como estamos en modo edición si hacemos clic con el botón izquierdo añadiríamos un nuevo nodo. Para **desplazarnos** sin salir del modo edición podemos conseguirlo haciendo clic en el botón **derecho** del ratón y movernos por el mapa sin soltarlo.

Para digitalizar la vía vamos marcando nodos de manera consecutiva intentando seguir el eje de esta y respetar la forma siguiéndola sobre la ortofotografía. Es interesante que además pongamos un nodo en cada intersección que tenga la vía, lo que facilitará digitalizar las vías que conectan con ésta más adelante.



Un par de atajos de teclado útiles a la hora de digitalizar vías:

- Pulsar la tecla `Alt` mientras digitalizas vías, te permite hacer que el próximo nodo, aunque esté conectado al nodo anterior, forme una vía nueva.
- Cuando tenemos una vía seleccionada (también funciona con vías cerradas) tener la tecla `Shift` + `Ctrl` pulsada te permite rotar el elemento seleccionado.
- Si pulsamos `Ctrl` + `Alt` podremos cambiar la escala del elemento seleccionada.
- Por último, si mientras digitalizamos hacemos clic con el ratón en el ángulo de la barra inferior (el cuarto elemento) activaremos el modo de ayuda de dibujo de geometrías que nos asiste con ángulos establecidos e intersecciones.

### 2.5.3 Áreas

Las áreas no son más que una vía que empieza y acaba en el mismo punto y tiene una etiqueta que la identifica.

En este ejemplo, digitalizaremos el área de aparcamiento que hay en la zona en la que estamos trabajando, teniendo en cuenta que deberemos cerrar la vía pulsando al final sobre el primer nodo que digitalicemos.





Los edificios son seguramente el caso más típico de áreas a digitalizar.



## El *plugin building tools*

Al activar este plugin se nos muestra un nuevo modo en la barra de herramientas. Este modo permite dibujar edificios de forma muy eficiente al evitar dibujar todos los nodos del mismo. Está pensado para dibujar edificios que tienen una forma rectangular.

Si una vez dibujado el primer edificio lo seleccionamos, los siguientes edificios se crean orientados en la misma dirección, teniendo únicamente que marcar las esquinas opuestas del mismo.

## La herramienta *Crear áreas*

Existen diversas herramientas que permiten manipular las geometrías, una de las más interesantes (especialmente combinada con la anterior) es la de *Crear áreas*. Esta herramienta permite mover una sección de un área hacia fuera o dentro en paralelo a la dirección existente. En la figura *Dibujar un edificio ortogonal* se muestra el proceso.

1. Dibujar un edificio con el modo *Edificio* (en este caso sobre la terraza)
2. Dibujar los puntos de corte de la zona de la fachada a desplazar
3. Mediante la herramienta *Crear áreas* llevar meter la fachada hacia dentro
4. Mover el edificio a su ubicación en la parte inferior

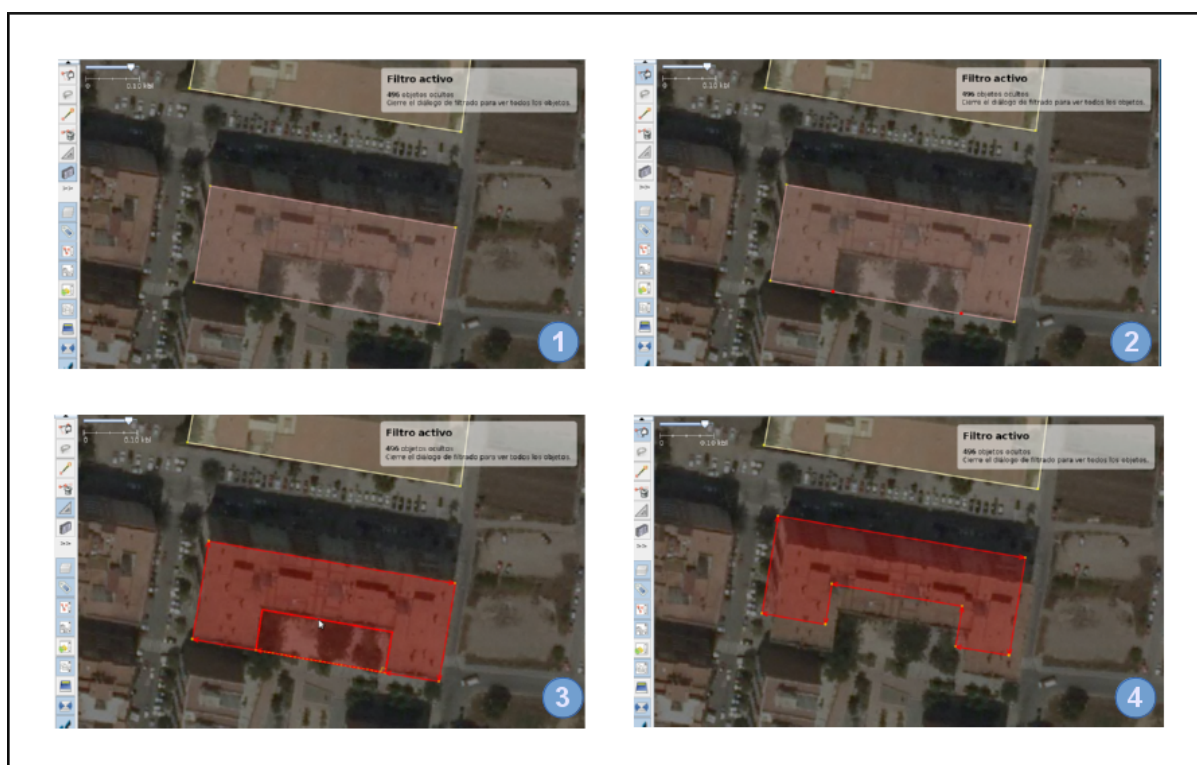




Figura 2.2: Dibujar un edificio ortogonal

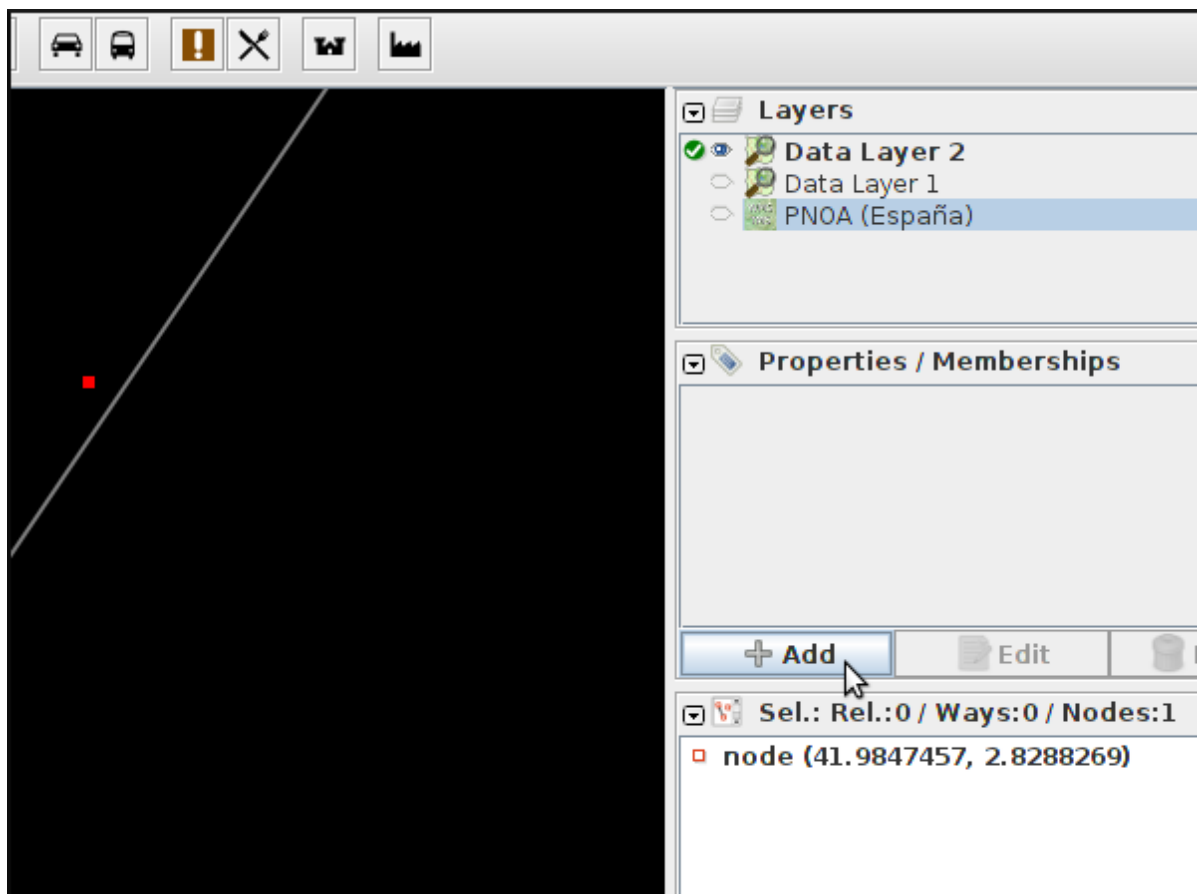
## 2.6 Añadiendo etiquetas

**Nota:** Para el siguiente paso es preferible desactivar la capa del PNOA seleccionándola y pulsando el

botón correspondiente .

Seleccionaremos el primer árbol que hemos digitalizado para lo que hay que entrar en modo selección pulsando el botón selección  o la tecla S y hacemos clic sobre uno de los nodos que representan a los árboles, puede que tengamos que hacer un poco de *zoom*.




Una vez seleccionado, pulsamos el botón Add de la ventana *Properties/Memberships* para poder añadir las Etiquetas correspondientes.



### 2.6.1 ¿Qué etiquetas se emplean para indicar que es un árbol?

Lo mejor **SIEMPRE** es consultar la wiki de OSM donde tienen un [listado de elementos comunes en los mapas Map Features en español](#) y cómo emplearlos. En este caso buscaremos la entrada de árbol en la página y vemos que se corresponde con el par clave/valor `natural/tree`.



natural	stone		Marca la posición de una roca grande y aislada.
natural	tree		Marca la posición de un árbol aislado o significativo por su tamaño.
natural	volcano		Cima de un volcán activo, dormido o extinto.

Pero además si pulsamos sobre la palabra `tree` nos lleva a la entrada específica de la wiki en la que explican las características a tener en cuenta y generalmente se detallan las claves a las que también suelen estar asociadas las entidades a cartografiar e incluso ejemplos.

En definitiva, los árboles suelen etiquetarse usando las siguientes claves:

- *natural* con el valor *tree*
- *name*
- *type*
- *height*
- *name:botanical*

Pueden asignarse etiquetas a grupos de elementos, para lo que primero hay que seleccionarlos manteniendo pulsada la tecla *Mayúsculas* mientras se va haciendo clic; para posteriormente aplicar la etiqueta, según el procedimiento ya visto.

También pueden *copiarse* etiquetas entre elementos, seleccionamos el elemento que tiene las etiquetas y lo copiamos con `Ctrl + C` y después seleccionamos el elemento destino y pulsamos `Ctrl + Shift + V` y le asignará automáticamente las etiquetas del primer elemento.

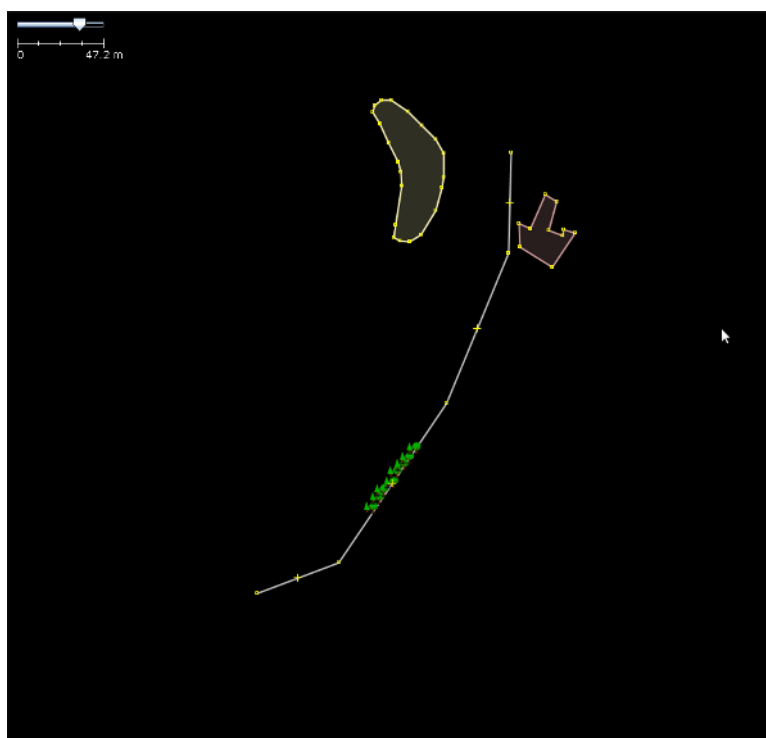
Ahora hay que proceder igual con los demás elementos de nuestro dibujo, tales como carreteras, parings, edificios, etc...

---

**Nota:** El *plugin building tools* ya inserta la etiqueta de edificio por nosotros.

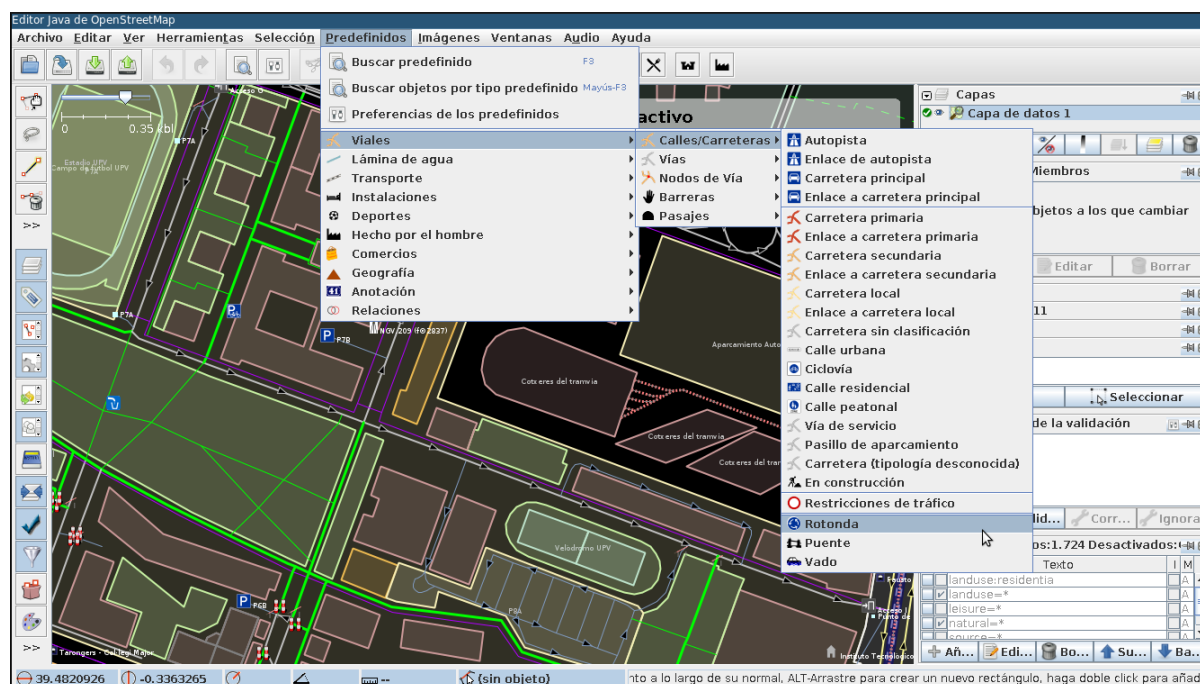
---

Consultaremos los elementos en su página correspondiente y añadiremos las etiquetas que creamos sean necesarias para describir la realidad. El resultado tras aplicar las etiquetas podría ser parecido a este:



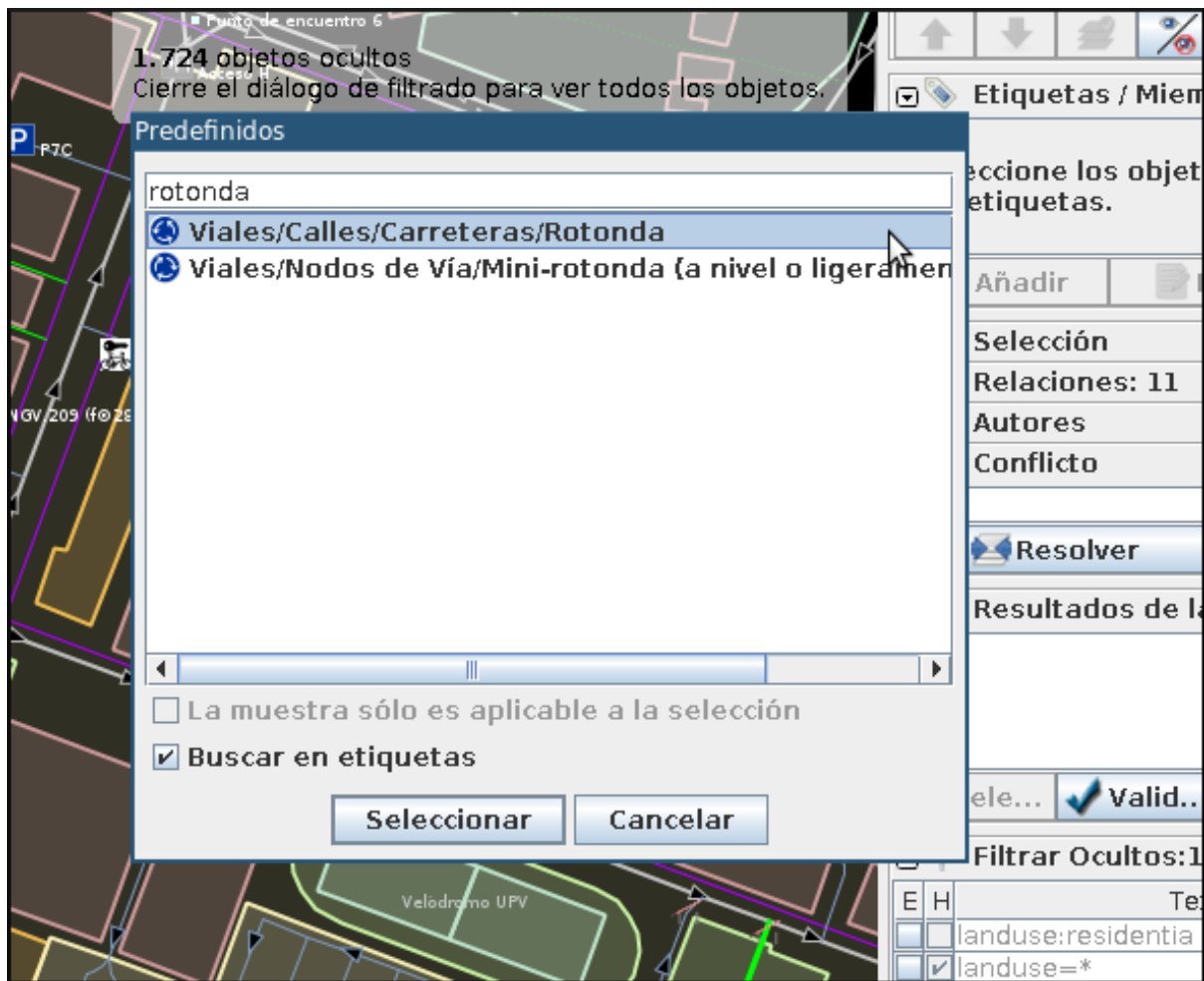
## 2.6.2 Añadir etiquetas predefinidas

Añadir las etiquetas una a una es muy costoso. En lugar de hacerlo de esta forma, JOSM ofrece todo un conjunto de diálogos predefinidos para los tipos de datos más comunes. Para ello una vez añadida o seleccionada la geometría (vía o nodo) podemos usar la entrada de menú *Predefinidos* y navegar por los diferentes tipos de datos. Por ejemplo para dar de alta una rotonda podemos navegar a *Predefinidos* → *Viales* → *Calles/Carreteras* → *Rotonda*.





Si ya sabemos cómo se identifica un elemento en el menú, resulta muy cómodo usar el buscador que se lanza con la tecla F3. Por lo tanto, basta con empezar a escribir *Rotonda* para que el buscador

encuentre nuestra etiqueta.



De cualquiera de estas dos formas, al final llegaremos a un diálogo que ofrece una interfaz para añadir las etiquetas más comunes referidas al tipo seleccionado, usando listas para elegir las opciones más habituales, botones de tipo *check* para indicar características que solo toman un valor e incluso disponemos de un enlace a la documentación ampliada en el wiki del proyecto y un botón que nos permite «anclar» el elemento a la barra de herramientas en caso de que vayamos a usar mucho este tipo de elemento.

Modificar objeto 1

Viales/Calles/Carreteras/Rotonda

Tipo:  ▼

Atributos opcionales:

Nombre:

Capa:  ▼

Carriles:  ▼

☐ Sentido único

☐ Puente

☐ Túnel



☐ Desmonte

☐ Terraplén

Pendiente:  ▼

Anchura (metros):

[Más información sobre esta característica](#)

 Aplicar Predeterminados  Cancelar

### 2.6.3 Especificar las fuentes

Es muy importante identificar los orígenes de datos de la información, ya que es una de las formas de medir la calidad de los datos que almacena OSM.

En España, si se digitalizan datos sobre la ortofotografía del PNOA hay que añadir a **TODOS** los elementos digitalizados el par clave valor *source/PNOA* y a ser posible la clave *source:date* cuyo valor corresponde con la fecha en la que se realizó el vuelo

Otros posibles orígenes de datos válidos para usar en España se pueden encontrar listados en la página web [Spain Datasources](#) de la wiki de OpenStreetMap.

## 2.7 Consejos generales sobre digitalización y etiquetado

**Acude SIEMPRE a la documentación y los expertos** En caso de duda es mejor consultar la wiki primero y si no se encuentra la respuesta acudir a las [lista de correo en español de OpenStreetMap](#)

**Don't map for the render** O lo que es lo mismo, en general y excepto en muy contadas excepciones, no hay que dibujar y etiquetar las cosas «para que queden bonito en el mapa», se debe dibujar y etiquetar *la realidad* o la mejor representación de ella que se pueda conseguir.

**No reinventar la rueda** Hay mucho planeta cartografiado en OpenStreetMap, posiblemente alguien ya haya solucionado el problema de representación de la realidad que se te presenta, muchas veces se aprende más intentando ver cómo han resuelto otros problemas similares, busca sitios donde ocurran los mismos fenómenos que quieras representar y mira como lo han hecho otros.

## 2.8 Plugins de JOSM

JOSM es un *software* en constante evolución. Una de sus características más interesantes es la capacidad para ampliar su funcionalidad utilizando extensiones (conocidos como *plugins*). Se puede acceder a la lista de *plugins* desde el diálogo de preferencias en *Editar* → *Preferencias* o pulsando F12. En este diálogo accedemos a la lista de *plugins* desde la cuarta sección. La primera vez habrá que descargar la lista de extensiones disponibles.

Algunos *plugins* interesantes son:

- **building tools**: añade herramientas para crear edificios de forma muy eficiente
- **imagery offset db**: ofrece una base de datos de correcciones de imágenes que están desplazadas de su ubicación correcta
- **notes**: añade la capa de notas de la web de OSM
- **utils plugin 2**: añade una gran cantidad de nuevas herramientas tanto para la selección como para la creación de entidades.

## 2.9 Guardando el archivo

Puede ser interesante guardar la capa que estamos editando para usarla con otras herramientas que veremos en el curso. Para guardar la capa que estamos editando pulsaremos con el botón derecho del ratón sobre el nombre de la capa y seleccionaremos la opción *Save as...* lo que nos permitirá guardar la información en formato `.osm` que es el formato XML de OpenStreetMap.

## 2.10 Subir al servidor

Por último, si la cartografía fuese de interés para el proyecto, es posible subir los cambios seleccionando la opción de menú *Archivo* → *Subir cambios*. Para poder subir los cambios previamente habrá que introducir las credenciales del usuario del proyecto en el diálogo de preferencias.

Para subir los cambios hay además que indicar un texto que describa el trabajo realizado y es también muy conveniente indicar en el diálogo el origen de los datos que se han usado como referencia.

Subir a 'https://api.openstreetmap.org/api/0.6/'

**9 objetos para añadir:**

- ▣ nodo {39.4894666, -0.3565404}
- ▣ nodo {39.4894953, -0.3567603}
- ▣ nodo {39.4895649, -0.357293}
- ▣ nodo {39.4895987, -0.3575521}
- ▣ nodo {39.4896218, -0.3567326}
- ▣ nodo {39.4896914, -0.3572653}
- ▣ nodo {39.4897743, -0.3564729}
- ▣ nodo {39.4899065, -0.3574846}
- ▣ edificio {0 nodos}

**Modificaciones** | **Avanzado**

**Etiquetas del nuevo conjunto de cambios**

**Preferencias**

**Introduzca un breve comentario sobre los cambios que se van a subir:**




Trabajando en Valencia

**Especifique el origen de datos para los cambios ([obtener de las capas actuales](#)):**

PNOA

Subiendo **9 objetos** a **1 conjunto de cambios** usando **1 petición** ([configuración avanzada](#))

Los objetos son subidos a un **nuevo conjunto de cambios**. El conjunto de cambios va a **cerrarse** después de esta subida ([configurar conjunto de cambios](#))

 **Subir cambios**  **Cancelar**  **Ayuda**

## 2.11 Ejercicio

Como ejercicio se propone que cada alumno del curso elija alguna de las zonas propuestas y haga una sesión de edición que mejore la cartografía de la zona. Las zonas seleccionadas son pequeños municipios de la Comunidad Valenciana que están pobremente cartografiados por lo que cualquier contribución realizada por el alumno será de ayuda. Por supuesto, el alumno es libre de elegir cualquier otra zona de OSM siempre que le permita practicar con la creación de nuevas geometrías, añadir etiquetas, etc.

Se recomiendan las siguientes tareas:

- Dar de alta calles y edificios públicos relevantes (colegios, equipamientos, etc.)
- Dar de alta zonas de uso del suelo: residencial, industrial, parques
- Dar de alta puntos de interés: farmacias, bancos,...
- Revisar nombres y sentidos de las vías

---

**Nota:** Aunque se darán unos datos de partida nuevos para las siguientes secciones del curso, los datos editados podrían usarse como extensión si el alumno así lo desea.

---





---

## Importando datos a PostGIS

---

### 3.1 Qué es Imposm

Se trata de una serie de *scripts* Python que permiten importar datos de OpenStreetMap a una base de datos Postgres. Los archivos a importar deben estar en el formato XML o PBF de OSM y la base de datos debe tener activada la extensión espacial PostGIS.

Su espíritu es optimizar la creación de bases de datos geográficas enfocadas a renderizar o a montar servicios WMS.

Los desarrolladores principales son [Omniscale](#), que es la empresa de Dominik Helle y Oliver Tonnhofer, que también están detrás del proyecto [MapProxy](#).

Funciona en Linux y Mac OS X y es código libre bajo licencia [Apache Software License 2.0](#).

### 3.2 Características

**Esquemas de base de datos personalizados** Crea tablas separadas para cada tipo de dato. Permite crear estilos independientes de manera sencilla y mejora el rendimiento de renderización.

**Soporte para Múltiples CPUs** Está pensado para usar procesos paralelos de manera que distribuye la carga de trabajo entre los CPUs y núcleos del equipo.

**Normaliza valores** Por ejemplo, todos los posibles valores booleanos `1`, `on`, `true` y `yes` se convierten en `TRUE`.

**Soporte para localización de cadenas de texto** Búsqueda personalizable de valores localizados

**Filtro por etiqueta o por valor** La importación es selectiva y configurable

**Cache eficiente de nodos** Para almacenar las calles y las relaciones es necesario almacenar todos los nodos. Imposm usa la base de datos basada en archivo [Tokyo Cabinet](#) que almacena pares clave valor para hacer una *cache* de estos datos. Así se reduce de manera significativa el uso de la memoria.

**Tablas generalizadas** Se pueden crear automáticamente tablas con menor resolución espacial, lo que permite por ejemplo preparar rápidamente renders de grandes redes a bajas resoluciones

**Vistas de uniones** Permite crear vistas que combinen distintas tablas. Por ejemplo podemos disponer de la cartografía de carreteras separada en tablas para autopistas, carreteras principales y calles, pero también una vista que integre todas estas tablas.

## 3.3 Limitaciones

No permite el uso de actualizaciones diferenciales

Solo permite el uso de bases de datos PostGIS, aunque podría implementarse con facilidad su uso con otras como SpatialLite, Oracle, etc.

Aunque es bastante eficiente con el uso de la memoria, las importaciones de datos masivas pueden llevar bastante tiempo: un archivo de 1 GB (comprimido, equivalente a Alemania) en un sistema con 2 GB RAM o Europa entera (~5 GB) en un sistema de 8 GB no darían problemas, pero un planet requerirá de unos 16 GB de RAM o más (tarda unas 20h con 8GB).

## 3.4 Ejercicio

A continuación se detalla una práctica guiada en la que se verán los detalles básicos del manejo de la aplicación Imposm.

Se espera del lector que vaya ejecutando las instrucciones que se detallan a continuación y en caso de duda pregunte al facilitador.

---

**Importante:** El entorno de trabajo que se supone para esta práctica es OSGeo Live 7.0.

---

Para trabajar lo primero que vamos a hacer es crear la carpeta en la que trabajaremos. Abrimos una terminal y cambiamos al directorio tecleando

```
$ cd ~
```

Creamos un nuevo directorio y accedemos al mismo

```
$ mkdir tallerimposm
$ cd tallerimposm
```

## 3.5 Instalación

Lo primero es instalar algunas dependencias del sistema (es probable que ya tengamos instaladas algunas de ellas):

```
$ sudo apt-get install build-essential python-dev protobuf-compiler \
    libprotobuf-dev libtokyocabinet-dev python-psycpg2 \
    libgeos-cl libgdall-dev libspatialindex-dev \
    python-virtualenv tree
```

El siguiente paso depende de si nuestra máquina tiene acceso a Internet por el puerto 443 y por tanto podemos instalar paquetes con `pip` o no. En el segundo caso se ofrece un entorno virtual ya funcional para descarga.

### 3.5.1 Si podemos instalar paquetes en el entorno virtual

Crear el entorno virtual e instalar los paquetes necesarios ejecutando:

```
$ virtualenv venv
$ source venv/bin/activate
(venv)$ pip install imposm rtree
```

### 3.5.2 Si no podemos instalar paquetes en el entorno virtual

Descargamos un [entorno virtual ya preparado](#) y nos aseguramos de descomprimirlo en la carpeta `/home/user/tallerimposm`. Una vez descargado lo activamos con:

```
$ cd /home/user/tallerimposm
$ source venv/bin/activate
```

---

**Importante:** Para que el taller funcione debe descomprimirse en la carpeta que se ha indicado, no funciona en ninguna otra ubicación sin hacer bastantes cambios en su configuración interna (y esto tampoco es aconsejable en cualquier caso).

---

### 3.5.3 Comprobar la versión de `imposm`

Para comprobar la versión de `imposm` ejecutamos:

```
(venv)$ imposm --version
```

Y deberíamos obtener:

```
Enabling Shapely speedups.
imposm 2.5.0
```

## 3.6 Obtener el juego de datos

Para este ejercicio vamos a usar una exportación de OpenStreetMap de la ciudad de Nottingham. Este juego de datos está ya en OSGeo Live 7 por lo que no tenemos que descargarlo.

El fichero que usaremos está es `/usr/local/share/data/osm/Nottingham.osm.bz2` y dispone de unos 4000 puntos de interés y unas 84000 vías.

## 3.7 Preparando la base de datos

El primer paso para la carga de datos es la creación de la base de datos. OSGeo Live 7.0 dispone de Posgres 9.1 con PostGIS 2.0. En esta combinación y con la configuración de OSGeo Live es muy sencillo crear una base de datos geográfica a la que nuestro usuario del sistema tendrá acceso. Para crear la base de datos `nott-osm` basta con ejecutar los siguientes comandos:

```
(venv)$ createdb -E UTF8 nott-osm
(venv)$ psql -d nott-osm -c "create extension postgis;"
```

Si por alguna razón queremos borrar la base de datos basta con ejecutar:

```
(venv)$ dropdb nott-osm
```

## 3.8 Primera importación

Podemos proceder a la primera importación de datos que realizaremos haciendo los tres pasos por separado:

- Lectura
- Escritura
- Optimización

### 3.8.1 Lectura

Se realiza empleando el comando:

```
$ imposm --read /usr/local/share/data/osm/Nottingham.osm.bz2
```

Como la cantidad de datos no es muy grande, solo tardará unos segundos. Una vez acaba podemos comprobar que ha creado los archivos de cache listando los archivos del directorio:

```
$ ls
```

```
imposm_coords.cache  imposm_nodes.cache  imposm_relations.cache  imposm_ways.cache  venv
```

Imposm ha generado los archivos `.cache` que son archivos binarios con los datos preparados para ser incluidos en la base de datos.

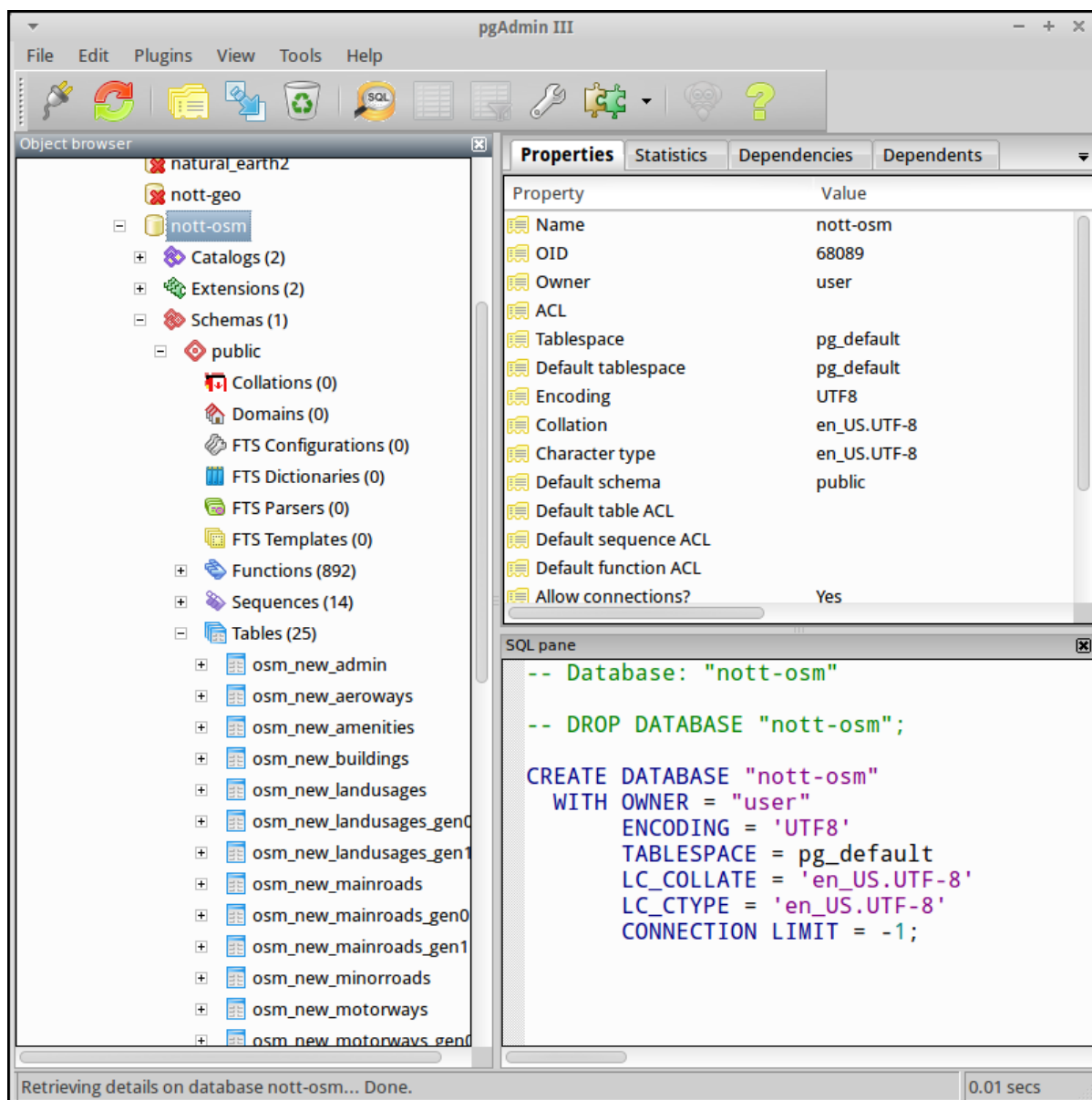
### 3.8.2 Escritura

Se realiza empleando el comando:

```
(venv) $ imposm --database nott-osm --host localhost --user user --write
```

Solicitará la contraseña del usuario `user` y cargará los datos que hay en los archivos `.cache`.

Podemos investigar qué ha hecho Imposm lanzando la aplicación pgAdmin III que está instalada en la máquina virtual en el menú `Development`. Podemos comprobar que ha creado 24 tablas nuevas, todas con el sufijo `new_`



El esquema de tablas y qué etiquetas ha importado son los estándar ya que aún no hemos cambiado los *mappings*. En concreto podremos encontrar:

- Amenities
- Places
- Transport\_points
- Administrative polygons
- Buildings
- Landusages
- Aeroways
- Waterareas
- Roads (en realidad repartidas en varias tablas en función de la categoría)
- Railways

- Waterways

También vienen unas tablas con geometrías de las vías de transporte generalizadas en función de dos tolerancias y unas vistas que agrupan todas las carreteras.

### 3.8.3 Optimización

El último paso de la carga de datos sería la optimización de los datos que se realiza empleando el comando:

```
(venv)$ imposm --database nott-osm --host localhost --user user --optimize
```

### 3.8.4 Todo en un paso

En realidad los tres pasos anteriores se podrían ejecutar en un solo comando:

```
$ imposm --database nott-osm --host localhost --user user \
  --read --write --optimize /usr/local/share/data/osm/Nottingham.osm.bz2
```

## 3.9 Flujo de trabajo

---

**Nota:** Mantén pgAdmin abierto y refresca con F5 para ver cómo van actualizándose las tablas.

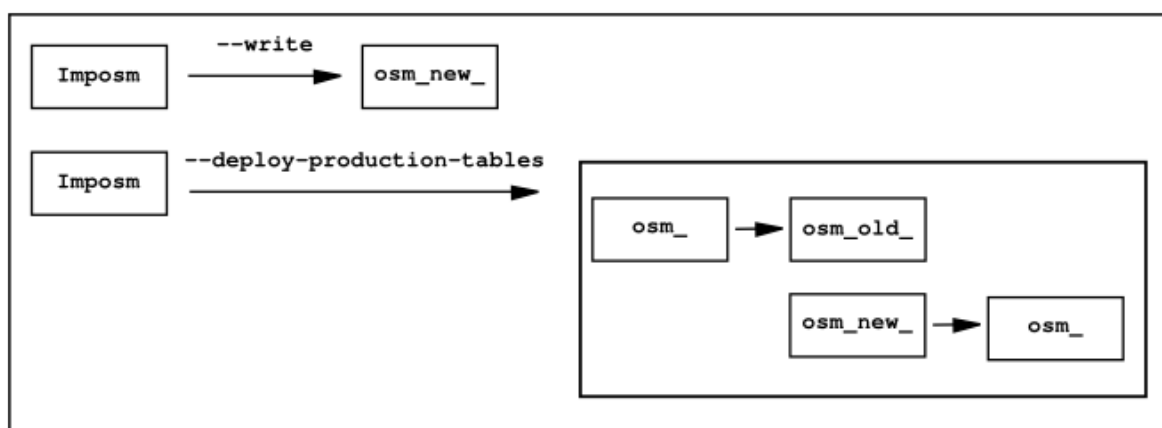
---

### 3.9.1 Pasar a producción

El flujo de trabajo recomendado permite el despliegue de las tablas conservando hasta 3 versiones a la vez del mismo juego de datos. El despliegue se inicia al ejecutar el comando:

```
$ imposm --database nott-osm --host localhost --user user --deploy-production-tables
```

La importación de datos se hace sobre tablas a las que se le añade el prefijo `osm_new_` en el nombre. Podremos comprobar con pgAdmin III como se ha cambiado el nombre de todas las tablas perdiendo el prefijo `new_`. Si ya hubiéramos hecho otro despliegue las actuales tablas `osm_` se renombrarán automáticamente a `osm_old_`. Cada vez que se hace un despliegue se borrarán primero las `osm_old_`.



Si volvemos a cargar la *cache* y a pasar a producción las tablas con:

```
$ imposm --database nott-osm --host localhost --user user \
  --write --optimize --deploy-production-tables
```

veremos como las tablas que no tengan prefijo pasarán a tener el prefijo **old\_**.

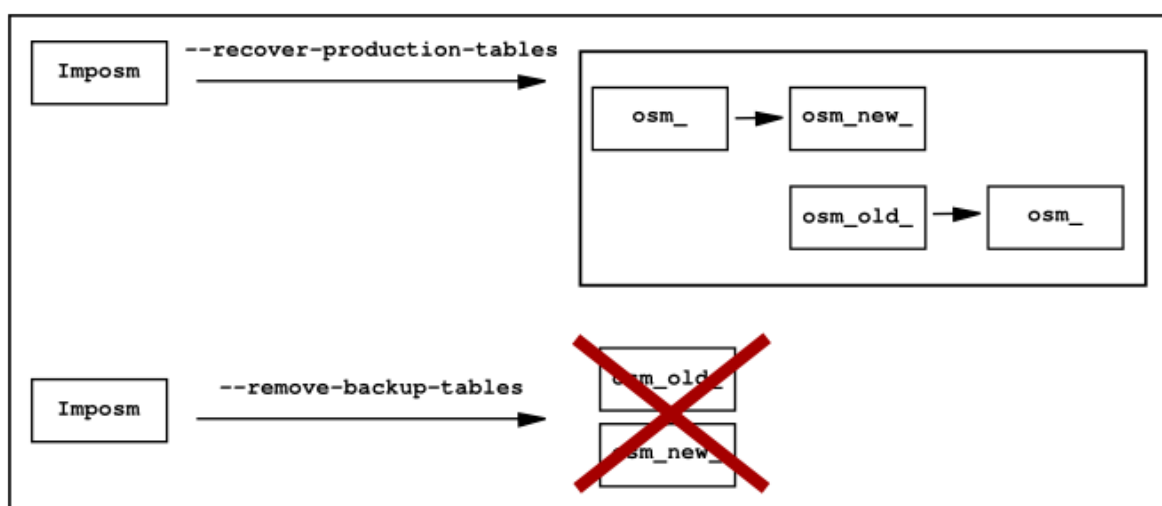
### 3.9.2 Revertir el despliegue y borrar tablas temporales

Para revertir el despliegue se puede ejecutar el comando:

```
$ imposm -d osm --recover-production-tables
```

Finalmente para borrar *definitivamente* las tablas marcadas con **old\_** y las marcadas con **new\_** se emplea el comando:

```
$ imposm --database nott-osm --host localhost --user user --remove-backup-tables
```



## 3.10 Modificando el *mapping*

El esquema de base de datos por defecto que utiliza Imposm viene de los [elementos y etiquetas más comunes de OSM](#). Este esquema permite trasladar los datos empleando el paquete `imposm.mapping` y las estructuras definidas en el archivo:

```
/usr/local/lib/python2.7/dist-packages/imposm/defaultmapping.py
```

### 3.10.1 Tablas

Hay definidas tres clases de Python para las geometrías base: `Points`, `LineStrings` y `Polygons` y todas las tablas tienen que ser instancias de una de ellas. Las tres clases usan los mismos argumentos:

**name** Nombre de la tabla (sin prefijos).

**mapping** El *mapping* de los pares clave/valor básicos que se meterán en la tabla.

**fields** El *mapping* de campos adicionales que también son pares clave/valor de OSM y que se convertirán en columnas de la tabla.



**field\_filter** Filtros que permitan discriminar los datos que se introducen.

### 3.10.2 *mapping*

El argumento *Mapping* debe ser un diccionario (un diccionario de Python) en la que las claves de OSM (p.e. *highway*, *leisure*, *amenity*, etc.) son las claves del diccionario y los valores de OSM (p.e. *motorway*, *trunk*, *primary*, etc.) los valores de las claves del diccionario.

Para una tabla de paradas de autobús, de tranvía y de ferrocarril el *mapping* debería ser parecido a este:

```
mapping = {
    'highway': (
        'bus_stop',
    ),
    'railway': (
        'station',
        'halt',
        'tram_stop',
    )
}
```

### 3.10.3 *fields*

El argumento *fields* debe ser una lista (o una tupla) con el nombre de la columna y su tipo de dato. Se emplea para añadir información adicional a la tabla. Imposm tiene clases para los tipos de datos más comunes que son las responsables de hacer sustituciones como *1*, *yes* y *true* a `TRUE` en caso de datos booleanos por lo que se recomienda su uso:

```
fields = (
    ('tunnel', Bool()),
    ('bridge', Bool()),
    ('oneway', Direction()),
    ('ref', String()),
    ('z_order', WayZOrder()),
)
```

En el ejemplo la línea `('tunnel', Bool())` convertirá los valores de la clave `tunnel` a valores booleanos.

### 3.10.4 Ejemplo

```
towers = Points(
    name = 'towers',
    mapping = {
        'man_made': (
            'tower',
            'water_tower',
        )
    }
    fields = (
        ('height', Integer()),
    )
)
```

### 3.10.5 Ampliando el esquema por defecto

El esquema que carga Imposm por defecto es generalmente insuficiente ya que se suele emplear un abanico de datos mucho más amplio.

Por ejemplo, en nuestro caso no se está incluyendo en la base de datos ningún registro de los siguientes tipos y subtipos:

- Amenity
  - restaurant
  - pub
  - cafe
  - place of worship
  - parking
- Natural
- Tourism
- Barrier

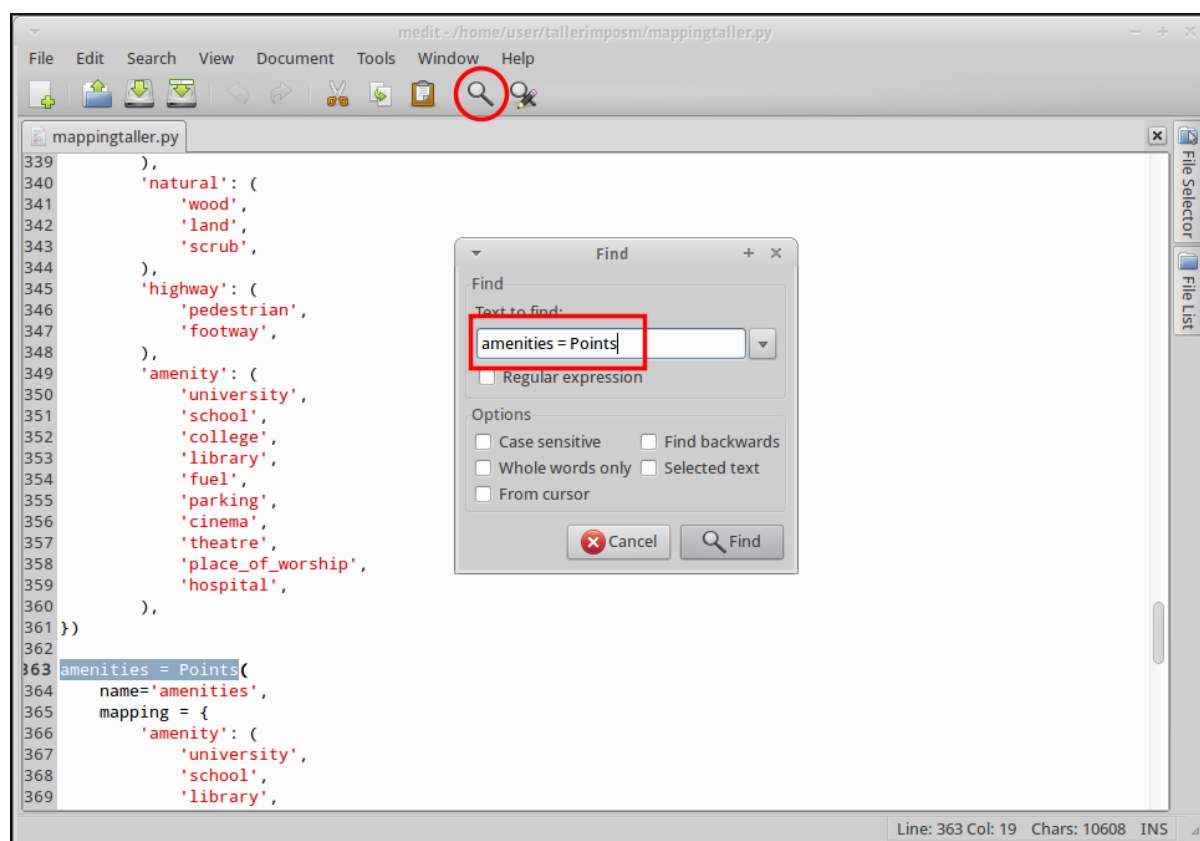
Por lo que debemos modificar el archivo de *mapping* para que los incluya. El archivo *mapping* se encuentra en la siguiente localización:

```
/usr/local/lib/python2.7/dist-packages/imposm/defaultmapping.py
```

lo copiamos y editamos empleando los siguientes comandos:

```
(venv) $ cd ~/tallerimposm
(venv) $ cp venv/lib/python2.7/site-packages/imposm/defaultmapping.py mappingtaller.py
(venv) $ medit mappingtaller.py
```

Buscamos la cadena `amenities = Points` usando el comando buscar de *medit* pulsando en la lupa de la barra de herramientas.



Como podemos ver, Imposm por defecto tiene determinados tipos de Amenity cuando son puntos pero no tiene ninguno de los indicados en la lista referida un par de párrafos más arriba.

Vamos a añadir al argumento *mapping* los elementos que le faltan (no importa el orden) respetando la sintaxis de tuplas de Python de forma que quede de la siguiente manera:

```
amenities = Points(
    name='amenities',
    mapping = {
        'amenity': (
            'university',
            'school',
            'library',
            'fuel',
            'hospital',
            'fire_station',
            'police',
            'townhall',
            'restaurant',
            'pub',
            'cafe',
            'place_of_worship',
            'parking',
        ),
    },
)
```

El caso de los árboles (*natural/tree*) es distinto ya que por defecto Imposm no incluye un *mapping* para la clave *Natural*, por lo que la crearemos desde cero, justo debajo del objeto *amenities* vamos a crear un nuevo objeto para poder importarlos.

```
arboles = Points(  
    name = 'arboles',  
    mapping = {  
        'natural': (  
            'tree',  
        ),  
    },  
)
```

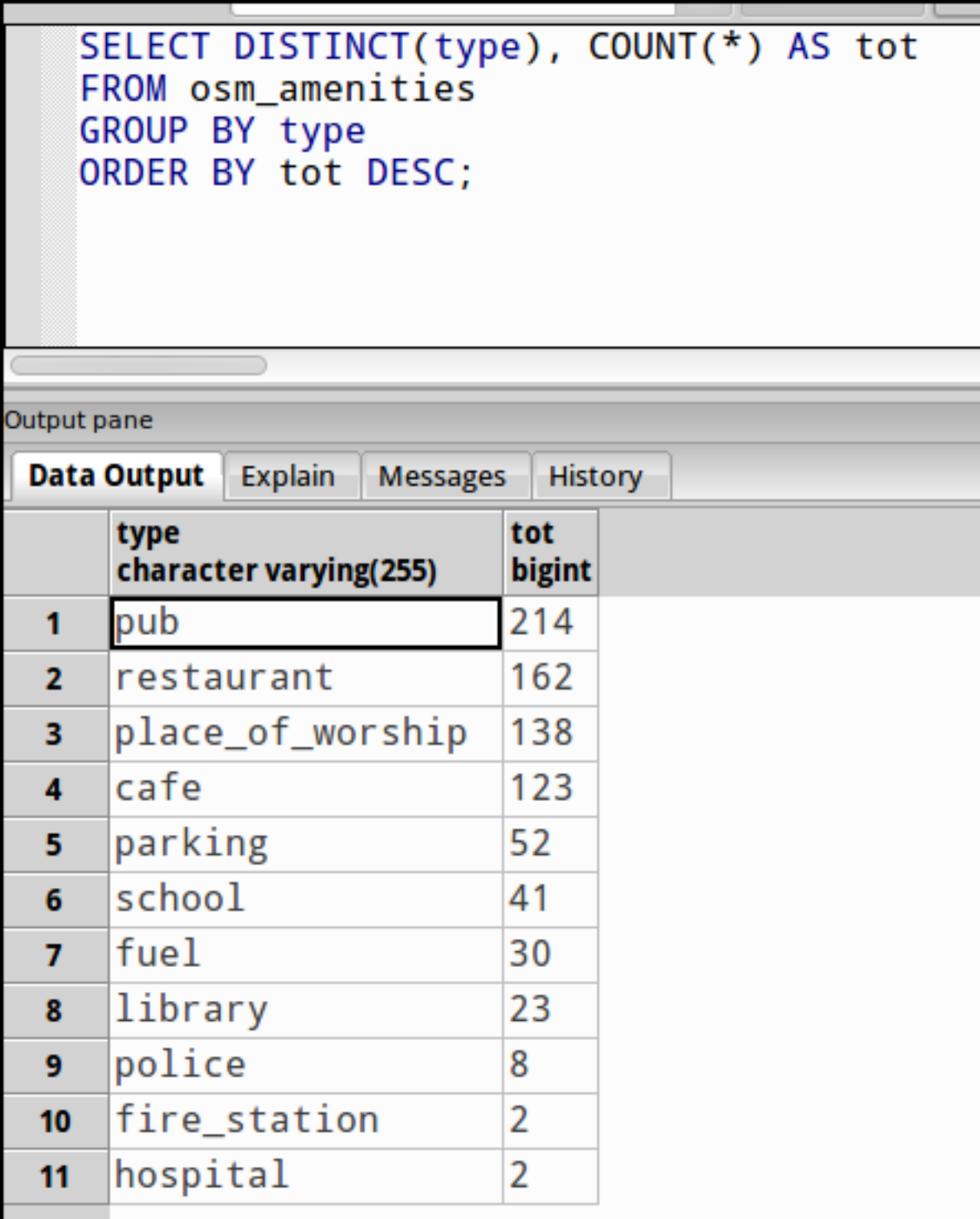
Guardamos el archivo con y salimos de *medit*.

Ejecutamos el comando para escribir y optimizar los datos en la base de datos:

```
(venv)$ imposm --database nott-osm --host localhost --user user \  
--read /usr/local/share/data/osm/Nottingham.osm.bz2 \  
--write --optimize --deploy-production-tables \  
--overwrite-cache --remove-backup-tables -m mappingtaller.py
```

En este caso es necesario volver a leer los datos y generar los archivos de cache, ya que hemos modificado la estructura de los datos. Con la opción `--overwrite-cache` se sobrescribirán directamente los archivos necesarios.

Con pgAdmin podemos comprobar como se han importado 1059 árboles y si echamos un vistazo a la tabla `osm_amenities` veremos que se han importado puntos con las etiquetas que hemos elegido.



The screenshot shows a SQL query in a text editor and its results in an 'Output pane'. The query is:

```
SELECT DISTINCT(type), COUNT(*) AS tot
FROM osm_amenities
GROUP BY type
ORDER BY tot DESC;
```

The 'Output pane' has tabs for 'Data Output', 'Explain', 'Messages', and 'History'. The 'Data Output' tab is selected, showing a table with two columns: 'type' (character varying(255)) and 'tot' (bigint). The results are ordered by 'tot' in descending order.

	type character varying(255)	tot bigint
1	pub	214
2	restaurant	162
3	place_of_worship	138
4	cafe	123
5	parking	52
6	school	41
7	fuel	30
8	library	23
9	police	8
10	fire_station	2
11	hospital	2

### 3.10.6 Ejercicio

Como ejercicio del taller se propone crear el *mapping*, escribir los datos en la base de datos y desplegar las tablas para las claves de OSM siguientes:

**tourism (71 puntos nuevos):** information, hotel, artwork y attraction

**barrier (1688 puntos nuevos):** gate, bollard, entrance, cycle\_barrier, lift\_gate, stile y fence

## 3.11 Referencias y enlaces

- [Web de Imposm](#)
- [Web de Omniscale](#)
- [Cómo instalar un template de PostGIS](#)





---

## TileMill, el estudio cartográfico

---

**TileMill** es una herramienta que permite un acercamiento al diseño cartográfico a través de un lenguaje que es familiar a los desarrolladores web. Se trata de un producto de escritorio (aunque se puede ejecutar para acceder vía *web*). El objetivo de **TileMill** es diseñar cartografía de la forma más sencilla y atractiva posible generando como productos finales diferentes visualizaciones, tal y como se verá más adelante.

**TileMill** es *software* libre, está desarrollado por [Mapbox](#) y el código fuente está disponible en su [repositorio en GitHub](#).

### 4.1 Iniciando TileMill

---

**Nota:** Este taller está diseñado para ejecutarse en OSGeo Live 7.0

---

Arrancamos **TileMill** seleccionando la opción del menú *Geospatial* → *Spatial Tools* → *TileMill*

### 4.2 Secciones

La interfaz de **TileMill** dispone de las siguientes secciones:

**Editor:** Es el espacio de trabajo del estudio, donde se cargan datos y se le da estilo a la cartografía.

**Projects:** Espacio para administrar los proyectos que tenemos cargados en **TileMill**. Solo podemos tener cargado un proyecto cada vez en el editor.

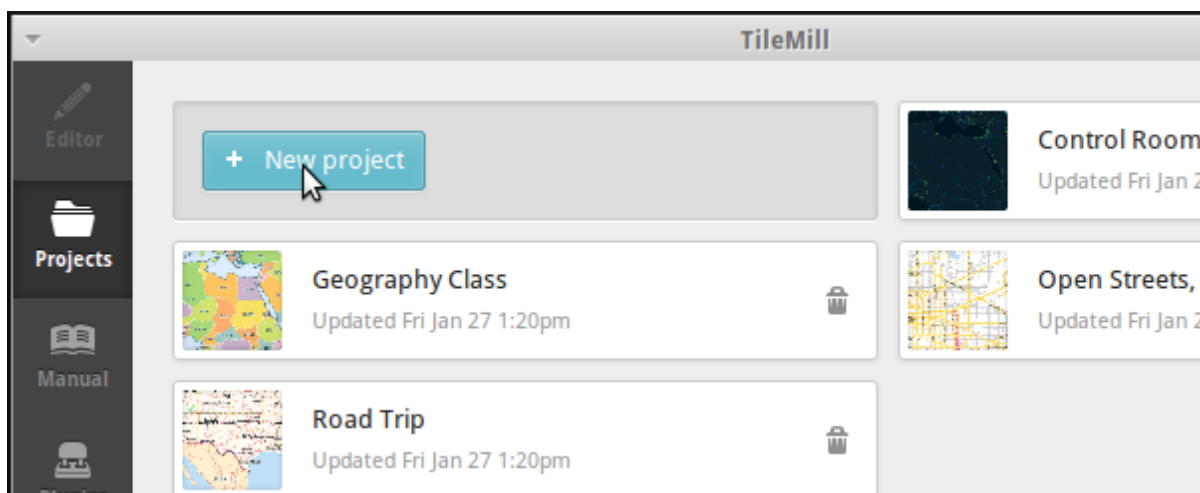
**Manual:** Documentación integrada con diversos apartados sobre cómo funciona **TileMill**.

**Plugins:** Sección para activar funcionalidad adicional de **TileMill**.

**Settings:** Configuración de **TileMill**.

### 4.3 Creando el proyecto

**TileMill** carga por defecto la pestaña de *Projects* y en ella tenemos el botón + *New Project* que pulsaremos para definir nuestro proyecto.



Nos muestra la ventana de información del proyecto en la que deberemos introducir los datos básicos que lo identifiquen.

**Filename** cfp2014

**Name** Curso TileMill CFP 2014

**Description** Mapa de Nottingham

**File format** PNG 24

**Default data** Dejar marcado

Y pulsamos el botón *Add*

Al abrir el proyecto, pulsando sobre el en la pestaña *Projects* vemos que se han cargado una capa de países por defecto y que tiene un nivel de visualización bastante alto.

## 4.4 Añadiendo datos

El primer paso siempre es añadir datos y el primer paso para añadirlos es tener claros sus metadatos, **siempre** hay que poner especial atención a:

- Formato
- Tamaño
- Sistema de referencia

La confusión en cualquiera de los tres campos puede llevarnos a que la cartografía no se pueda cargar o no quede alineada correctamente.

**TileMill** no puede reproyectar los datos que usa como origen de información de los mapas que componen, por lo que siempre se le deben proporcionar en uno de los SRS soportados que son EPSG:900913 (EPSG:3857) y WGS84 (EPSG:4326) aunque existe la posibilidad de forzar la reproyección introduciendo los valores adecuados para [proj4](#) que suelen poder conseguirse en <http://epsg.io>.

#### 4.4.1 Formatos vectoriales admitidos

**CSV** Se trata de archivos de hoja de cálculo con variables separadas **por comas** y que tienen la información geográfica en columnas que se llaman «lat» o «latitude» o incluso «geo\_longitude», **TileMill** reconoce automáticamente el nombre de esas columnas.

**ESRI Shapefile** Uno de los formatos vectoriales más populares antiguamente. Si el archivo `.prj` no está presente **TileMill** intentará averiguar el SRS de la información contenida.

**KML** Este formato soportado, tiene algunas limitaciones para ser usado en **TileMill** ya que no reconoce algunas de las funcionalidades avanzadas de los KMLs (estilos embebidos, imágenes, modelos 3D). Tampoco reconoce el formato KMZ.

**GeoJSON** Es uno de los formatos más populares actualmente, es un formato basado solamente en texto con una estructura flexible.

#### 4.4.2 Formatos *raster* admitidos

**GeoTIFF** Es uno de los formatos más conocidos para almacenar imágenes aéreas, satélite y modelos de elevación del terreno. Para manipular la información raster **TileMill** emplea **GDAL** que es una potentísima biblioteca de acceso a datos raster.

#### 4.4.3 Bases de datos admitidas

**SQLite** Es el sistema de bases de datos basadas en un solo archivo más popular del Software Libre. Estas bases de datos se pueden generar empleando un software de escritorio como **QGis**.

**PostGIS** Literalmente el elefante en la habitación. El mayor proyecto de base de datos relacional geográfica del Software Libre.

### 4.5 Introducción al lenguaje CartoCSS

**CartoCSS** es el lenguaje que utiliza **TileMill** para aplicar estilos a las primitivas cartográficas. Está basado en *Cascadenik* que es un pre-procesador de estilos más antiguo. **CartoCSS** utiliza la biblioteca de renderizado de cartografía **Mapnik**, otro excelente componente de *software libre*.

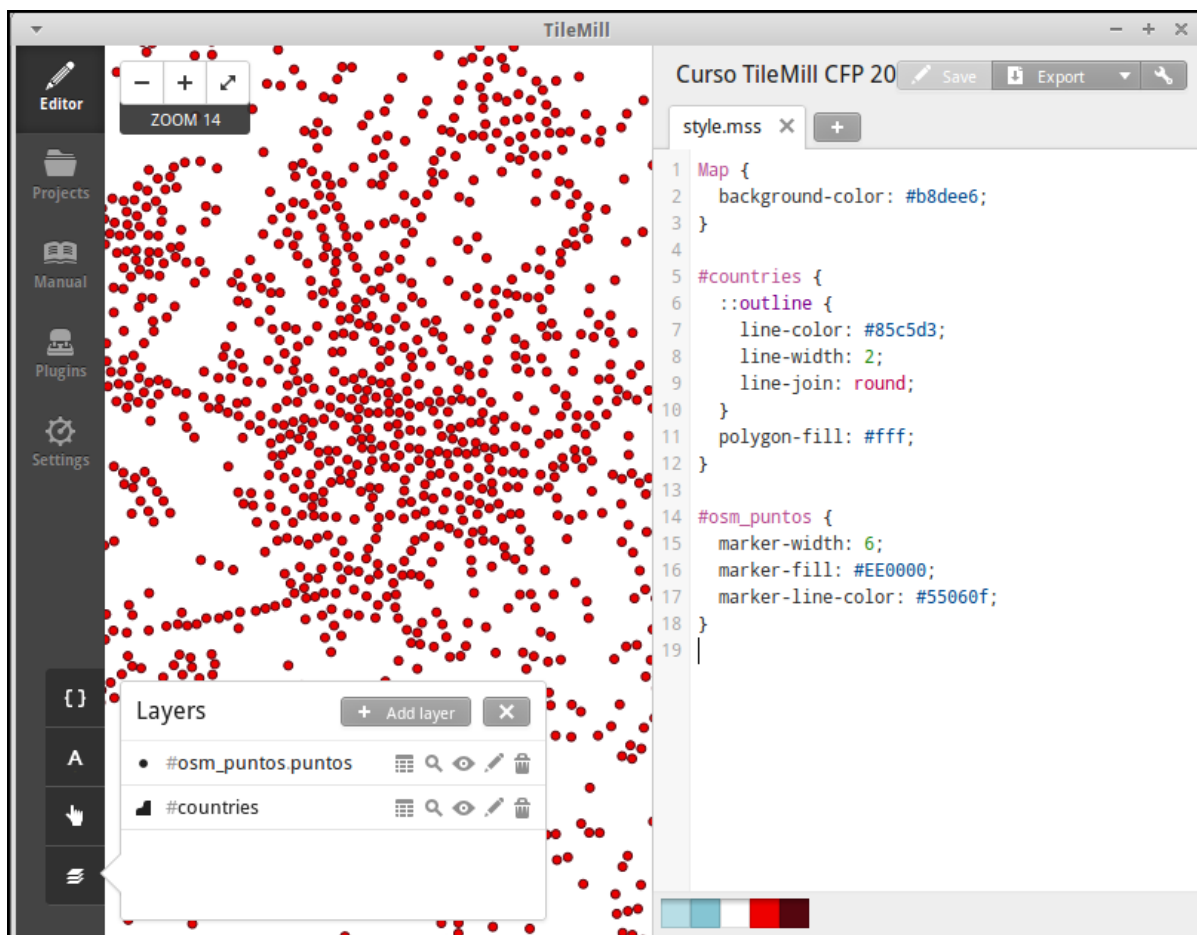
**Mapnik** se configura directamente mediante ficheros XML, pero poca gente entiende XML así que hace un tiempo que aparecieron estas herramientas que generan su XML a partir de un lenguaje más sencillo y expresivo, en definitiva para hacer «la vida más fácil» a los usuarios de **Mapnik**.

**TileMill** usa **Mapnik** por debajo y **CartoCSS** es el lenguaje con el que le comunica cómo deben quedar las cosas.

### 4.5.1 Pintando puntos

```
#osm_puntos {
  marker-width: 6;
  marker-fill: #EE0000;
  marker-line-color: #55060f;
}
```

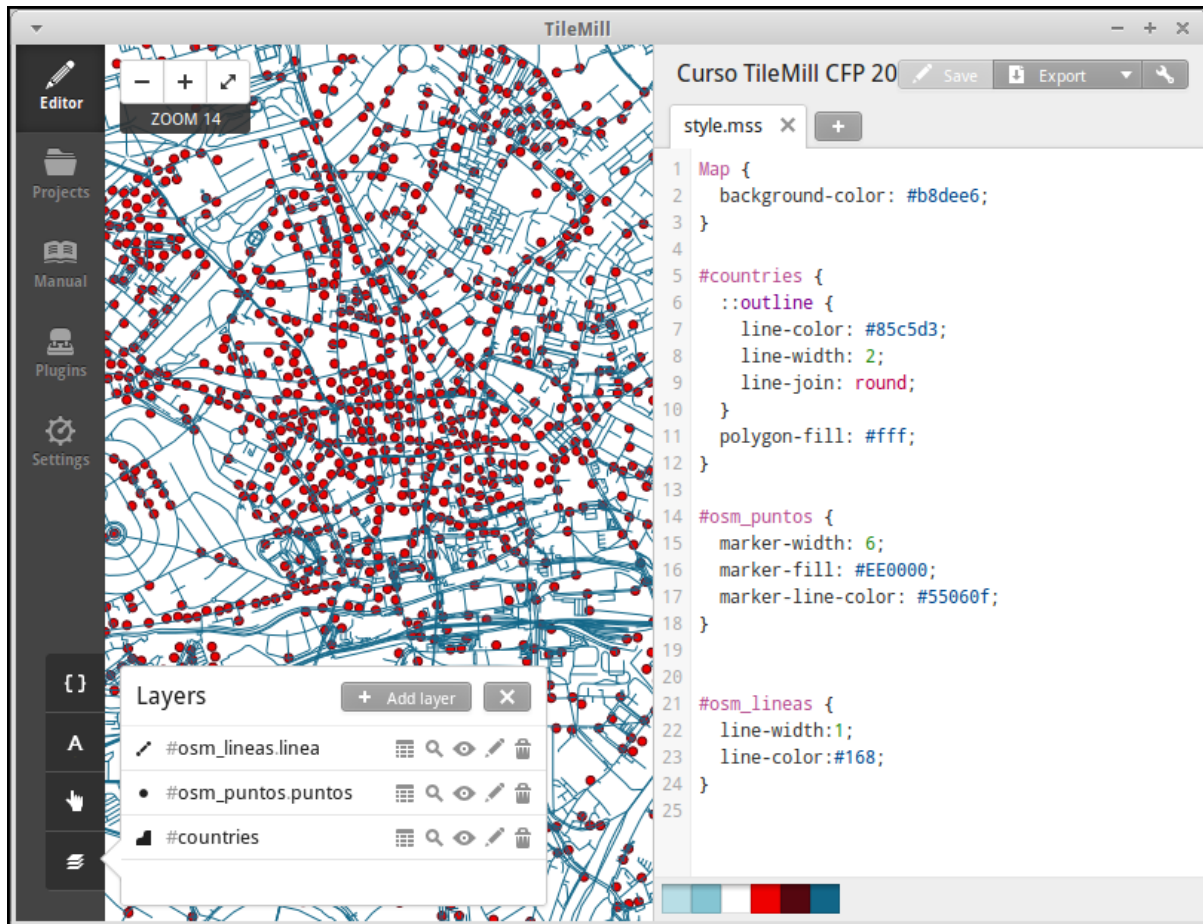
Existen dos tipos de *puntos* **Point** y **Marker** entre los dos suman 30 propiedades.



### 4.5.2 Pintando líneas

```
#osm_lineas {
  line-width: 1;
  line-color: #168;
}
```

Existen 19 propiedades distintas para las líneas.

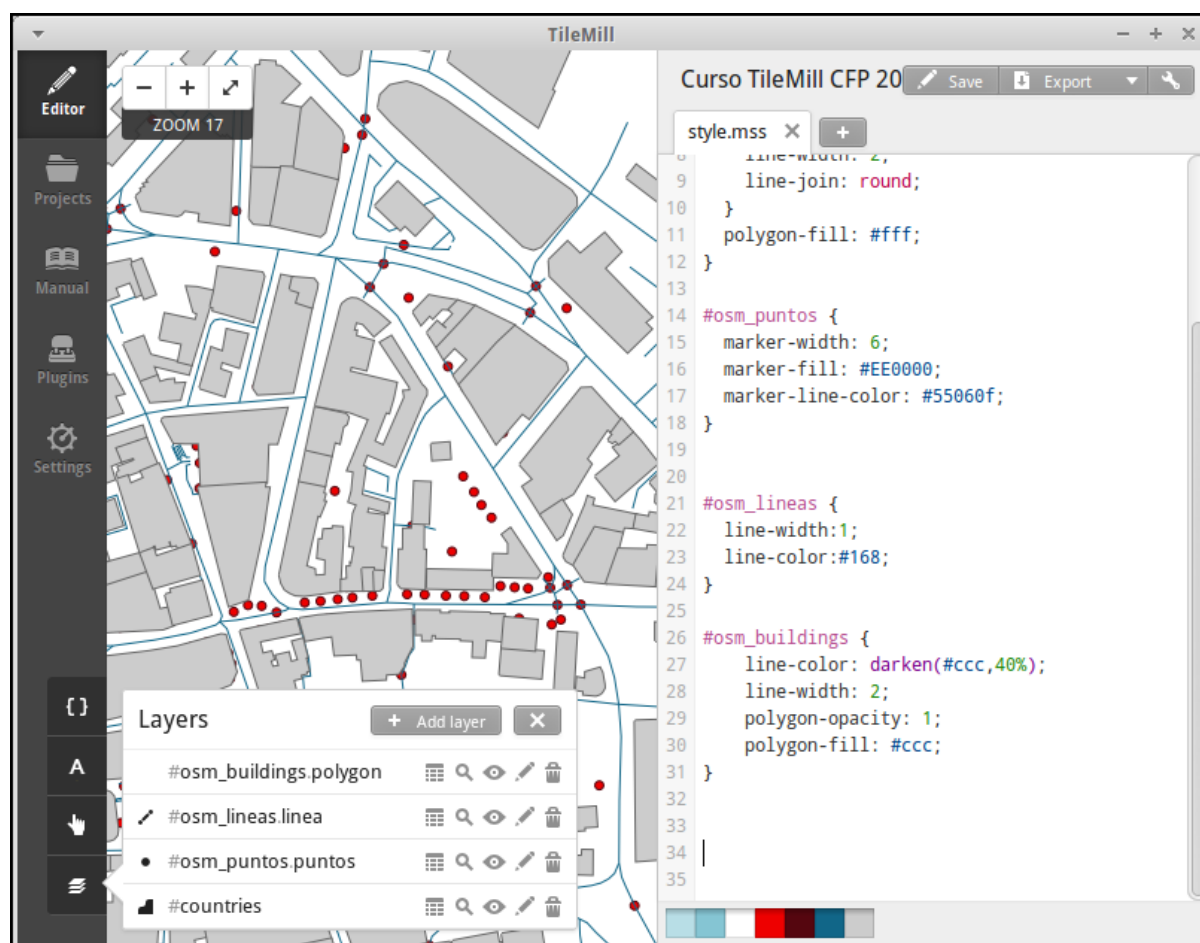


### 4.5.3 Pintando áreas

```
#osm_buildings {
  line-color: darken(#ccc, 40%);
  line-width: 2;
  polygon-opacity: 1;
  polygon-fill: #ccc;
}
```

Existen 5 propiedades distintas para las áreas.





#### 4.5.4 Pintando con clase

También se pueden usar clases y condiciones para filtrar las propiedades por atributos o por el nivel de **zoom** en el que nos encontremos. Finalmente los selectores se pueden anidar para compartir propiedades. Más información en la [documentación sobre selectores](#).

En el ejemplo siguiente se seleccionan todos los puntos de la capa `osm_puntos` que tengan algún dato en el campo `tourism` (pidiendo que sea distinto a la cadena de texto vacía) y se aplicará solo a partir del nivel de `zoom` 14. En ese selector se establecen unas propiedades generales de tamaño y color del borde y a continuación se anidan selectores por cada una de las clases a renderizar estableciendo solo la propiedad que va a cambiar, es decir, el color del símbolo.

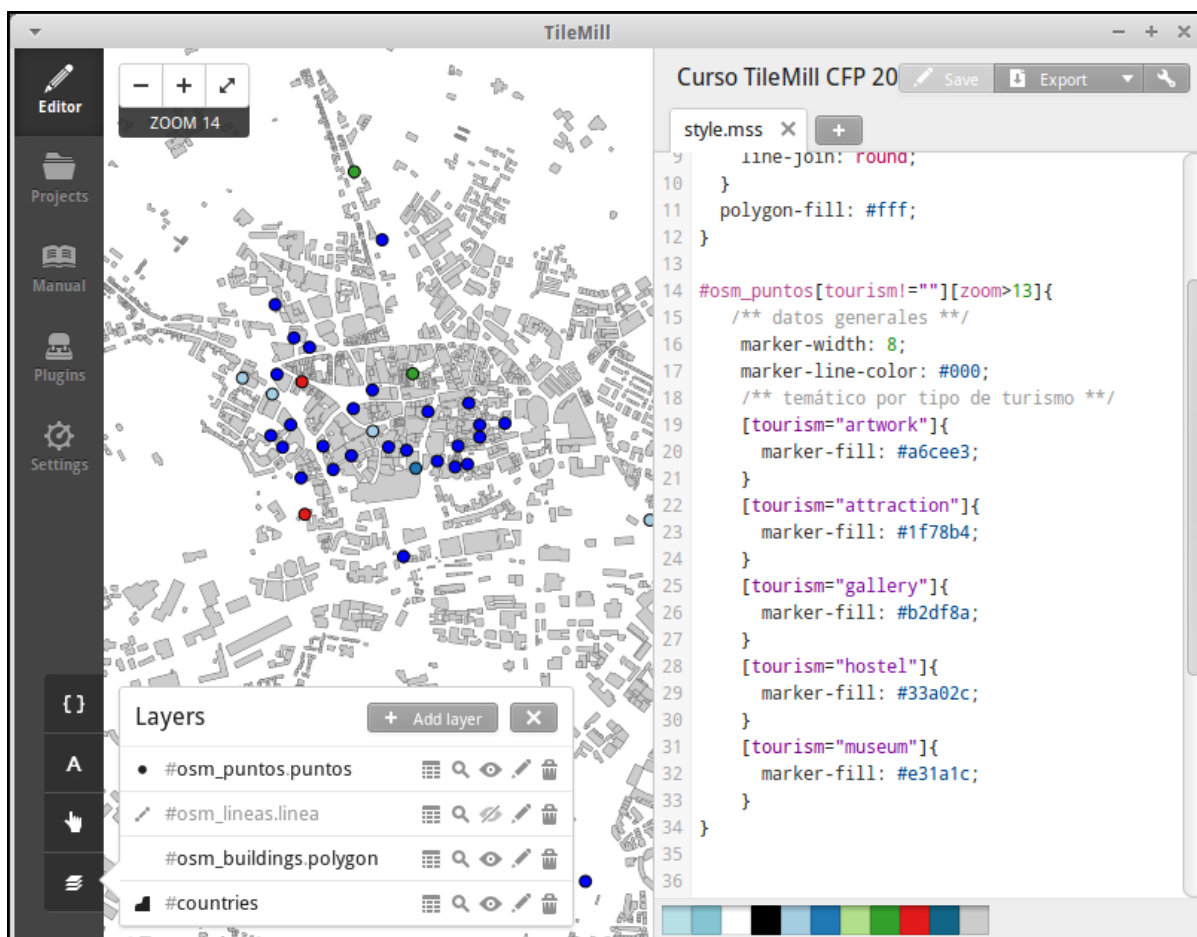
```
#osm_puntos[tourism!=""][zoom>13]{
  /** propiedades generales **/
  marker-width: 8;
  marker-line-color: #000;

  /** temático por tipo de turismo **/
  [tourism="artwork"]{
    marker-fill: #a6cee3;
  }
  [tourism="attraction"]{
    marker-fill: #1f78b4;
  }
  [tourism="gallery"]{
    marker-fill: #b2df8a;
  }
}
```

```

}
[tourism="hostel"]{
  marker-fill: #33a02c;
}
[tourism="museum"]{
  marker-fill: #e31a1c;
}
}

```



#### 4.5.5 Y alguna cosilla más

El uso de @ te permite definir **variables**

```

/** colores para agua y bosque*/
@water : #c0d8ff;
@forest: #cea;

/** estilos para usos del suelo
de para agua y bosque */
#landusage{
  /* características generales */
  line-color: #000;
  line-width: 2;
  polygon-opacity: 1;

  [type="water"]{

```

```
    polygon-fill: @water;
  }
  [type="wood"]{
    polygon-fill: @forest;
  }
}
```

Y existen funciones para operar sobre los colores para aclararlos, oscurecerlos, etc. ([referencia de color](#)) :

```
@border-water: darken(@water, 50%);
```

En algo como esto:

```
@water : #c0d8ff;
@forest : #cea;

/** los bordes más oscuros **/
@border-water : darken(@water, 50%);
@border-forest : darken(@forest, 50%);


#landusage{
  /* características generales */
  line-width : 2;
  polygon-opacity : 1;

  [type="water"]{
    polygon-fill : @water;
    line-color : @border-water;
  }
  [type="wood"]{
    polygon-fill : @forest;
    line-color : @border-forest;
  }
}
```

## 4.6 Taller

En las siguientes secciones se espera que el alumno repita las acciones propuestas para cargar los distintos tipos de datos soportados por TileMill y aplicando estilos similares a los indicados.

### 4.6.1 Añadiendo una capa de puntos

Procederemos ahora a añadir nuestra primera capa de puntos, para lo que desplegaremos el menú de capas pulsando en el botón  y seleccionamos + *Add layer*

En la ventana que aparece seleccionaremos la opción de *PostGIS* y rellenamos los campos como se indica.

**Add layer**

File SQLite **PostGIS**

ID  select in Carto #id

Class  select in Carto .class

\* **Connection**  host=? port=? user=? password=? dbname=?   
Provide your PostGIS authentication and connection parameters.

\* **Table or subquery**

**Unique key field**  SQL field containing a unique key for each feature

**Geometry field**  SQL field containing feature geometry

**Extent**    
Auto-calculate and cache the extent to limit the query by.

**SRS**   Autodetect  
SRS projection string for this datasource. TileMill can often autodetect this value.

**Advanced**  option1="?" option2="?" Optional, advanced arguments to pass to Mapnik.

**ID** osm\_puntos

**Class** puntos

**Connection** dbname=nott-osm host=localhost port=5432 user=user password=user

**Table or subquery** osm\_places

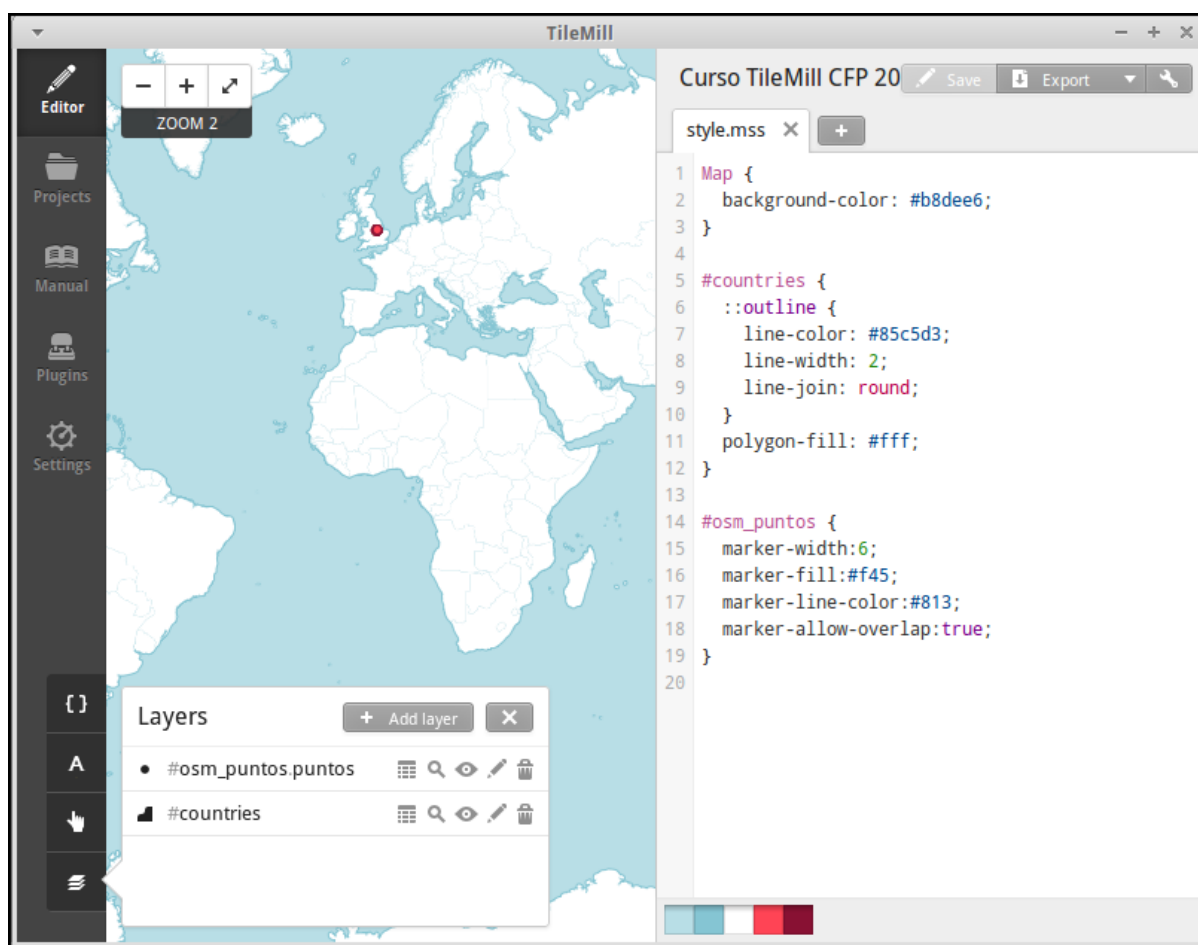
**Unique key field** osm\_id

**Geometry field** geometry

**SRS** Seleccionamos 900913

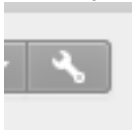
Y pulsamos *Save & Style* para que añada los datos con un estilo por defecto.

Veremos como inmediatamente aparece un punto en la zona de Inglaterra.



## Corrigiendo la visualización por defecto

En realidad nuestra zona de trabajo es bastante más pequeña que la que muestra por defecto TileMill, por lo que modificaremos las preferencias para que muestre por defecto una zona más ajustada a nuestro

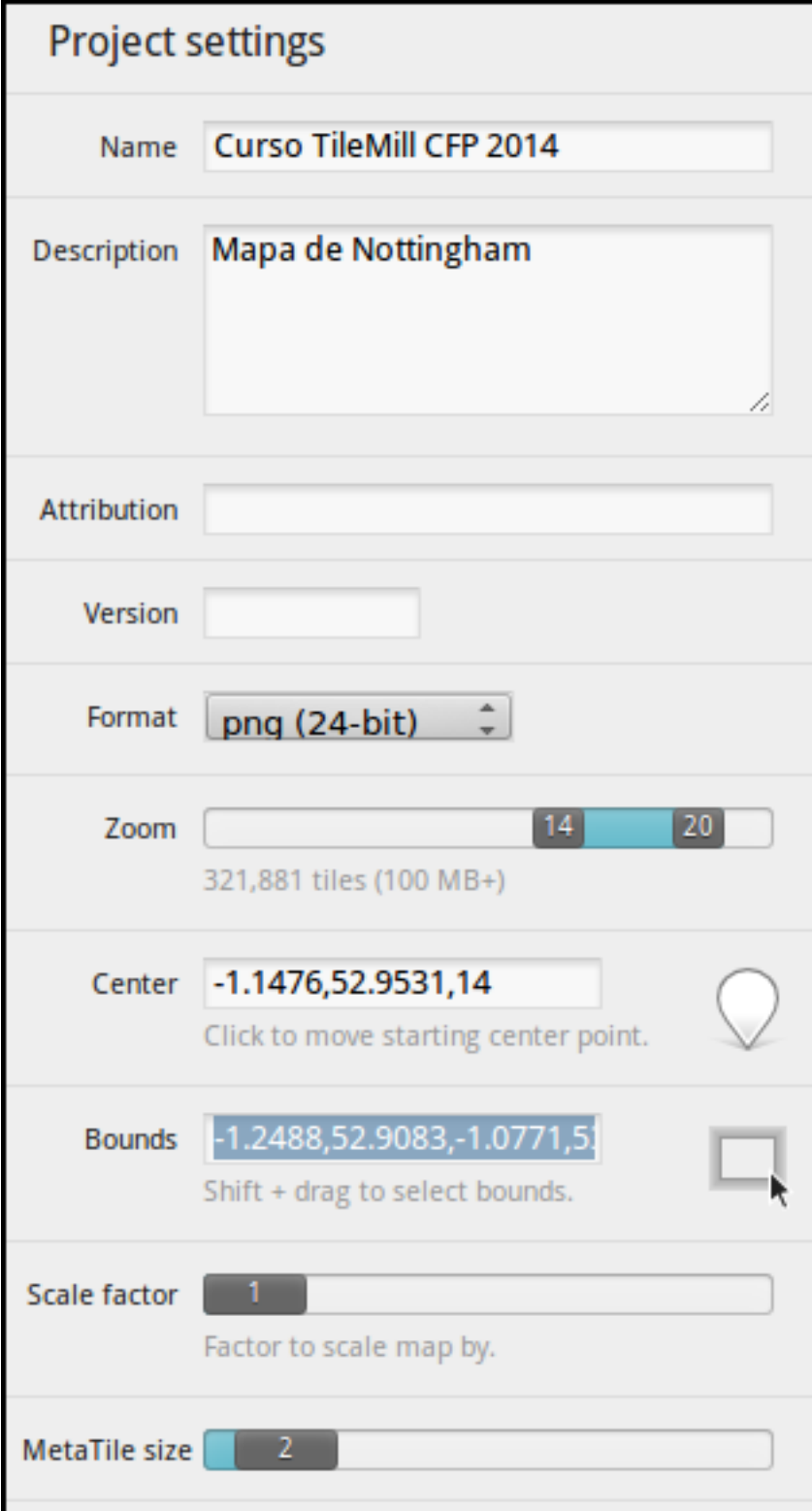
juego de datos. Para ello pulsaremos en el botón de configuración del proyecto  y lo configu-

ramos de la siguiente forma:

**Zoom** Desplazar las barras para que los niveles de *zoom* estén entre 12 y 20

**Center** -1.1476,52.9531,12

**Bounds** -1.2488, 52.9083, -1.0771, 53.0076



The image shows a 'Project settings' dialog box with the following fields and controls:

- Name:** Curso TileMill CFP 2014
- Description:** Mapa de Nottingham
- Attribution:** (empty text field)
- Version:** (empty text field)
- Format:** png (24-bit)
- Zoom:** A slider ranging from 14 to 20, with a value of 14 selected. Below the slider, it says '321,881 tiles (100 MB+)'. The slider bar is blue with a white handle.
- Center:** -1.1476,52.9531,14. Below the text is a map icon and the instruction 'Click to move starting center point.'
- Bounds:** -1.2488,52.9083,-1.0771,51. Below the text is a selection box icon and the instruction 'Shift + drag to select bounds.'
- Scale factor:** A slider with a value of 1 selected. Below the slider is the instruction 'Factor to scale map by.'
- MetaTile size:** A slider with a value of 2 selected.

#### 4.6.2 Añadiendo elementos lineales

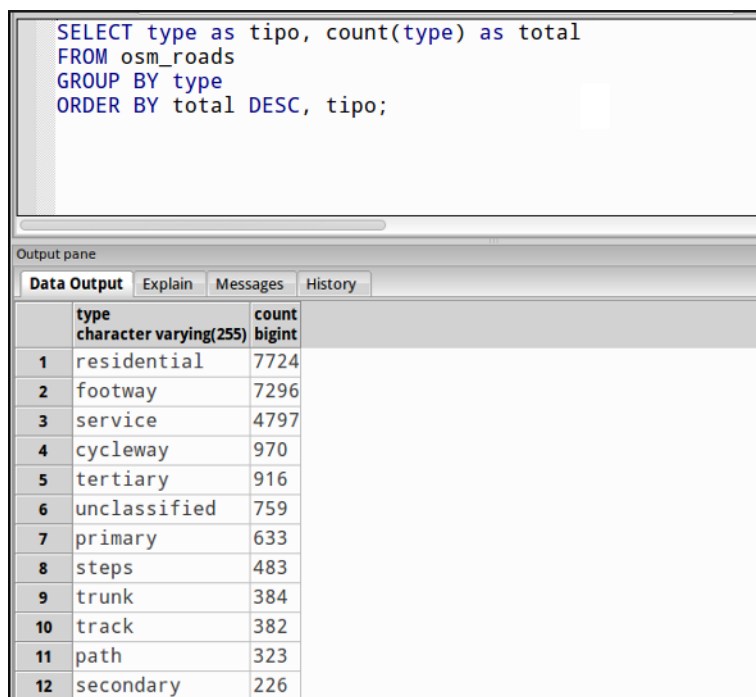
Para representar las calles utilizaremos una de las *ayudas* que proporciona ImpOSM; como ya hemos dicho, por defecto separa las vías en varias tablas, pero también crea una vista de PostGIS que aglutina toda la información relativa a estas.

Añadiremos una nueva capa de PostGIS que lea la información de la tabla `osm_roads`



Para obtener todos los distintos tipos de vía podemos usar emplearemos *pgAdmin III* donde podemos lanzar la consulta:

```
SELECT type as tipo, count(type) as total
FROM osm_roads
GROUP BY type
ORDER BY total DESC, tipo;
```



The screenshot shows the pgAdmin III interface. The top pane contains the SQL query: `SELECT type as tipo, count(type) as total FROM osm_roads GROUP BY type ORDER BY total DESC, tipo;`. The bottom pane, titled 'Output pane', has tabs for 'Data Output', 'Explain', 'Messages', and 'History'. The 'Data Output' tab is selected, displaying a table with 12 rows of road types and their counts, ordered by count in descending order.

	type character varying(255)	count bigint
1	residential	7724
2	footway	7296
3	service	4797
4	cycleway	970
5	tertiary	916
6	unclassified	759
7	primary	633
8	steps	483
9	trunk	384
10	track	382
11	path	323
12	secondary	226

Añadiremos una entrada para cada tipo de vía.

- residential
- footway
- service
- cycleway
- tertiary
- unclassified
- primary
- steps

Para representarlo usaremos el código como el siguiente:

```
#osm_lineas {
    line-width:1;
    line-color:#cce;
    [type = 'footway'], [type = 'cycleway'] {
        line-color:#f2f974;
    }
    [type = 'residential'],      [type = 'service']  {
        line-color:#aaa;
    }
    [type = 'steps'] {
```

```

        line-color:#7cc7fd;
    }
    [type = 'unclassified'] {
        line-color:#ff9f3b;
    }
    [type = 'primary'] {
        line-width:2;
        line-color:darken(#ff9f3b, 30%);
    }
    [type = 'tertiary'] {
        line-width:1.5;
        line-color:darken(#8beb18, 10%);
    }
}

```


### 4.6.3 Añadiendo los edificios

Añadiremos ahora los edificios, que están en la tabla *osm\_buildings*.

```

#osm_edificios {
    line-color:#a71b62;
    line-width:0.5;
    polygon-opacity:1;
    polygon-fill:#d86ebb;
}

```

**Importante:** El **orden de renderizado** de las capas es el orden en el que aparecen en el gestor de capas , para cambiar el orden basta pulsar en el indicador del tipo de capa (puntos, líneas y áreas) que hay junto al nombre y arrastrar hacia arriba o hacia abajo la capa.

### 4.6.4 Añadiendo etiquetas

Por último, añadiremos los nombres de las calles, para lo cual primero tenemos que definir una variable, preferentemente al principio de todas las definiciones, que tenga el nombre de la fuente y las posibles fuentes sustitutas si la fuente no está instalada en el sistema.

```
@futura_med: "Futura Medium", "Function Pro Medium", "Ubuntu Regular", "Trebuchet MS Regular"
```

**TileMill** incorpora un gestor de fuentes que nos permite ver qué fuentes hay instaladas en el sistema al

que se accede empleando el botón de fuentes , las fuentes instaladas aparecen en **negrita** y el

gestor nos permite copiar y pegar literalmente el nombre de la fuente.

Aunque la capa de calles ya tiene el campo *name* que es el que vamos a utilizar, es siempre muy recomendable volver a añadir la capa y usarla exclusivamente para las etiquetas. En este caso rellenaremos los campos con los siguientes datos:

**ID** calles\_nombres

**Class** nombres

**Connection** dbname=nott-osm host=localhost port=5432 user=user password=user

**Table or subquery** (SELECT \* FROM osm\_roads WHERE name IS NOT NULL) AS foo

**Unique key field** osm\_id

**Geometry field** geometry

**SRS** Seleccionamos 900913

En esta ocasión en vez de la tabla, hemos usado una subconsulta, de forma que solo carguemos en memoria las entidades que tengan algún valor en el campo *name*. A las subconsultas hay que añadirles un alias para que **TileMill** las reconozca.

**TileMill** habrá asignado a la capa un estilo por defecto para capas de líneas, aunque nosotros lo vamos a modificar para que represente textos:

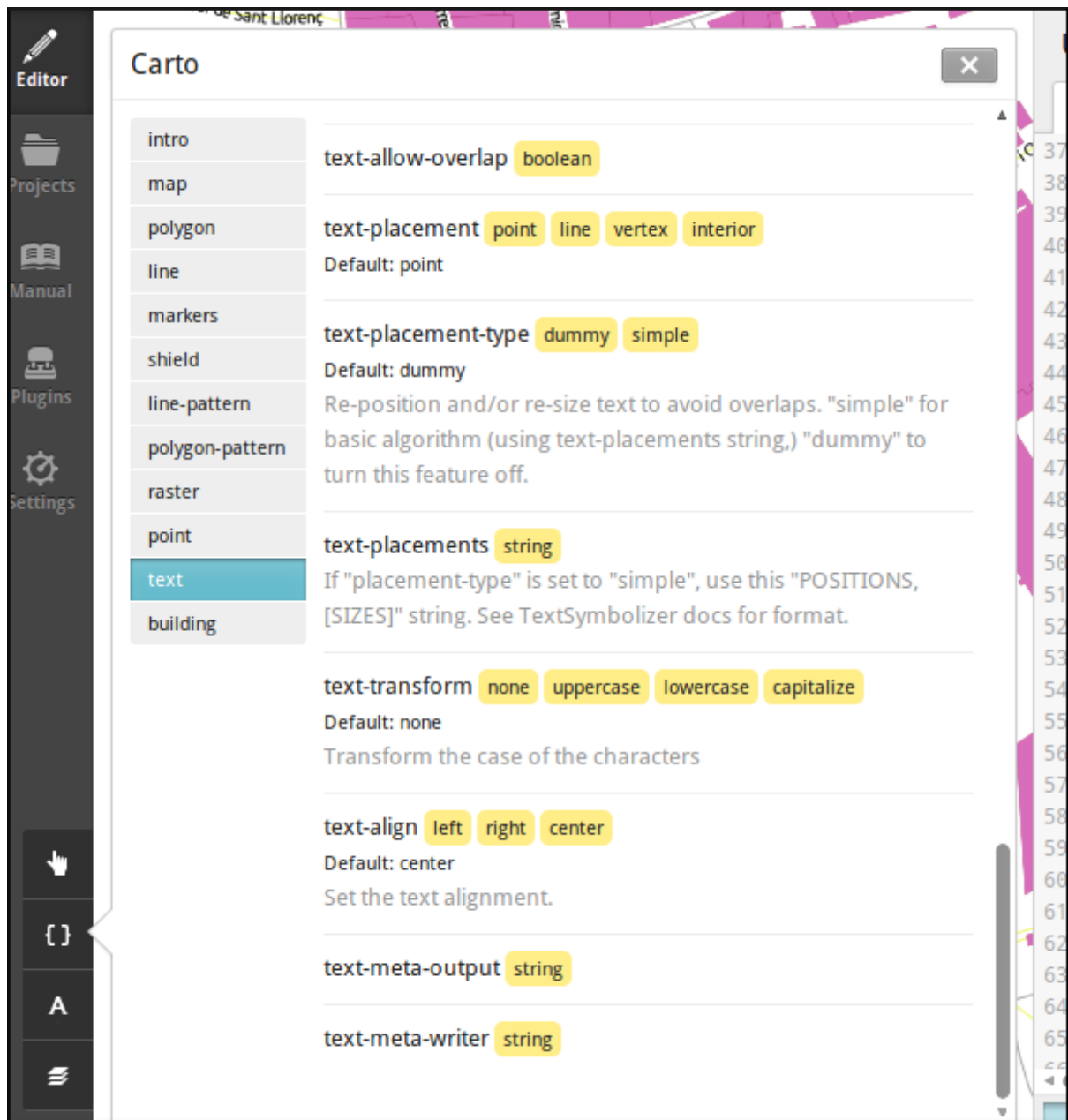
```
#calles_nombres {  
  text-name: "[name]";  
  text-face-name: @futura_med;  
  text-placement: line;  
}
```

Estos son los elementos mínimos para que una etiqueta aparezca en TileMill, aunque si vamos a la

ayuda del programa



y vemos la sección *text* veremos que las etiquetas tienen 30 opciones de configuración distintas.



## 4.7 Más sobre el lenguaje CartoCSS

### 4.7.1 Usando iconos como marcadores

Para usar los iconos deben referenciarse con una ruta relativa a la carpeta del proyecto

Por ejemplo para pintar puntos de interés

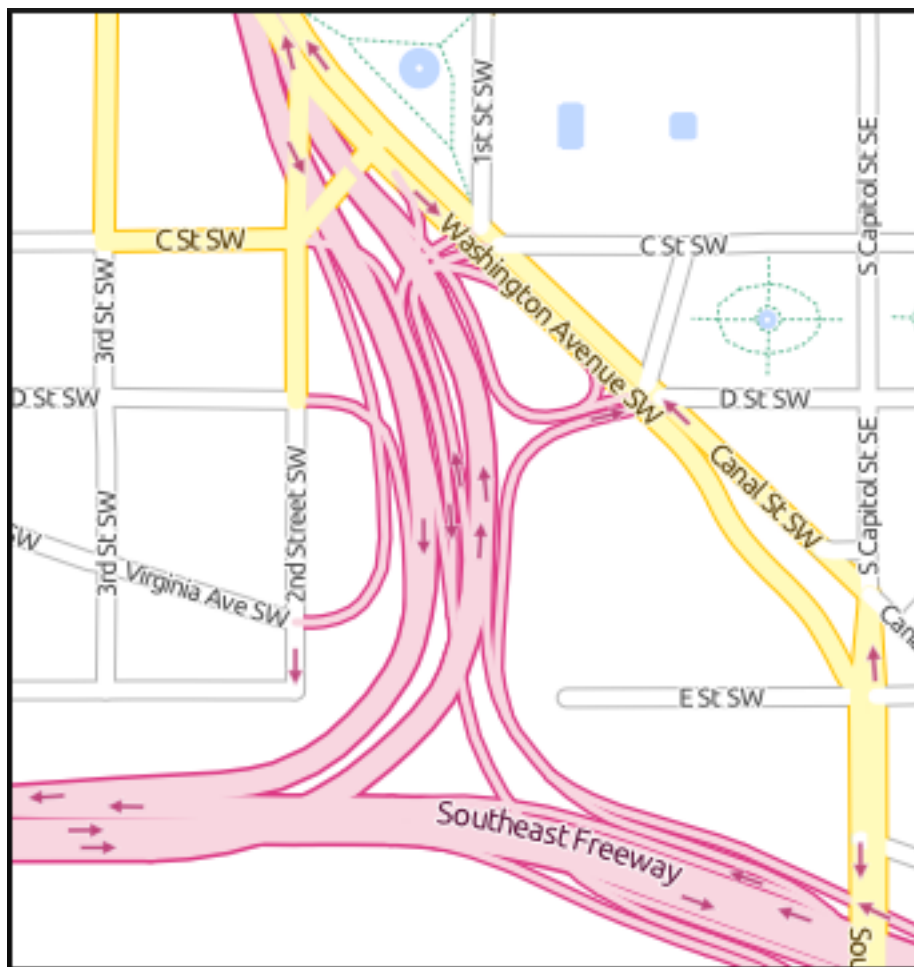
```
.amenity.place[zoom=15] {
  [type='police'] {
    point-file: url(../res/come-9px.png);
  }
  [type='fuel'] {
    point-file: url(../res/petrol-9px.png);
  }
}
```

```
[type='townhall'],  
[type='university'] {  
  point-file: url(../res/poi-9px.png);  
}  
}
```



#### 4.7.2 Pintando cajas de carretera

```
.highway[TYPE='motorway'] {  
  .line[zoom>=7] {  
    line-color: spin(darken(@motorway, 36), -10);  
    line-cap: round;  
    line-join: round;  
  }  
  .fill[zoom>=10] {  
    line-color: @motorway;  
    line-cap: round;  
    line-join: round;  
  }  
}  
  
.highway[zoom=13] {  
  .line[TYPE='motorway'] { line-width: 2.0 + 2; }  
  .fill[TYPE='motorway'] { line-width: 2.0; }  
}
```



## 4.8 Extra: OSM-Bright

Mapbox ha liberado un proyecto en [TileMill](#) para presentar datos de OSM con un estilo atractivo y bien documentado. Para poder usar este proyecto debemos descargar algunos datos base (línea de costa y límites administrativos) y luego ejecutar un *script* configurando algunos parámetros en un fichero.

En este caso usaremos como fuente de datos de nuevo el fichero de extracción de OSM de Nottingham, creando una nueva base de datos e importando el fichero usando **imposm** y un fichero de *mapping* de etiquetas proporcionado por OSM-Bright.

En primer lugar nos esplazamos a la carpeta de trabajo y activamos un entorno virtual con **imposm** 2.5.0 instalado:

```
$ cd /home/user/tallerimposm
$ source venv/bin/activate
(venv)$ imposm --version
Enabling Shapely speedups.
imposm 2.5.0
```

A continuación descargamos OSM-Bright y lo descomprimos obteniendo una carpeta `osm-bright-master` a la que nos desplazaremos:

```
(venv)$ wget -O osmbright.zip "https://github.com/mapbox/osm-bright/archive/master.zip"
(venv)$ unzip osmbright.zip
```

...

```
(venv) $ cd osm-bright-master
```

En esta carpeta descargaremos tres ficheros con cartografía (puede tardar un rato, uno de ellos es bastante pesado):

```
(venv) $ wget http://tilemill-data.s3.amazonaws.com/osm/coastline-good.zip
```

```
(venv) $ wget http://tilemill-data.s3.amazonaws.com/osm/shoreline_300.zip
```

```
(venv) $ wget http://mapbox-geodata.s3.amazonaws.com/natural-earth-1.3.0/physical/10m-lar
```

Creamos la base de datos `nott-osm-2` y activamos la extensión de PostGIS:

```
(venv) $ createdb -E UTF8 nott-osm-2
```

```
(venv) $ psql -d nott-osm-2 -c "create extension postgis"
```

Y ya estamos listos para cargar la base de datos usando **imposm** y el fichero `e mapping` que hay en la carpeta `imposm-mapping.py`:

```
(venv) $ imposm --user user -d nott-osm-2 -m imposm-mapping.py \
--read --write --optimize --deploy-production-tables \
/usr/local/share/data/osm/Nottingham.osm.bz2
```

El siguiente paso es configurar el fichero de OSM-Bright, para ello copiamos el archivo de ejemplo `configure.py.sample` y lo editamos con *medit*:

```
(venv) $ cp configure.py.sample configure.py
```

```
(venv) $ medit configure.py
```

En este fichero deberemos establecer:

- **imposm** como el importador que hemos usado
- Si queremos, un nombre para el proyecto
- Los parámetros de conexión a la base de datos
- Las coordenadas de la zona de interés, que para la zona de Nottingham son `-139015.78, 6966053.88, -119902.22, 6984403.83`



```

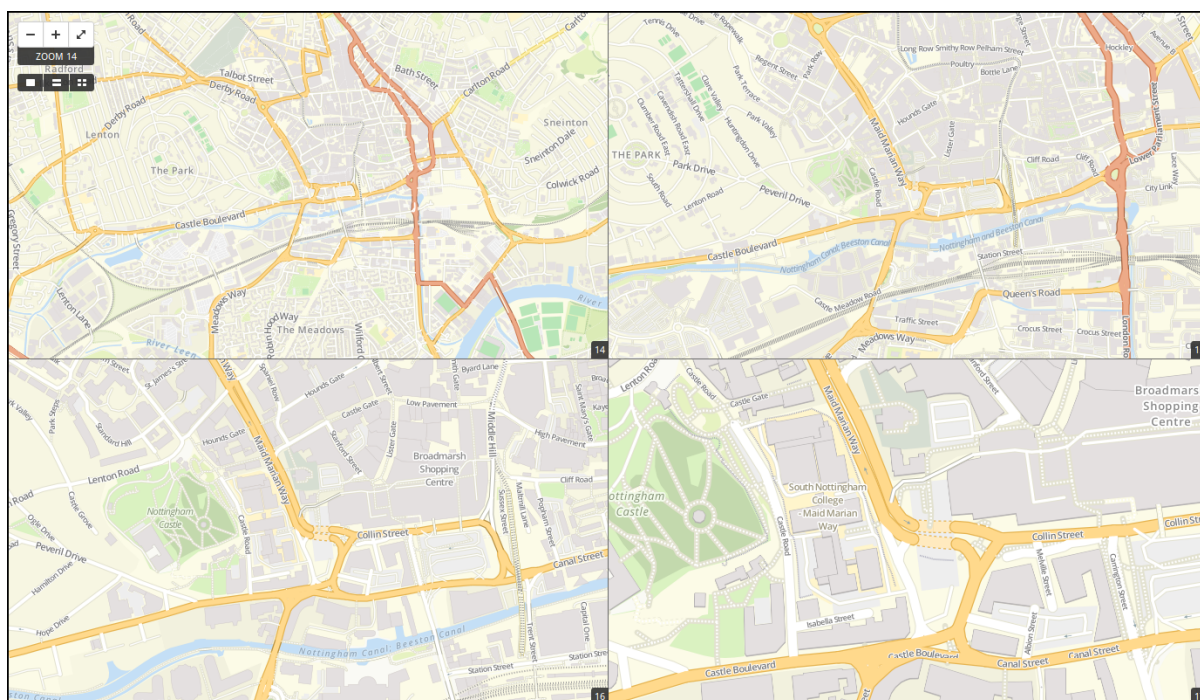
1#!/usr/bin/env python
2
3from os import path, getcwd
4from collections import defaultdict
5config = defaultdict(defaultdict)
6
7config["importer"] = "imposm" # either 'imposm' or 'osm2pgsql'
8
9# The name given to the style. This is the name it will have in the TileMill
10# project list, and a sanitized version will be used as the directory name
11# in which the project is stored
12config["name"] = "OSM Bright Nottingham"
13
14# The absolute path to your MapBox projects directory. You should
15# not need to change this unless you have configured TileMill specially
16config["path"] = path.expanduser("~/Documents/MapBox/project")
17
18# PostGIS connection setup
19# Leave empty for Mapnik defaults. The only required parameter is dbname.
20config["postgis"]["host"] = "localhost"
21config["postgis"]["port"] = "5432"
22config["postgis"]["dbname"] = "nott-osm-2"
23config["postgis"]["user"] = "user"
24config["postgis"]["password"] = "user"
25
26# Increase performance if you are only rendering a particular area by
27# specifying a bounding box to restrict queries. Format is "XMIN,YMIN,XMAX,YMAX"
28# in the same units as the database (probably spherical mercator meters). The
29# whole world is "-20037508.34 -20037508.34 20037508.34 20037508.34".
30# Leave blank to let Mapnik estimate.
31config["postgis"]["extent"] = "-139015.78, 6966053.88, -119902.22351674443, 6984403.837465665"
32
33# Land shapefiles required for the style. If you have already downloaded
34# these or wish to use different versions, specify their paths here.
35# OSM land shapefiles from MapBox are indexed for Mapnik and have blatant
36# errors corrected (eg triangles along the 180 E/W line). They can be download

```

Con el fichero correctamente configurado estamos listos para crear el proyecto de TileMill. Simplemente en la misma carpeta ejecutamos el *script* `make.py`:

```
(venv) $ python make.py
installing to /home/user/Documents/MapBox/project/OSMBrightNottingham
```

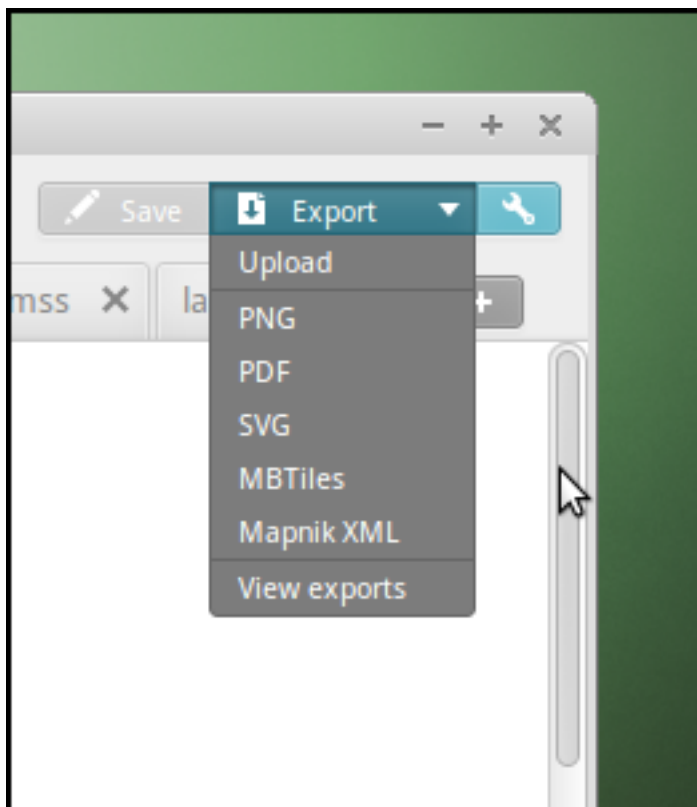
A continuación podemos abrir **TileMill** y deberíamos de tener un nuevo proyecto. Al abrir este proyecto probablemente tarde unos segundos en responder ya que tiene que traer de la base de datos un buen número de elementos. Tras unos instantes podremos navegar por la cartografía. Si activamos el *plugin* `tilemill-lots` podremos además ver cómo cambia la simbología en los diferentes niveles de *zoom*.



La configuración de este proyecto es amplia y compleja, pero activando y desactivando capas y usando el mencionado *plugin* podemos ir repasando cómo se han ido filtrando los diferentes tipos, y cómo los diseñadores han ido jugando con los niveles de *zoom* para definir la simbología desde las escalas más pequeñas hasta las de mayor detalle.

## 4.9 Exportando los mapas

**TileMill** genera diferentes productos cartográficos a partir del diseño realizado. Es decir, una vez estamos satisfechos con la simbología, podemos exportar el mapa en diferentes formatos y que responden a diferentes necesidades.



- Carga en [Mapbox](#): subir las teselas al servicio de este proveedor para poder insertarlo fácilmente en nuestras aplicaciones *web*.
- PNG: generar una única imagen, lista para insertar en cualquier informe, página *web* o cualquier otro documento.
- PDF: generar una única imagen dentro de un documento en formato PDF.
- MBTiles: renderizar las teselas en un fichero en formato [MBTiles](#). Este fichero luego puede utilizarse en cualquier aplicación que soporte este formato.
- SVG: la salida en este formato mantiene el formato vectorial y puede ser utilizado por ejemplo para impresiones de alta calidad al permitir su escalado sin perder detalle.
- XML de Mapnik: esta salida no devuelve una imagen sino un fichero para la librería [Mapnik](#) que podemos luego utilizar con otras herramientas. En este fichero se encuentra definido todo lo necesario para acceder a los datos y darles la simbología seleccionada.

#### 4.9.1 Montando un TMS

Como hemos visto, [TileMill](#) genera un fichero en formato [MBTiles](#) para poder llevar nuestra renderización de un sitio a otro fácilmente. En ocasiones por otro lado, resulta conveniente exportar las teselas almacenadas en la base de datos de este formato a una estructura de carpetas siguiendo el estándar [TMS](#), ya que de esta forma puede resultar accesible por ejemplo por un cliente *web* como [OpenLayers](#).

Para extraer las imágenes de este tipo de ficheros podemos usar la herramienta [mbutil](#), desarrollada por [Mapbox](#) y que ofrece un ejecutable para la línea de comandos que exporta el fichero fácilmente.

Para instalarla teniendo un entorno virtual activado basta con ejecutar:

```
(venv) $ pip install mbutil
```

Para generar un TMS ejecutamos:

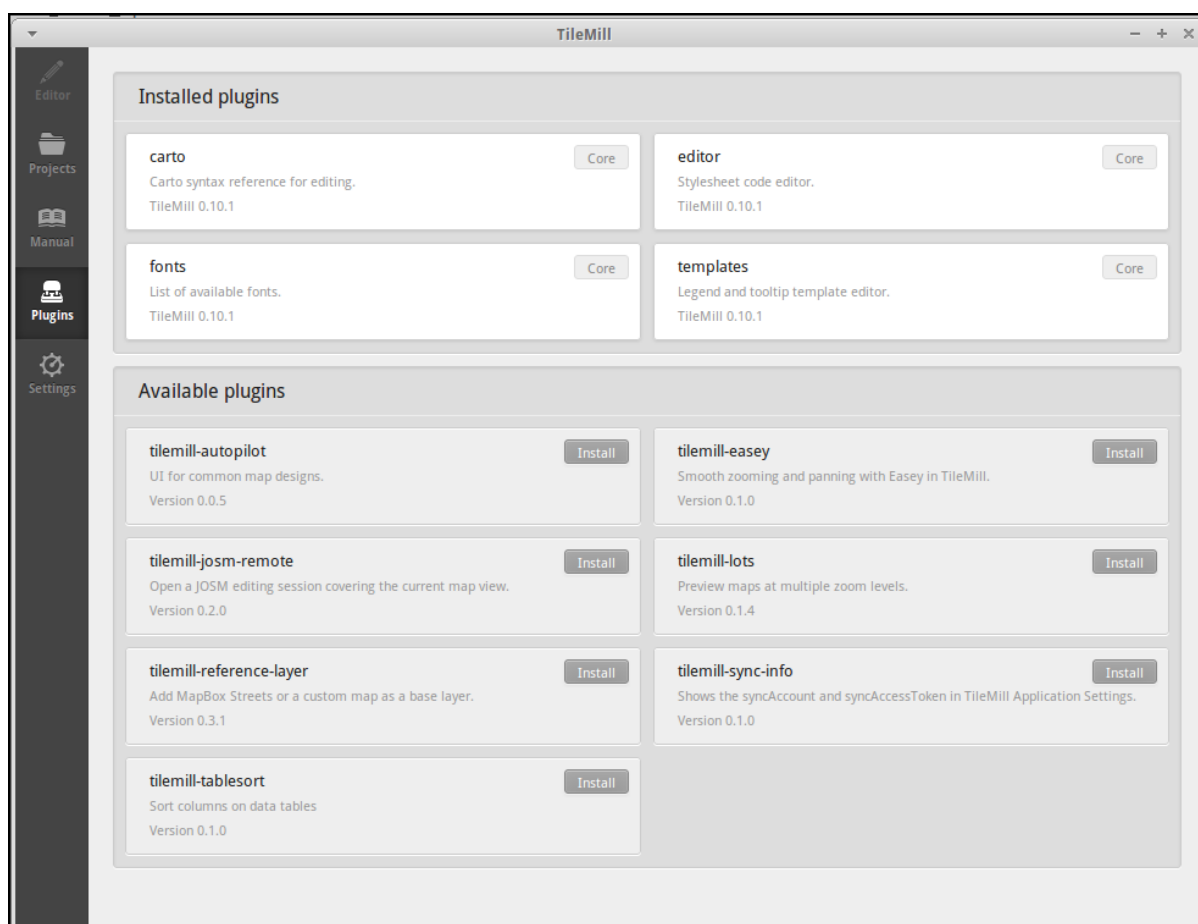
```
(venv) $ mb-util --scheme=tms exportado.mbtiles directorio/
```

## 4.10 Otras alimañas

### 4.10.1 Extensiones

**TileMill** es un *software* que dispone de extensiones conocidas como *plugins*. Esta funcionalidad se introdujo a partir de la versión 0.9 aprovechando que **NodeJS**, el *software* sobre el que está construido., facilitó el mecanismo para gestionarlos.

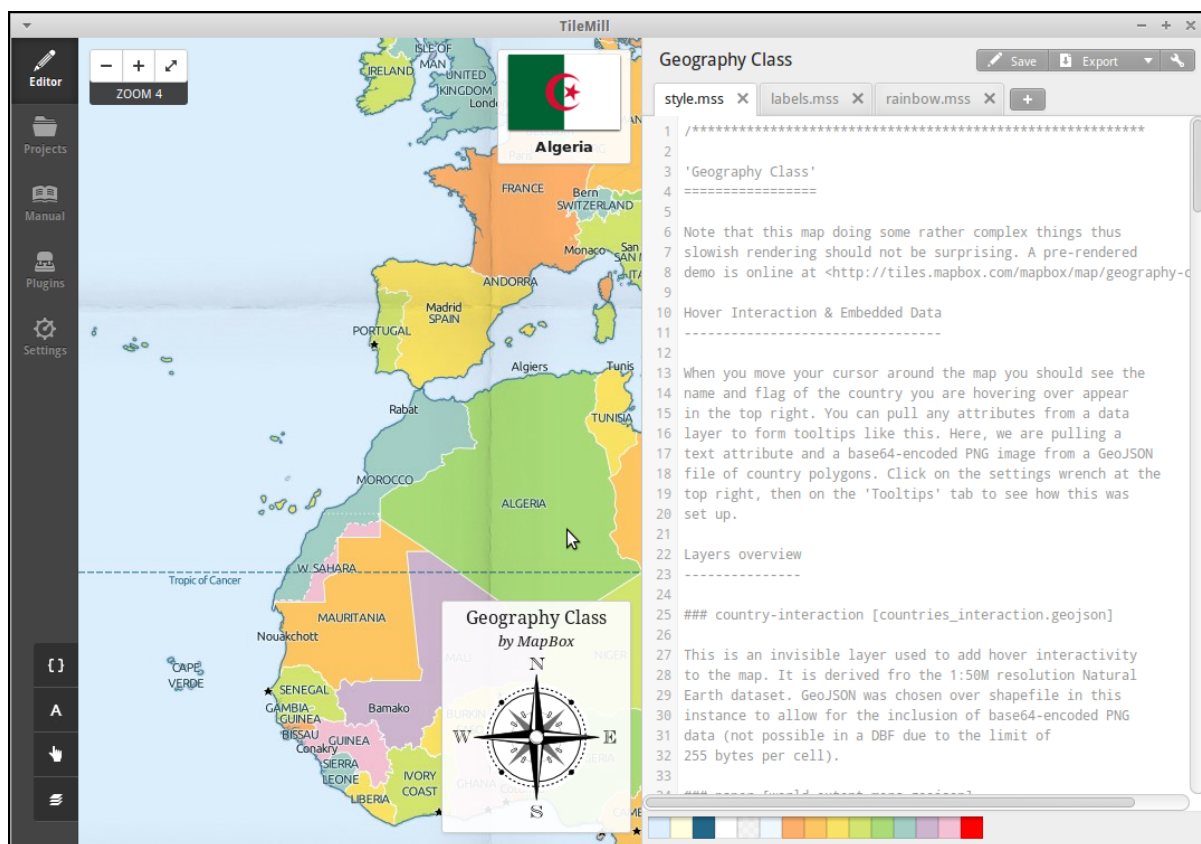
Las funcionalidades principales de **TileMill** se agrupan en cuatro extensiones básicas que no se pueden desactivar (marcadas como *Core*) y 7 extensiones opcionales disponibles que añaden funcionalidades diversas como poder ver el mapa en varios niveles de *zoom* a la vez, poder utilizar mapas de **Mapbox** como mapa base o poder ordenar las columnas en la vista de tabla.



### 4.10.2 Mapas interactivos

**TileMill** admite cierta interactividad que se puede configurar para cada mapa. Esta interactividad solo es útil si se va a subir el mapa al servicio de alojamiento de teselas de **Mapbox**, ya que en los productos

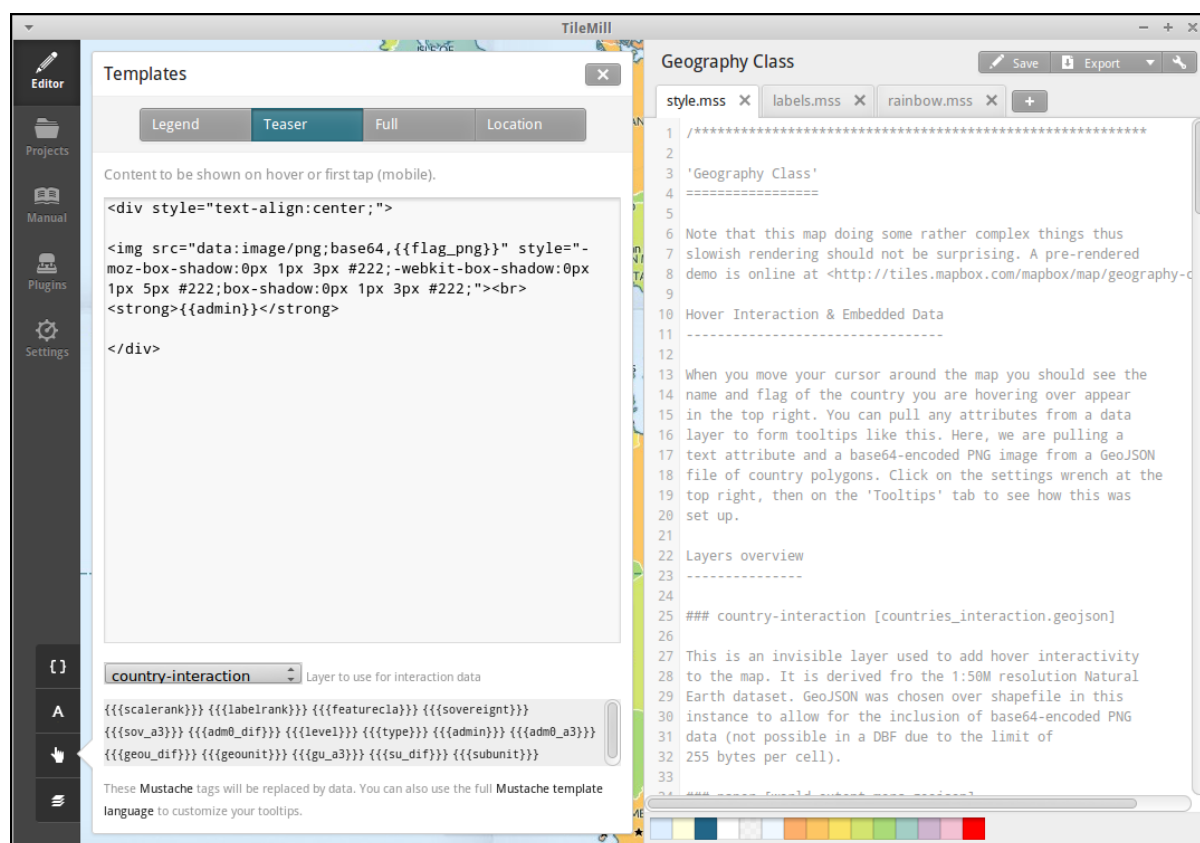
generados revisados (imágenes, MBTiles, etc.) no se puede acceder a esta funcionalidad. El proyecto *Geography Class* está cargado por defecto en la instalación de **TileMill** y es un ejemplo excelente de interacción en el mapa.



Las dos características más interesantes a configurar son:

- **Leyenda:** aparecerá sobre el mapa en la esquina inferior derecha. Se trata de un documento HTML estático que deberemos editar directamente en TileMill.
- **Tooltip:** se configura una plantilla en HTML en la que se puede hacer referencia a los valores del objeto sobre el que el ratón se posiciona. El *tooltip* solo puede acceder a los campos de una única capa.

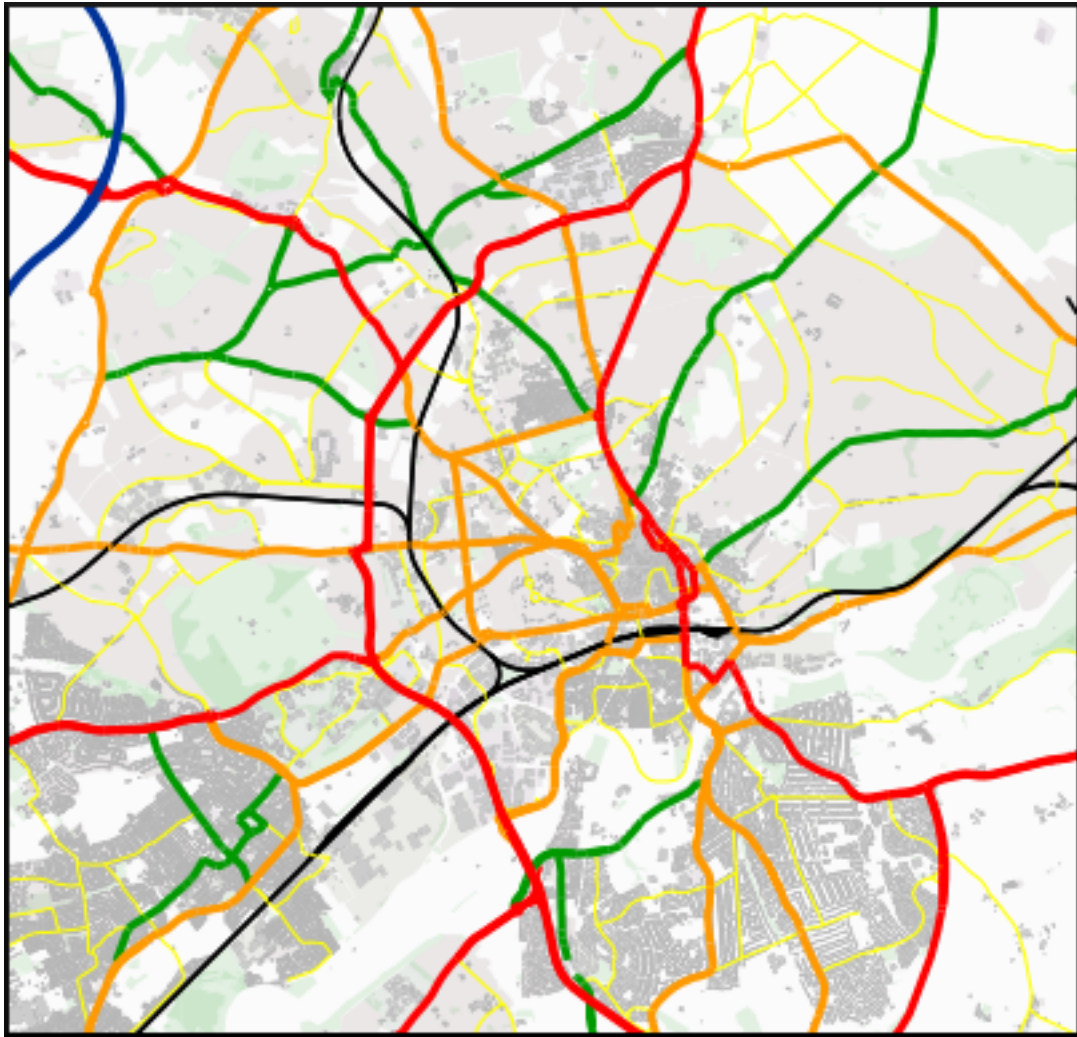
Estas opciones se establecen haciendo clic sobre el icono con forma de mano en la parte inferior izquierda de la interfaz de TileMill.



## 4.11 Ejercicio

Empleando la cartografía existente en la base de datos `nott-osm` se requiere realizar un diseño cartográfico que destaque los siguientes elementos:





#### 4.11.1 Vías

Se destacará la red viaria de forma que exista una clasificación de carreteras que permita identificar visualmente los siguientes tipos:

- *motorway* y *motorway\_link*
- *trunk* y *trunk\_link*
- *primary* y *primary\_link*
- *secondary* y *secondary\_link*
- *tertiary*
- *residential*
- *pedestrian*, *path* y *cycleway*
- *rail*
- *resto*

**Tabla** osm\_roads



### 4.11.2 Edificios

Se destacará el entramado urbano que permita diferenciar los siguientes tipos:

- *city\_hall, conservatory y museum*
- *college, library y university*
- *flat, flats, house y residential*
- *industrial, light\_industry\_units*
- *retail, shop, shopping\_mall y shops*
- *resto*

**Tabla** osm\_buildings

### 4.11.3 Áreas



Se destacarán con un color distinto debajo de la capa de edificios las siguientes áreas urbanas:

- *commercial, retail*
- *hospital*
- *industrial*
- *nature\_reserve, park, wood*
- *residential*
- *university*

En caso de existir una correspondencia entre un tipo de edificios y un área, el área deberá ser un 20 % más oscura que el edificio pero emplear el mismo tono, además los polígonos de área deberán tener una opacidad del 10 %.

¿Qué tabla hay que emplear? Abre la aplicación *Geospatial* → *Databases* → **pgAdminIII** y explora la base de datos **nott-osm** para averiguar dónde está la información.

### 4.11.4 Lugares de interés

Para la simbología de lugares de interés emplearemos la biblioteca de símbolos **Maki** desarrollada también por la empresa *Mapbox* y el código que hemos visto en la sección *Usando iconos como marcadores*. Se representarán las comisarías de policía  y los hospitales  cada uno con su símbolo puntual distintivo.

La biblioteca de iconos *Maki* ha sido especialmente diseñada para ser empleada con **TileMill** y proporciona los iconos tanto en formato raster (.png) como en formato vectorial (.svg). En la página *web* está el enlace para descargar la biblioteca en ambos formatos en un archivo .zip. También se puede encontrar el [enlace a un tutorial sobre las posibilidades de empleo en TileMill](#).

Para usar las imágenes de Policía y Hospital:

1. Crearemos una carpeta `imgs` dentro del directorio del proyecto que podemos encontrar en `/home/user/Documents/MapBox/project/cfp2014`.

2. Descomprimos el archivo `.zip`, que es una copia del repositorio de *GitHub*, y navegamos hasta la carpeta `renders`.
3. Copiaremos las imágenes `police*.png` y `hospital*.png` en el directorio `imgs` que hemos creado anteriormente.

---

**Importante:** *¿Qué tabla hay que emplear?* Abre la aplicación *Geospatial* → *Databases* → **pgAdminIII** y explora la base de datos **nott-osm** para averiguar dónde está la información.

---

## 4.12 Referencias y enlaces

- [Página principal de TileMill](#)
- [Referencia del lenguaje CartoCSS](#)
- [Estilo OSM Bright de Mapbox para cartografía de OpenStreetMap](#)

Este taller pretende ser una breve introducción a un flujo de trabajo que permite tener mapas funcionales y estéticos. A lo largo del taller vamos a ver una serie de herramientas FOSS4G que permiten crear estos mapas de una manera sencilla y con un lenguaje de diseño cartográfico especialmente próximo a los desarrolladores web.

El taller tiene tres partes en las que veremos:

- Por qué la base de datos del proyecto OpenStreetMap (OSM) es tan relevante y cómo obtengo sus datos y cómo se trabaja con JOSM.
- La herramienta *ImpOSM* que convierte el XML de OSM en una base de datos geográfica.
- Cómo hacer buenos mapas para la web con Mapnik y su lenguaje Carto.

---

**Nota:** La url del taller es <http://bit.ly/cfp2014-jitw>

---



---

### ¿Qué es Geoinquietos Valencia?

---

La idea detrás de Geoinquietos Valencia es la misma que la que hay detrás de los otros grupos de Geoinquietos (Barcelona, Cantabria, Zona norte y Madrid) se trata de compartir la pasión que tenemos por todo lo geo y mantenernos al tanto de noticias y novedades. ¿Cómo? Pues muy fácil, asistiendo a las reuniones que tienen lugar aproximadamente cada mes.

La estructura de estas reuniones no puede ser más sencilla... nos reunimos en un sitio con suficiente espacio y a ser posible conexión a Internet y a ser muy posible proyector, para tratar los temas que se hayan propuesto con anterioridad usando [una wiki](#) como espacio de colaboración ágil. Todo el mundo está invitado a añadir contenidos para enriquecer la página.

Estos temas a tratar también pueden estar aderezados con alguna pequeña charla de uno de los asistentes en la que puede explicar en que consiste su trabajo, su hobby, algo que le haya resultado curioso, pero todo relacionado con el mundillo geo.



---

### Facilitadores

---

- Pedro-Juan Ferrer @vehrka · pferrer@osgeo.org
- Jorge Sanz @xurxosanz · jsanz@osgeo.org





---

### Autores

---

- Pedro-Juan Ferrer @vehrka · pferrer@osgeo.org
  - Project Manager en Omnium SI
  - Geofriki
- Iván Sanchez @realivansanchez · ivan@sanchezortega.es
  - Developer en Aptomar
  - Más geofriki aún si cabe
- Santiago Tramoyeres @santracraus
  - Freelance for hire
  - Muy friki de lo suyo
- Jorge Sanz @xurxosanz · jsanz@osgeo.org
  - GISguy en Prodevelop
  - Geofriki



---

### Licencia

---

Excepto donde quede reflejado de otra manera, la presente documentación se halla bajo licencia [Creative Commons Reconocimiento Compartir Igual](#)

