**Configuration of**

**Open Street Map Server**

**to render**

**Indian Map with Disputed Boundaries**



**Author: Shatrughan Saxena**

**azaadshatru@yahoo.com**

**Contents**

[**Manually building a tile server (18.04 LTS)** 3](#_Toc14332886)

[**Software installation** 3](#_Toc14332887)

[**Installing postgresql / postgis** 4](#_Toc14332888)

[**Installing osm2pgsql** 5](#_Toc14332889)

[**Mapnik** 6](#_Toc14332890)

[**Install mod\_tile and renderd** 6](#_Toc14332891)

[**Stylesheet configuration** 7](#_Toc14332892)

[**Download OSM Bright** 8](#_Toc14332893)

[**Configuring OSM Bright** 9](#_Toc14332894)

[**Compiling the stylesheet** 9](#_Toc14332895)

[**Setting up your webserver** 10](#_Toc14332896)

[Configure renderd 10](#_Toc14332897)

[Configure mod\_tile 10](#_Toc14332898)

[**Tuning your system** 11](#_Toc14332899)

[Tuning postgresql 11](#_Toc14332900)

[**Loading data** 11](#_Toc14332901)

[**Running renderd for the first time** 13](#_Toc14332902)

[**Running renderd in the background** 14](#_Toc14332903)

[**Viewing tiles** 15](#_Toc14332904)

[**Leaflet** 15](#_Toc14332905)

[**Correcting Indian Map** 17](#_Toc14332906)

[**Disputed border between India, Pakistan, and India, China** 17](#_Toc14332907)

[**Install osm-boundaries package** 17](#_Toc14332908)

[**Edit OSMBright.xml Style File** 20](#_Toc14332909)

[**Correct Map of India** 23](#_Toc14332910)

[**Manually building a tile server (18.04 LTS)**](https://switch2osm.org/manually-building-a-tile-server-18-04-lts/)

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](https://www.ubuntu.com/) 18.04 LTS (Bionic Beaver).

## **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 similar to that used on the main OpenStreetMap.org tile servers.

Note that these instructions are have been written and tested against a newly-installed Ubuntu 18.04 server. If you have got other versions of some software already installed (perhaps you upgraded from an earlier Ubuntu version, or you set up some PPAs to load from) then you may need to make some adjustments.

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.

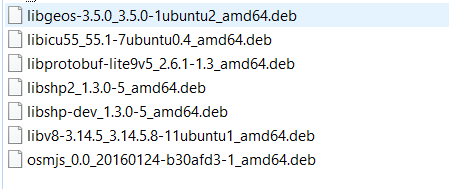
In order to build these components, a variety of dependencies need to be installed first:

sudo apt install libboost-all-dev git-core tar unzip wget bzip2 build-essential autoconf libtool libxml2-dev libgeos-dev libgeos++-dev libpq-dev libbz2-dev libproj-dev munin-node munin protobuf-c-compiler libfreetype6-dev libtiff5-dev libicu-dev libgdal-dev libcairo-dev libcairomm-1.0-dev apache2 apache2-dev libagg-dev liblua5.2-dev ttf-unifont lua5.1 liblua5.1-dev libgeotiff-epsg curl osmosis

Say yes to install. This will take a while, so go and have a cup of tea. This list includes various utilities and libraries, the Apache web server, and “carto” which is used to convert Carto-CSS stylesheets into something that “mapnik” the map renderer can understand.

Also you need to download certain packages which are not available in default repositories but required. Following is the list of packages. Their version may differ as per your OS version and other depending libraries: You can install in the sequence they are listed in the image below, using

**sudo dpkg -I <package name>**



When that is complete, install the second set of prerequisites.

## **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. The below steps will make sure you get the latest version of postgres.

sudo apt-get install wget ca-certificates

wget --quiet -O - https://www.postgresql.org/media/keys/ACCC4CF8.asc | sudo apt-key add -

sudo sh -c 'echo "deb http://apt.postgresql.org/pub/repos/apt/ `lsb\_release -cs`-pgdg main" >> /etc/apt/sources.list.d/pgdg.list'

sudo apt-get install postgresql postgresql-contrib postgis

Here “postgresql” is the database we’re going to store map data and “postgis” adds some extra graphical support to it. Again, say yes to install.

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 *osm* wherever it is used below. This should be the username that will render maps with Mapnik.

sudo -u postgres -i

createuser osm # answer yes for superuser (although this isn't strictly necessary)

createdb -E UTF8 -O osm gis

While still working as the “postgres” user, set up PostGIS on the PostgreSQL database (again, substitute your username for *osm* below):

psql

(that’ll put you at a “postgres=#” prompt)

\c gis

\password osm

Enter password and make it same as password for osm OS user which you will be creating soon in next steps.

(it’ll answer ‘You are now connected to database “gis” as user “postgres”.’)

CREATE EXTENSION postgis;

(it’ll answer CREATE EXTENSION)

CREATE EXTENSION hstore;

(it’ll answer CREATE EXTENSION)

ALTER TABLE geometry\_columns OWNER TO osm;

(it’ll answer ALTER TABLE)

ALTER TABLE spatial\_ref\_sys OWNER TO osm;

(it’ll answer ALTER TABLE)

\q

(it’ll exit psql and go back to a normal Linux prompt)

exit

(to exit back to be the user that we were before we did “sudo -u postgres -i” above)

If you haven’t already created one create a Unix user for this user, too, choosing a password when prompted:

sudo useradd -m osm

sudo passwd osm (Keep the password same as DB OSM user password)

## **Installing osm2pgsql**

We will need install various bits of software from source. The first of this is “osm2pgsql”. Various tools to import and manage OpenStreetMap data into a database exist. Here we’ll use “osm2pgsql”, which is probably the most popular.

su - osm

mkdir ~/src

cd ~/src

git clone git://github.com/openstreetmap/osm2pgsql.git

cd osm2pgsql

The build mechanism used by osm2pgsql has changed since older versions, so we’ll need to install some more prerequisites for that:

sudo apt install make cmake g++ libboost-dev libboost-system-dev libboost-filesystem-dev libexpat1-dev zlib1g-dev libbz2-dev libpq-dev libgeos-dev libgeos++-dev libproj-dev lua5.2 liblua5.2-dev

Again, say yes to install.

mkdir build && cd build

cmake ..

(the output from that should end with “build files have been written to…)

make

(the output from that should finish with “[100%] Built target osm2pgsql”)

sudo make install

## **Mapnik**

Next, we’ll install Mapnik. We’ll use the default version in Ubuntu 18.04:

sudo apt-get install autoconf apache2-dev libtool libxml2-dev libbz2-dev libgeos-dev libgeos++-dev libproj-dev gdal-bin libmapnik-dev mapnik-utils python-mapnik ttf-dejavu

We’ll check 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! You can leave Python with this command:

>>> quit()

## **Install mod\_tile and renderd**

Next, we’ll install mod\_tile and renderd. “mod\_tile” is an Apache module that handles requests for tiles; “renderd” is a daemon that actually renders tiles when “mod\_tile” requests them. We’ll use the “switch2osm” branch of mod\_tile: <https://github.com/SomeoneElseOSM/mod_tile>, which is itself forked from <https://github.com/openstreetmap/mod_tile>, but modified so that it supports Ubuntu 16.04, and with a couple of other changes to work on a standard Ubuntu server rather than one of OSM’s rendering servers.Compile the mod\_tile source code:

cd ~/src

git clone -b switch2osm git://github.com/SomeoneElseOSM/mod\_tile.git

cd mod\_tile

./autogen.sh

(that should finish with “autoreconf: Leaving directory `.'”)

./configure

(that should finish with “config.status: executing libtool commands”)

make

Note that some “worrying” messages will scroll up the screen here. However it should finish with “make[1]: Leaving directory ‘/home/osm/src/mod\_tile'”.

sudo make install

(that should finish with “make[1]: Leaving directory ‘/home/osm/src/mod\_tile'”)

sudo make install-mod\_tile

(that should finish with “chmod 644 /usr/lib/apache2/modules/mod\_tile.so”)

sudo ldconfig

(that shouldn’t reply with anything)

## **Stylesheet configuration**

Now that all of the necessary software is installed, you will need to download and configure a stylesheet. The style we’ll use here is the one that use by the “standard” map on the openstreetmap.org website. It’s chosen because it’s well documented and should work anywhere in the world (including in places with non-latin place names). There are a couple of downsides though – it’s very much a compromise designed to work globally, and it’s quite complicated to understand and modify, should you need to do that.

The home of “OpenStreetMap Carto” on the web is

<https://github.com/gravitystorm/openstreetmap-carto/>

and it has its own installation instructions at

<https://github.com/gravitystorm/openstreetmap-carto/blob/master/INSTALL.md>

although we’ll cover everything that needs to be done here. Here we’re assuming that we’re storing the stylesheet details in a directory below “src” below the home directory of the “osm” user (or whichever other one you are using)

cd ~/src

git clone git://github.com/gravitystorm/openstreetmap-carto.git

cd openstreetmap-carto

Next, we’ll install a suitable version of the “carto” compiler. This is later than the version that ships with Ubuntu, so we need to do:

sudo apt install npm nodejs

sudo npm install -g carto

carto -v

That should respond with a number that is at least as high as:

carto 1.1.0

Now that all of the necessary software is installed, you will need to download and configure a stylesheet. For these instructions we will use [OSM Bright](https://github.com/mapbox/osm-bright), a good all-purpose stylesheet with a style reminiscent of popular web maps. (If you’d like a map that looks exactly the same as the one on openstreetmap.org, you’ll need to use the [openstreetmap-carto](https://github.com/gravitystorm/openstreetmap-carto) project which has [its own installation instructions](https://github.com/gravitystorm/openstreetmap-carto/blob/master/INSTALL.md).)

Together with most modern stylesheets, OSM Bright is written in a stylesheet language called CartoCSS which is reminiscent of the CSS used for web design. Its main advantage is easy reading/writing, leading to rapid prototyping (using utilities such as [Tilemill](https://www.mapbox.com/tilemill/), [Kosmtik](https://github.com/kosmtik/kosmtik) or [Sputnik](https://github.com/systemed/sputnik)). However, you do typically need to adapt third-party CartoCSS stylesheets for your own server setup, and then compile them to the XML required by Mapnik; that’s what we’ll do here.

We will use /usr/local/share/maps/style as a common directory for our stylesheet files and resources.

## **Download OSM Bright**

To begin with, we need to download both the OSM Bright stylesheet, and also the additional data resources it uses (for coastlines and the like).  Again, substitute your username for osm in the “chown” command below.

sudo mkdir -p /usr/local/share/maps/style

sudo chown osm /usr/local/share/maps/style

cd /usr/local/share/maps/style

wget <https://github.com/mapbox/osm-bright/archive/master.zip>

wget <https://osmdata.openstreetmap.de/download/simplified-land-polygons-complete-3857.zip>

wget <https://osmdata.openstreetmap.de/download/land-polygons-split-3857.zip>

mkdir ne\_10m\_populated\_places\_simple

cd ne\_10m\_populated\_places\_simple

wget

<http://www.naturalearthdata.com/http//www.naturalearthdata.com/download/10m/cultural/ne_10m_populated_places_simple.zip> (this URL is correct URL although it doesn’t look like. Please use it as is)

unzip ne\_10m\_populated\_places\_simple.zip

rm ne\_10m\_populated\_places\_simple.zip

cd ..

We then move the downloaded data into the osm-bright-master project directory:

unzip '\*.zip'

mkdir osm-bright-master/shp

mv land-polygons-split-3857 osm-bright-master/shp/

mv simplified-land-polygons-complete-3857 osm-bright-master/shp/

mv ne\_10m\_populated\_places\_simple osm-bright-master/shp/

To improve performance, we create index files for the larger shapefiles:

cd osm-bright-master/shp/land-polygons-split-3857

shapeindex land\_polygons.shp

cd ../simplified-land-polygons-complete-3857/

shapeindex simplified\_land\_polygons.shp

cd ../..

## **Configuring OSM Bright**

The OSM Bright stylesheet now needs to be adjusted to include the location of our data files. Edit the file osm-bright/osm-bright.osm2pgsql.mml in your favorite text editor, for example:

vi osm-bright/osm-bright.osm2pgsql.mml

Find the lines with URLs pointing to shapefiles (ending .zip) and replace each one with these appropriate pairs of lines:

"file": "/usr/local/share/maps/style/osm-bright-master/shp/land-polygons-split-3857/land\_polygons.shp",

"type": "shape"

"file": "/usr/local/share/maps/style/osm-bright-master/shp/simplified-land-polygons-complete-3857/simplified\_land\_polygons.shp",

"type": "shape",

"file": "/usr/local/share/maps/style/osm-bright-master/shp/ne\_10m\_populated\_places\_simple/ne\_10m\_populated\_places\_simple.shp",

"type": "shape"

Note that we are also adding “type”: “shape” to each one. Finally, in the section dealing with “ne\_places”, replace the “srs” and “srs-name” lines with this one line:

"srs": "+proj=longlat +ellps=WGS84 +datum=WGS84 +no\_defs"

## **Compiling the stylesheet**

We now have a fully working CartoCSS stylesheet. Before Mapnik can use it, we need to compile it into XML using the command-line carto compiler. First of all, we use OSM Bright’s own preprocessor, which we need to edit for our setup:

cp configure.py.sample configure.py

vi configure.py

Change the config line pointing to ***~/Documents/Mapbox/project*** to ***/usr/local/share/maps/style*** instead and change dbname from osm to ***gis***. Save and exit.

Run the pre-processor and then carto:

./make.py

cd ../OSMBright/

carto project.mml > OSMBright.xml

You now have a Mapnik XML stylesheet at ***/usr/local/share/maps/style/OSMBright/OSMBright.xml*** .

## **Setting up your webserver**

Next we need to plug renderd and mod\_tile into the Apache webserver, ready to receive tile requests.

### Configure renderd

Change the the renderd settings by editing the /usr/local/etc/renderd.conf (you’ll need to do it as root via “sudo”) and change the following five lines, uncommenting (removing the ‘;’) when required. They are found in the [renderd], [mapnik] and [default] sections.

socketname=/var/run/renderd/renderd.sock

plugins\_dir=/usr/local/lib/mapnik/input

font\_dir=/usr/share/fonts/truetype/ttf-dejavu

XML=/usr/local/share/maps/style/OSMBright/OSMBright.xml

HOST=localhost

Create the files required for the mod\_tile system to run (remember to change username to your user’s name ‘osm’):

sudo mkdir /var/run/renderd

sudo chown osm /var/run/renderd

sudo mkdir /var/lib/mod\_tile

sudo chown osm /var/lib/mod\_tile

### Configure mod\_tile

Next, we need to tell the Apache web server about our new mod\_tile installation.

Using your favourite text editor, create the file /etc/apache2/conf-available/mod\_tile.conf and add one line:

sudo vi /etc/apache2/conf-available/mod\_tile.conf

Copy the following line in the newly created mod\_tile.conf fie and save it.

LoadModule tile\_module /usr/lib/apache2/modules/mod\_tile.so

Apache’s default website configuration file needs to be modified to include mod\_tile settings. Modify the file /etc/apache2/sites-available/000-default.conf to include the following lines immediately after the admin e-mail address line:

sudo vi /etc/apache2/sites-available/000-default.conf

add the following lines:

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

Tell Apache that you have added the new module, and restart it:

sudo a2enconf mod\_tile

And reload apache twice:

**sudo service apache2 reload**

**sudo service apache2 reload**

## **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 9.3 needs to be tuned for the amount of data you are about to add to it. Edit the file /etc/postgresql/9.3/main/postgresql.conf and make the following changes:

shared\_buffers = 128MB

checkpoint\_segments = 20

maintenance\_work\_mem = 256MB

These changes require a kernel configuration change, which needs to be applied every time that the computer is rebooted. As root via “sudo”, 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**

Initially, we’ll load only a small amount of test data. Other download locations are available, but “[download.geofabrik.de](http://download.geofabrik.de/)” has a wide range of options. In this example we’ll download the data for India, which is about 700Mb.

Browse to <http://download.geofabrik.de/asia/india.html>  and note the “This file was last modified” date (e.g. “2017-02-26T21:43:02Z”). We’ll need that later if we want to update the database with people’s subsequent changes to OpenStreetMap. Download it as follows:

mkdir /opt/data

cd /opt/data

wget http://download.geofabrik.de/asia/india-latest.osm.pbf

The following command will insert the OpenStreetMap data you downloaded earlier into the database. This step is very disk I/O intensive; importing the full planet might take many hours, days or weeks depending on the hardware. 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 -d gis --create --slim -G --hstore --tag-transform-script ~/src/openstreetmap-carto/openstreetmap-carto.lua -C 2500 --number-processes 1 -S ~/src/openstreetmap-carto/openstreetmap-carto.style /opt/data/india-latest.osm.pbf

It’s worth explaining a little bit about what those options mean:

-d gis

The database to work with (“gis” used to be the default; now it must be specified).

--create

Load data into an empty database rather than trying to append to an existing one.

--slim

osm2pgsql can use different table layouts; “slim” tables works for rendering.

-G

Determines how multipolygons are processed.

--hstore

Allows tags for which there are no explicit database columns to be used for rendering.

--tag-transform-script

Defines the lua script used for tag processing. This an easy is a way to process OSM tags before the style itself processes them, making the style logic potentially much simpler.

-C 2500

Allocate 2.5 Gb of memory to osm2pgsql to the import process. If you have less memory you could try a smaller number, and if the import process is killed because it runs out of memory you’ll need to try a smaller number or a smaller OSM extract.

--number-processes 1

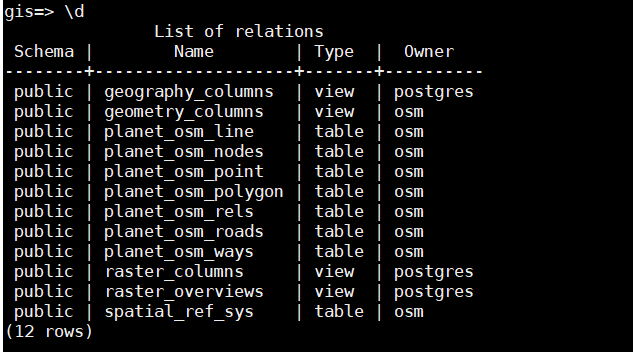
Use 1 CPU. If you have more cores available, you can use more.

-S

Create the database columns in this file (actually these are unchanged from “openstreetmap-carto”)

The final argument is the data file to load.

Following tables will be created after above process is complete successfully



DB Tweaks before running rendering process:

By default the z\_order and population column are not added in planet\_osm\_point table. So you need to add that otherwise you will get error in rendering logs.

As osm user:

#psql -d gis

gis=> ALTER TABLE planet\_osm\_point ADD COLUMN z\_order integer;

gis=> ALTER TABLE planet\_osm\_point ADD COLUMN population text;

## **Running renderd for the first time**

Next, we’ll run renderd to try and render some tiles. Initially we’ll run it in the foreground so that we can see any errors as they occur:

renderd -f -c /usr/local/etc/renderd.conf

You may see some warnings here – don’t worry about those for now. You shouldn’t get any errors. If you do, save the full output in a paste bin and ask a question about the problem somewhere like help.openstreetmap.org (linking to the paste bin – don’t include all the text in the question).

Point a web browser at: http://<your web server>/hot/0/0/0.png



You should see a map of whole world map without boundaries and India with India boundaries in your browser and some more debug on the command line, including “DEBUG: START TILE” and “DEBUG: DONE TILE”. Ignore any “DEBUG: Failed to read cmd on fd” message – it is not an error. If you don’t get a tile and get other errors again save the full output in a paste bin and ask a question about the problem somewhere like help.openstreetmap.org.

If that all works, press control-c to stop the foreground rendering process.

## **Running renderd in the background**

Next we’ll set up “renderd” to run in the background. First, edit the “~/src/mod\_tile/debian/renderd.init” file so that “RUNASUSER” is set to the non-root account that you have used before, such as “osm”, then copy it to the system directory:

vi ~/src/mod\_tile/debian/renderd.init

sudo cp ~/src/mod\_tile/debian/renderd.init /etc/init.d/renderd

sudo chmod u+x /etc/init.d/renderd

sudo cp ~/src/mod\_tile/debian/renderd.service /lib/systemd/system/

The “renderd.service” file is a “systemd” service file. The version used here just calls old-style init commands. In order to test that the start command works:

sudo /etc/init.d/renderd start

(that should reply with “[ ok ] Starting renderd (via systemctl): renderd.service.”)

To make it start automatically every time:

sudo systemctl enable renderd

## **Viewing tiles**

In order to see tiles, we’ll cheat and use an html file “sample\_leaflet.html” in mod\_tile’s “extras” folder. Just open that file in a web browser on the machine where you installed the tile server. If that isn’t possible because you’re installing on a server without a local web browser, you can edit it to replace “127.0.0.1” with the IP address of the server and copy it to below “/var/www/html”.

From an ssh connection do:

tail -f /var/log/syslog | grep " TILE "

(note the spaces around “TILE” there)

That will show a line every time a tile is requested, and one every time rendering of one is completed.

When you load that page you should see some tile requests. Zoom out gradually. You’ll see requests for new tiles show up in the ssh connection. Some low-zoom tiles may take a long time (several minutes) to render for the first time, but once done they’ll be ready for the next time that they are needed.

Congratulations. Head over to the [using tiles](https://switch2osm.org/using-tiles/) section to create a map that uses your new tile server. We are using Leaflet for our maps.

## **Leaflet**

To display your slippy map with Leaftlet, download JavaScript and CSS from [leaftletjs.com](http://leafletjs.com/download.html) and extract it to the web root folder.

cd /var/www/html/

sudo wget http://cdn.leafletjs.com/leaflet/v1.2.0/leaflet.zip

sudo unzip leaflet.zip

Next, create the **index.html** file.

sudo vi /var/www/html/index.html

Paste the following HTML code in the file. Replace red-colored text and adjust the longitude, latitude and zoom level in green color according to your needs.

*<html>*

*<head>*

*<title>My Open Street Maps</title>*

*<link rel="stylesheet" type="text/css" href="leaflet.css"/>*

*<script type="text/javascript" src="leaflet.js"></script>*

*<style>*

*#map{width:100%;height:100%}*

*</style>*

*</head>*

*<body>*

*<div id="map"></div>*

*<script>*

*var map = L.map('map').setView([23.259933,77.412613],4.5);*

*L.tileLayer('http://<your web server>/hot/{z}/{x}/{y}.png',{maxZoom:18}).addTo(map);*

*</script>*

*</body>*

*</html>*

Save and close the file. Restart apache2 service. Now you can view your slippy map by typing your server IP address in browser.

http://<your web server>/



The above map shows the India only map without India’s complete administrative

boundaries also called as disputed boundaries.

**Correcting Indian Map**

## **Disputed border between India, Pakistan, and India, China**

In some cases, local administrative departments require certain disputed map regions

displayed in a particular manner. The OSM database includes information about disputed

borders, and extra properties on boundaries which have tags like ''claimed\_by''. One

strategy to get around displaying disputed borders as the local administrative body would

like it is:

Do not generate any lines on tiles for disputed borders. Overlay the ''claimed\_by'' border of

the administrative agency in question over the existing imported OSM data while rendering

tiles. There may be many methods to get around this problem. If you get stuck, seek help in the OpenStreetMap community. Usually, their

<http://webchat.oftc.net/?randomnick=1&channels=osm> | RC channel or

<https://lists.openstreetmap.org/listinfo/dev> | mailing list may be good starting places.

This section requires some knowledge of editing OSM xml style files. For a quick visual

reference. See the difference between the <http://openstreetmap.in/#4/25.36/80.95> | Indian

and <https://www.openstreetmap.org/#map=4/26.27/87.10> | global map versions of this

region. The disputed areas are the state of Jammu and Kashmir, and Arunachal Pradesh.

In this example, we will make the following tweaks to the standard OSM setup:

Install the <https://github.com/mapbox/osm-boundaries> | osm-boundaries package.

Run the ''run.py'' script on the ''india-latest.osm.pbf'' file.

This will generate a file called ''osm\_admin\_x-y.osm.pbf'', where ''x'' and ''y'' were

''admin\_level'' arguments given to the script.

## **Install osm-boundaries package**

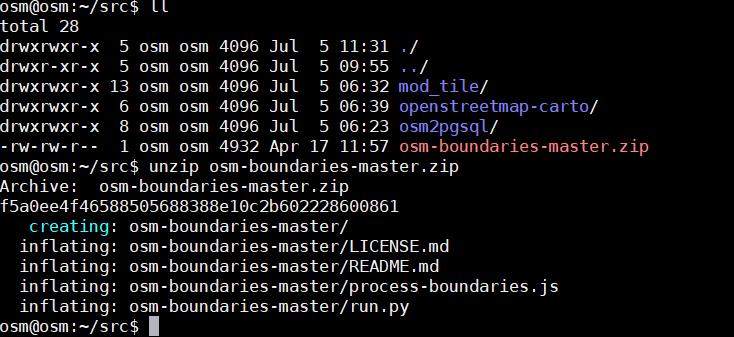
**login as osm user:**

**Step 1** - download the osm-boundaries packages from the

<https://github.com/mapbox/osm-boundaries>

**Step 2** - copy the osm-boundaries-master.zip to ~/src/ folder

**Step 3** - unzip osm-boundaries-master.zip



**Step 4** – sudo apt install python-pip

**Step 5** – sudo pip install psycopg2

**Step 6** – edit run.py to add password fields in sql commands, the edited file will look like this

Changes are highlighted in yellow color:

## Arguments & help

ap = argparse.ArgumentParser(description='Process OSM administrative ' +

'boundaries as individual ways.')

ap.add\_argument('-d', dest='db\_name', default='gis',

help='PostgreSQL database.')

ap.add\_argument('-U', dest='db\_user', default='osm',

help='PostgreSQL user name.')

ap.add\_argument('-H', dest='db\_host', default='localhost',

help='PostgreSQL host.')

ap.add\_argument('-p', dest='db\_port', default='5432',

help='PostgreSQL port.')

ap.add\_argument('-P', dest='db\_password', default='xxxxxxx',

help='PostgreSQL Password.')

ap.add\_argument('-f', dest='min\_admin\_level', type=int, default=2,

help='Minimum admin\_level to retrieve.')

ap.add\_argument('-t', dest='max\_admin\_level', type=int, default=4,

help='Maximum admin\_level to retrieve.')

ap.add\_argument(dest='osm\_input', metavar='/opt/data/planet-latest.osm.pbf',

help='An OpenStreetMap PBF file to process.')

args = ap.parse\_args()

## PostgreSQL setup

# Set up the db connection

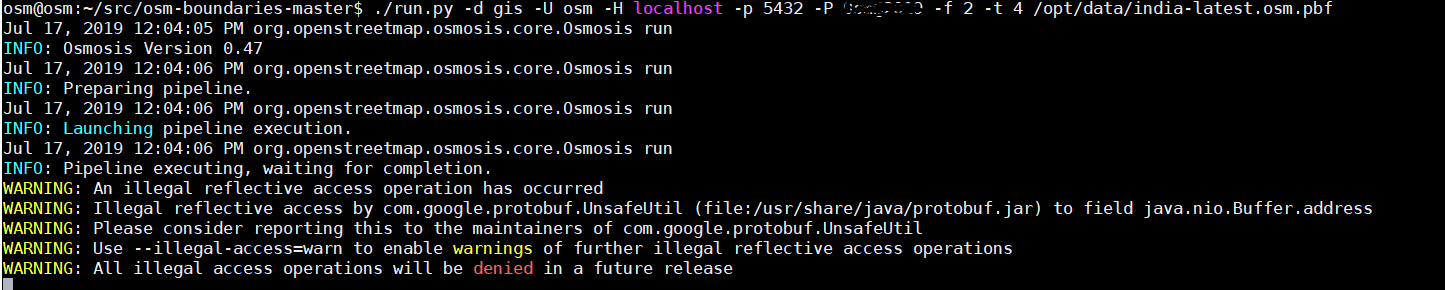
con = psycopg2.connect("dbname={0} user={1} host={2} port={3} password={4}".format(

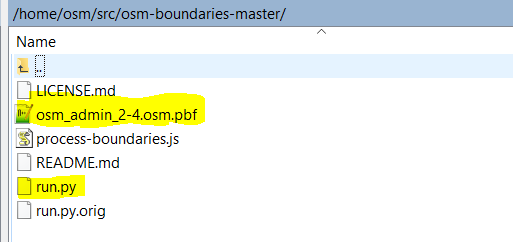
args.db\_name, args.db\_user, args.db\_host, args.db\_port, args.db\_password))

cur = con.cursor()

**Step 7**- run the run.py using the following command:

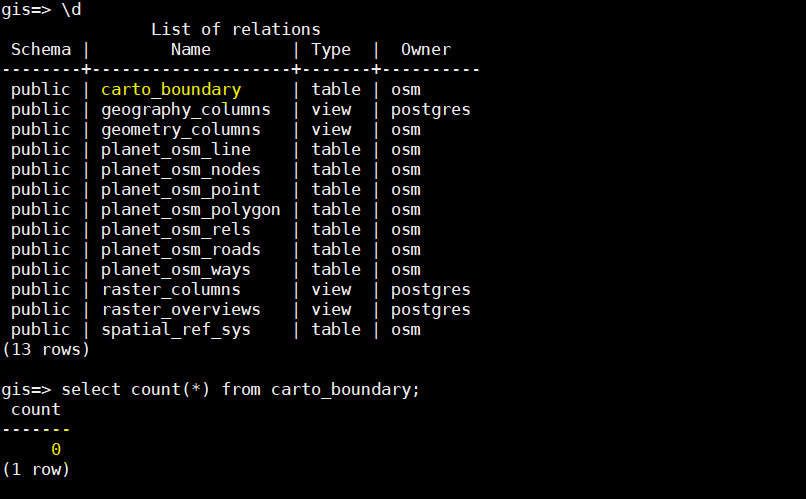
./run.py -d gis -U osm -H localhost -p 5432 -P xxxxxx -f 1 -t 4 /opt/data/india-latest.osm.pbf



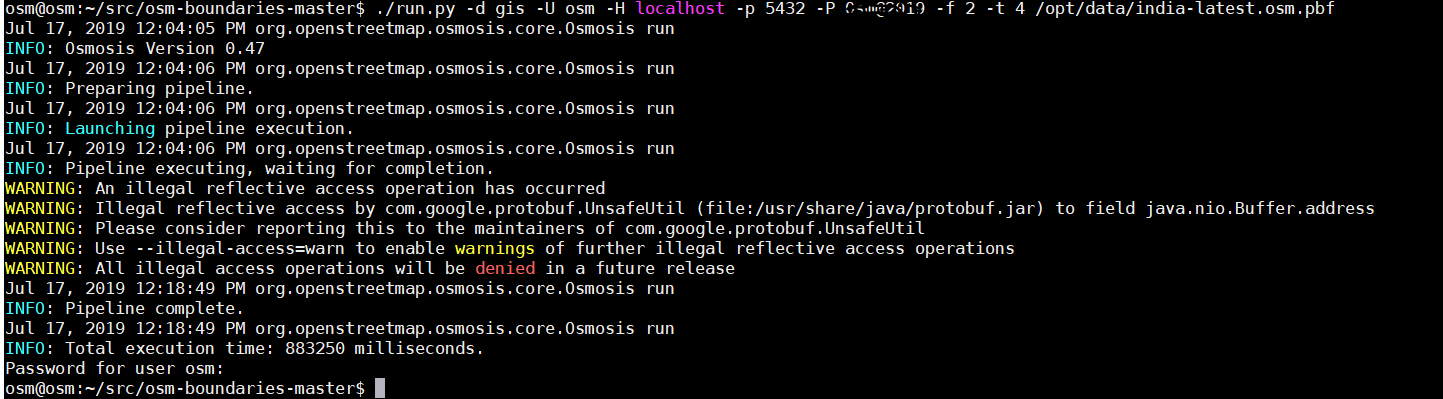


The above process can take hours to complete, so please be patient. You can see a new

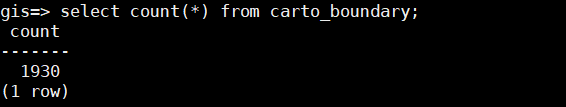
table carto\_boundary created inside gis DB with 0 rows.



Successful completion of run.py script



Make sure you have data populated inside this table, other you need to check what went wrong and run run.py once again.



## **Edit OSMBright.xml Style File**

Hide any borders which have the ''disputed'' tag set to yes. Overlay our custom ''claimed\_by (India)'' data on the map when rendering tiles. For the second step above, find the

OSMBright.xml style after cloning the repository and building the style file. Find the Layer

definition for ''admin''. These are the administrative boundaries we need to selectively hide. The existing xml would look something like this:

**/usr/local/share/maps/style/OSMBright/OSMBright.xml**

**Existing entry will look like the code below:**

<Layer name="admin" srs="+proj=merc +a=6378137 +b=6378137 +lat\_ts=0.0 +lon\_0=0.0 +x\_0=0.0 +y\_0=0.0 +k=1.0 +units=m +nadgrids=@null +wktext +no\_defs +over">

<StyleName><![CDATA[admin]]></StyleName>

<Datasource>

<Parameter name="dbname"><![CDATA[gis]]></Parameter>

<Parameter name="extent"><![CDATA[-20037508.34,-20037508.34,20037508.34,20037508.34]]></Parameter>

<Parameter name="geometry\_field"><![CDATA[way]]></Parameter>

<Parameter name="id"><![CDATA[admin]]></Parameter>

<Parameter name="key\_field"><![CDATA[]]></Parameter>

<Parameter name="project"><![CDATA[osm-bright-imposm]]></Parameter>

<Parameter name="table"><![CDATA[( SELECT way, admin\_level

FROM planet\_osm\_line

WHERE boundary = 'administrative'

AND admin\_level IN ('2','3','4')

) AS data]]></Parameter>

<Parameter name="type"><![CDATA[postgis]]></Parameter>

</Datasource>

</Layer>

**Edit it to look like this:**

<Layer name="admin" srs="+proj=merc +a=6378137 +b=6378137 +lat\_ts=0.0 +lon\_0=0.0 +x\_0=0.0 +y\_0=0.0 +k=1.0 +units=m +nadgrids=@null +wktext +no\_defs +over">

<StyleName>admin</StyleName>

<Datasource>

<Parameter name="dbname"><![CDATA[gis]]></Parameter>

<Parameter name="extent"><![CDATA[-20037508.34,-20037508.34,20037508.34,20037508.34]]></Parameter>

<Parameter name="geometry\_field"><![CDATA[geom]]></Parameter>

<Parameter name="id"><![CDATA[admin]]></Parameter>

<Parameter name="key\_field"><![CDATA[]]></Parameter>

<Parameter name="project"><![CDATA[osm-bright-imposm]]></Parameter>

<Parameter name="table"><![CDATA[( SELECT geom, admin\_level, disputed, maritime FROM carto\_boundary WHERE admin\_level IN ('1','2','3','4') AND disputed = 0 ) AS data]]></Parameter>

<Parameter name="type"><![CDATA[postgis]]></Parameter>

</Datasource>

</Layer>

Explanation of the changes: \* Instead of using ''planet\_osm\_line'', use the ''carto\_boundary'' table, which has the ''disputed'' information.

**Only apply the style and render the boundary if ''disputed = 0''.**

In the next step, we will overlay the ''claimed\_by'' information. For this, the

<https://github.com/datameet/maps/tree/master/Country> should be used to generate the data. For the borders claimed by India, download the file india-osm.geojson. Rename the file and remove hyphen from file name (**indiaosm.geojson**). Copy the file to

server at **/usr/local/share/maps/style/osm-bright-master/geojson/boundary\_claimed\_by\_india** folder. If this folder doesn’t exist, create one.

**Add the following definitions to the OSMBright.xml style file.**

<Layer name="india\_boundary" status="on" minimum-scale-denominator="7500" maximum-scale-denominator="5000000000" srs="+proj=longlat +ellps=WGS84 +datum=WGS84 +no\_defs">

<StyleName>india\_boundary</StyleName>

<Datasource>

<Parameter name="layer"><![CDATA[**indiaosm**]]></Parameter>

<Parameter name="type"><![CDATA[ogr]]></Parameter>

<Parameter name="file"><![CDATA[**/usr/local/share/maps/style/osm-bright-master/geojson/boundary\_claimed\_by\_india/indiaosm.geojson**]]></Parameter>

</Datasource>

</Layer>

Additionally, you would need to create the ''india\_boundary'' style, which should be a copy

of the ''admin'' style minus all entries using filters. In our case it looks like this: all filter

enteries are removed.

<Style name="india\_boundary" filter-mode="first" opacity="0.5">

<Rule>

<MaxScaleDenominator>3000000</MaxScaleDenominator>

<LineSymbolizer stroke-width="4" stroke-linecap="round" stroke-linejoin="round" stroke="#444466" />

</Rule>

<Rule>

<MaxScaleDenominator>12500000</MaxScaleDenominator>

<MinScaleDenominator>3000000</MinScaleDenominator>

<LineSymbolizer stroke-width="2" stroke-linecap="round" stroke-linejoin="round" stroke="#444466" />

</Rule>

<Rule>

<MaxScaleDenominator>50000000</MaxScaleDenominator>

<MinScaleDenominator>12500000</MinScaleDenominator>

<LineSymbolizer stroke-width="1.2" stroke-linecap="round" stroke-linejoin="round" stroke="#444466" />

</Rule>

<Rule>

<MinScaleDenominator>50000000</MinScaleDenominator>

<LineSymbolizer stroke-width="0.8" stroke-linecap="round" stroke-linejoin="round" stroke="#444466" />

</Rule>

<Rule>

<LineSymbolizer stroke-linejoin="round" stroke="#444466" />

</Rule>

</Style>

Once you have made these changes, restart the apache server and start the tile rendering

process.

You should now see the administrative boundaries as required. Please wait for some time

for tiles to get rendered.

## **Correct Map of India**

