switch2osm

Make the switch to OpenStreetMap

Up to date

OpenStreetMap is updated every minute of every hour of every day, and these updates are available to you in real-time.

Our fantastic community is making OpenStreetMap better right now. If there are features you need - you can add them and see them live within minutes.

Manually building a tile server (12 04)

This page describes how to install, setup and configure all the necessary software to operate your own tile server. The step-by-step instructions are written for Ubuntu Linux 12.04 LTS (Precise Pangolin), however they should transfer fairly straightforwardly to other versions of Ubuntu or Linux distributions.

This howto is based on an earlier version for Ubuntu Linux 10.04

Software installation

The OSM tile server stack is a collection of programs and libraries that work together to create a tile server. As so often with OpenStreetMap, there are many ways to achieve this goal and nearly all of the components have alternatives that have various specific advantages and disadvantages. This tutorial describes the most standard version that is also used on the main OpenStreetMap.org tile server.

It consists of 5 main components: Mod_tile, renderd, mapnik, osm2pgsql and a postgresql/postgis database. Mod_tile is an apache module, that serves cached tiles and decides which tiles need re-rendering – either because they are not yet cached or because they are outdated. Renderd provides a priority queueing system for rendering requests to manage and smooth out the load from rendering requests. Mapnik is the software library that does the actual rendering and is used by renderd.

In order to build these components, a variety of dependencies need to be installed first.

Mapnik requires a recent version of Boost, which can be installed on Ubuntu like this:

```
sudo add-apt-repository ppa:mapnik/boost
sudo apt-get update
sudo apt-get install libboost-dev libboost-filesystem-dev libboost-program-options-dev libboost-python-dev libboost-regex-dev libboost
```

The remaining dependencies can be installed with the following instructions

```
sudo apt-get install subversion git-core tar unzip wget bzip2 build-essential autoconf libtool libxml2-dev libgeos-dev libpq-dev libbz
```

Installing postgresql / postgis

On Ubuntu there are pre-packaged versions of both postgis and postgresql, so these can simply be installed via the Ubuntu package manager.

```
sudo apt-get install postgresql-9.1-postgis postgresql-contrib postgresql-server-dev-9.1
```

Now you need to create a postgis database. The defaults of various programs assume the database is called gis and we will use the same convention in this tutorial, although this is not necessary. Substitute your username for *username* in the two places below. This should be the username that will render maps with Mapnik.

```
sudo -u postgres -i
createuser username # answer yes for superuser (although this isn't strictly necessary)
createdb -E UTF8 -O username gis
exit
```

Set up PostGIS on the postresql database:

```
psql -f /usr/share/postgresql/9.1/contrib/postgis-1.5/postgis.sql -d gis
```

This should respond with many lines ending with:

```
...
CREATE FUNCTION
COMMIT
...
DROP FUNCTION
```

Give your newly-created user permission to access some of the PostGIS extensions' data. Make sure you replace username with your user's name:

```
psql -d gis -c "ALTER TABLE geometry_columns OWNER TO username; ALTER TABLE spatial_ref_sys OWNER TO username;"
```

Installing osm2pgsql

Although there might be a osm2pgsql package in the repository, it is likely rather old and so we need to compile a newer one from source

```
mkdir ~/src
cd ~/src
git clone git://github.com/openstreetmap/osm2pgsql.git
```

```
cd osm2pgsql
./autogen.sh
./configure
make
sudo make install
psql -f /usr/local/share/osm2pgsql/900913.sql -d gis
```

Install Mapnik library

Next, we need to install the Mapnik library. Mapnik is used to render the OpenStreetMap data into the tiles used for an OpenLayers web map.

Build the Mapnik library from source:

```
cd ~/src
git clone git://github.com/mapnik/mapnik
cd mapnik
git branch 2.0 origin/2.0.x
git checkout 2.0

python scons/scons.py configure INPUT_PLUGINS=all OPTIMIZATION=3 SYSTEM_FONTS=/usr/share/fonts/truetype/
python scons/scons.py
sudo python scons/scons.py install
sudo ldconfig
```

Verify that Mapnik has been installed correctly:

```
python
>>> import mapnik
>>>
```

If python replies with the second chevron prompt >>> and without errors, then Mapnik library was found by Python. Congratulations!

Install mod_tile and renderd

Compile the mod_tile source code:

```
cd ~/src
git clone git://github.com/openstreetmap/mod_tile.git
cd mod_tile
./autogen.sh
./configure
make
sudo make install
sudo make install-mod_tile
sudo ldconfig
```

Install Mapnik style-sheet

Next, we need to install the OpenStreetMap Mapnik tools, which include the default style file and tools to help Mapnik render OpenStreetMap data:

```
cd ~/src
svn co http://svn.openstreetmap.org/applications/rendering/mapnik mapnik-style
```

Mapnik uses prepared files to generate coastlines and ocean areas for small scale maps since it is faster than reading the entire database for this information. Downloading the coastline data requires about 400Mb of download.

```
cd ~/src/mapnik-style
sudo ./get-coastlines.sh /usr/local/share
```

Software configuration

Now that all of the necessary software is installed, you will need to configure some of the software to work correctly.

Configure mapnik style-sheet

In order for mapnik to find the correct postgis database and the coast line data, you will need to configure the mapnik style-sheet to your local settings. There are a number of different style-sheets publically available and you can of course create your own ones. Each style-sheet might be slightly different in how to configure it and so this page describes how to configure the “default” OpenStreetMap style-sheet that was installed with the mapnik-tools.

In your style-sheet directory (e.g. ~/src/mapnik-style) there should be a directory inc. There are a number of files you need to adapt in this directory:

```
cd inc
cp fontset-settings.xml.inc.template fontset-settings.xml.inc
cp datasource-settings.xml.inc.template datasource-settings.xml.inc
cp settings.xml.inc.template settings.xml.inc
```

Now you need to modify each of these files:

settings.xml.inc

replace

```
<!ENTITY symbols "%(symbols)s">
```

with

```
<!ENTITY symbols "symbols">
```

replace

```
<!ENTITY osm2pgsql_projection "&srs%(epsg)s;">
```

with (assuming you will be using the defaul projection 900913)

```
<!ENTITY osm2pgsql_projection "&srs900913;">
```

replace

```
<!ENTITY dwithin_node_way "&dwithin_%(epsg)s;">
```

with (assuming you will be using the default projection 900913)

```
<!ENTITY dwithin_node_way "&dwithin_900913;">
```

replace

```
<!ENTITY world_boundaries "%(world_boundaries)s">
```

with (assuming you installed the coast line data in /usr/local/share/world_boundaries)

```
<!ENTITY world_boundaries "/usr/local/share/world_boundaries">
```

replace

```
<!ENTITY prefix "%(prefix)s">
```

with (assuming you are using the default database table prefix)

```
<!ENTITY prefix "planet_osm">
```

datasource-settings.xml.inc

In this file you will need to enter your database settings. You are running postgresql on the same machine as the rendering stack, so you can comment out the parameters "password", "host" and "port" with an HTML-style comment. This will enable mapnik to use the "unix local user" as an authentication method.

Change the "dbname" from "%(dbname)s" to "gis", "estimate_extent" to "false", and "extent" to "-20037508,-19929239,20037508,19929239″ so that the file now reads:

```
<!--
Settings for your postgres setup.

Note: feel free to leave password, host, port, or use blank
-->

<Parameter name="type">postgis</Parameter>
<!-- <Parameter name="password">%(password)s</Parameter> -->
<!-- <Parameter name="host">%(host)s</Parameter> -->
<!-- <Parameter name="port">%(port)s</Parameter> -->
<!-- <Parameter name="user">%(user)s</Parameter> -->
<Parameter name="dbname">gis</Parameter>
<!-- this should be 'false' if you are manually providing the 'extent' -->
<Parameter name="estimate_extent">false</Parameter>
<!-- manually provided extent in epsg 900913 for whole globe -->
<!-- providing this speeds up Mapnik database queries -->
<Parameter name="extent">-20037508,-19929239,20037508,19929239</Parameter>
```

fontset-settings.xml.inc

This file contains font definitions, and information about how to change the default font. It is recommended that you don't edit this right now.

Configure renderd

Change the the renderd settings by editing the /usr/local/etc/renderd.conf and change the following lines like so (remember to change username to your user's name):

```
socketname=/var/run/renderd/renderd.sock
plugins_dir=/usr/local/lib/mapnik/input
font_dir=/usr/share/fonts/truetype/ttf-dejavu
XML=/home/username/src/mapnik-style/osm.xml
HOST=localhost
```

Create the files required for the mod_tile system to run (remember to change username to your user's name):

```
sudo mkdir /var/run/renderd
sudo chown username /var/run/renderd
sudo mkdir /var/lib/mod_tile
sudo chown username /var/lib/mod_tile
```

Configure mod_tile

Next, we need to tell the Apache web server about our new mod_tile installation by creating the file /etc/apache2/conf.d/mod_tile and adding one line:

```
LoadModule tile_module /usr/lib/apache2/modules/mod_tile.so
```

Also, Apache's default website configuration file needs to be modified to include mod_tile settings. Modify the file /etc/apache2/sites-available/default to include the following lines immediately after the admin e-mail address line:

```
LoadTileConfigFile /usr/local/etc/renderd.conf
ModTileRenderdSocketName /var/run/renderd/renderd.sock
# Timeout before giving up for a tile to be rendered
ModTileRequestTimeout 0
# Timeout before giving up for a tile to be rendered that is otherwise missing
ModTileMissingRequestTimeout 30
```

Tuning your system

A tile server can put a lot of load on hard- and software. The default settings may therefore not be appropriate and a significant improvement can potentially be achieved through tuning various parameters.

Tuning postgresql

The default configuration for PostgreSQL 9.1 needs to be tuned for the amount of data you are about to add to it. Edit the file /etc/postgresql/9.1/main/postgresql.conf and make the following changes:

```
shared_buffers = 128MB
checkpoint_segments = 20
maintenance_work_mem = 256MB
autovacuum = off
```

These changes require a kernel configuration change, which needs to be applied every time that the computer is rebooted. As root, edit /etc/sysctl.conf and add these lines near the top after the other "kernel" definitions:

```
# Increase kernel shared memory segments - needed for large databases
kernel.shmmax=268435456
```

Reboot your computer. Run this:

```
sudo sysctl kernel.shmmax
```

and verify that it displays as 268435456.

Loading data into your server

Get the latest OpenStreetMap data

Retrieve a piece of OpenStreetMap data from http://planet.openstreetmap.org/. Since the whole planet is at least 18GB when compressed, there are links to smaller country or state sized extracts on that page. Since it is smaller and faster to interact with, the PBF file format is preferable when available. In this case we'll download the entire planet file by issuing the following command:

```
cd planet
wget http://planet.openstreetmap.org/pbf/planet-latest.osm.pbf
```

Importing data into the database

With the conversion tool compiled and the database prepared, the following command will insert the OpenStreetMap data you downloaded earlier into the database. This step is very disk I/O intensive and will take anywhere from 10 hours on a fast server with SSDs to several days for the full planet depending on the speed of the computer performing the import. For smaller extracts the import time is much faster accordingly, and you may need to experiment with different -C values to fit within your machine's available memory.

```
osm2pgsql --slim -d gis -C 16000 --number-processes 3 ~/planet/planet-latest.osm.pbf
```

Let's have a look at your data import as it progresses. The first part of the osm2pgsql output looks scary, but is normal:

```
Using projection SRS 900913 (Spherical Mercator)
Setting up table: planet_osm_point
NOTICE: table "planet_osm_point" does not exist, skipping
NOTICE: table "planet_osm_point_tmp" does not exist, skipping
Setting up table: planet_osm_line
NOTICE: table "planet_osm_line" does not exist, skipping
NOTICE: table "planet_osm_line_tmp" does not exist, skipping
Setting up table: planet_osm_polygon
NOTICE: table "planet_osm_polygon" does not exist, skipping
NOTICE: table "planet_osm_polygon_tmp" does not exist, skipping
Setting up table: planet_osm_roads
NOTICE: table "planet_osm_roads" does not exist, skipping
NOTICE: table "planet_osm_roads_tmp" does not exist, skipping
Mid: pgsql, scale=100, cache=4096MB, maxblocks=524289*8192
Setting up table: planet_osm_nodes
NOTICE: table "planet_osm_nodes" does not exist, skipping
NOTICE: CREATE TABLE / PRIMARY KEY will create implicit index "planet_osm_nodes_pkey" for table "planet_osm_nodes"
Setting up table: planet_osm_ways
NOTICE: table "planet_osm_ways" does not exist, skipping
NOTICE: CREATE TABLE / PRIMARY KEY will create implicit index "planet_osm_ways_pkey" for table "planet_osm_ways"
Setting up table: planet_osm_rels
NOTICE: CREATE TABLE / PRIMARY KEY will create implicit index "planet_osm_rels_pkey" for table "planet_osm_rels"
```

Don't be concerned by the NOTICE: entries above. All normal. Next, osm2pgsql will start reading the compressed planet file.

```
Reading in file: /home/user/planet/planet-latest.osm.bz2
```

As osm2pgsql reads the planet file it will give progress reports. The line below will refresh every few seconds and update the numbers in brackets. This part of the import takes a long time. Depending on your server, it will take between hours and days.

```
Processing: Node(10140k 200.1k/s) Way(0k 0.00k/s) Relation(0k 0.00/s)
```

As the import proceeds, first the Node number will increase untill they are all imported, then the Way number and finally the Relation number. The speed at which the import occurs depends strongly on the hardware available and if the node cache fits into ram. That said, if your node cache is big enough, you can probably expect the speed of nodes to be on the order of a few hundred k per second for nodes, a few 10s of k per second for ways and a few hundred per second for relations. If the node cache does not fit into ram or you have set the cache value to low, the process can be an order of magnitude slower. If you are importing the full planet, the import can take days. If you are only importing single countries, the time should be cut down to minutes or hours. Do not interrupt the import process unless you have decided to start over again from the beginning.

```
Processing: Node(593072k) Way(45376k) Relation(87k)
Exception caught processing way id=110802
Exception caught processing way id=110803
Processing: Node(593072k) Way(45376k) Relation(474k)
```

The exceptions shown above are due to minor errors in the planet file. The planet import is still proceeding normally.

The next stage of the osm2pgsql planet import process also will take between hours and days, depending on your hardware. It begins like this:

```
Node stats: total(593072533), max(696096737)
Way stats: total(45376969), max(55410575)
Relation stats: total(484528), max(555276)
Going over pending ways
processing way (752k)
```

The processing way number should update approximately each second.

```
Going over pending relations
node cache: stored: 515463899(86.91%), storage efficiency: 96.01%, hit rate: 85.97%
Committing transaction for planet_osm_roads
Committing transaction for planet_osm_line
Committing transaction for planet_osm_polygon
Sorting data and creating indexes for planet_osm_line
Sorting data and creating indexes for planet_osm_roads
Sorting data and creating indexes for planet_osm_polygon
Committing transaction for planet_osm_point
Sorting data and creating indexes for planet_osm_point
Stopping table: planet_osm_nodes
Stopping table: planet_osm_ways
Stopping table: planet_osm_rels
Building index on table: planet_osm_rels
Stopped table: planet_osm_nodes
Building index on table: planet_osm_ways
Stopped table: planet_osm_rels
Completed planet_osm_point
Completed planet_osm_roads
Completed planet_osm_polygon
Completed planet_osm_line
Stopped table: planet_osm_ways

Osm2pgsql took 86400s overall
```

The number of nodes, ways and relations processed will obviously differ by the size of the data extract you are using and the date of the data dump. The numbers shown here are not reflective of the full planet import, which is substantially larger.

Starting your tileserver

Now that everything is installed, set-up and loaded, you can start up your tileserver and hopefully everything is working. We'll run it interctively first, just to make sure that everything's working properly:

```
sudo mkdir /var/run/renderd
sudo chown username /var/run/renderd
renderd -f -c /usr/local/etc/renderd.conf
```

and on a different session:

```
sudo /etc/init.d/apache2 restart
```

If any FATAL errors occur you'll need to double-check any edits that you made earlier.

If not, try and browse to http://yourserveraddress/osm_tiles/0/0/0.png to see if a small picture of the world appears. The actual map tiles are being created as "metatiles" beneath the folder /var/lib/mod_tile.

If it does, you can stop the interactive renderd process and configure it to run automatically at machine startup as a daemon.

```
sudo cp  ~/src/mod_tile/debian/renderd.init /etc/init.d/renderd
sudo chmod u+x /etc/init.d/renderd
```

Edit the /etc/init.d/renderd file as root – you'll need to make a couple of changes to the DAEMON and DAEMON_ARGS lines so that they read:

```
DAEMON=/usr/local/bin/$NAME
DAEMON_ARGS="-c /usr/local/etc/renderd.conf"
```

Also, you'll need to change references to www-data so that they match your username – change "www-data" to what you changed "username" to in other files.

You should now be able to start mapnik by doing the following:

```
sudo /etc/init.d/renderd start
```

and stop it:

```
sudo /etc/init.d/renderd stop
```

Logging information is now written to /var/log/syslog instead of to the terminal.

Next, add a link to the interactive startup directory so that it starts automatically:

```
sudo ln -s /etc/init.d/renderd /etc/rc2.d/S20renderd
```

and then restart your server, browse to http://yourserveraddress/osm_tiles/0/0/0.png and everything should be working! You can also go to the page http://yourserveraddress/mod_tile which should give you some stats about your tileserver.

Next, you might want to have a look at something in the using tiles section to create a map that uses your new tile server.

Acknowledgements

With thanks to Ian Dees and Richard Weait

/pre

Search:

Search

Links

- OpenStreetMap
- OSM data download (planet.osm)
- OSM Foundation