GDAL 101 Script

Installation

Install QGIS, if necessary

Verification:

`orginfo`

Download the data

Check out the data in QGIS

OGR for Vector Data

`ogrinfo pools/pools.shp`

> Break this down: GDAL uses what it calls "drivers" to read & write spatial data. In this case, it automatically detected this was a shapefile and used the "ESRI Shapefile" driver. We'll talk more about drivers in a few minutes. This shapefile has 1 layer (of course, because it's a shapefile) called "pools". Pools is a polygon layer.
>
> Do the same command for the "zipcodes" shapefile.

`ogrinfo zipcodes/zipcodes.shp`

Now try:

`ogrinfo -so zipcodes/zipcodes.shp zipcodes`

Break this down:

> "-so" IMPORTANT, this means "summary only". If you don't use this, then ogr will happily print the attribute values for every record in your spatial layer. (thus, zipcodes vs pools: fewer records in case of mistype :))
>
> so on a layer shows field names & types; also spatial reference system.

GDAL for Raster Data

`gdalinfo dem10m/dem10m.dem`

> the "gdalinfo" command doesn't have a -so option, as a raster doesn't have multiple records that could be summarized.
>
> Notice GDAL auto-detected that this is a USGS DEM and used the appropriate driver.
>
> Here, we see the CRS, along with useful metadata like bounding box coordinates, statistics, and more.

> Use the same gdalinfo command on the DOQQ (what driver does GDAL use there?)

Converting Data

Exporting original data into new formats

Vector I/O Formats

`ogr2ogr --formats`

> Some formats are read-only, some also have write.
>
> Generally, only a fraction of the more common drivers are included in a GDAL installation. The version of GDAL you installed may have different drivers built-in than the version of someone else: if you want a specific driver, you may need to compile your own GDAL installation.

Raster I/O Formats

`gdal_translate --formats`

shapefile → geojson

let's make a webmap-friendly version of our zipcodes shapefile:

`ogr2ogr -f geojson zipcodes.geojson zipcodes/zipcodes.shp`

NOTE: This will generate Warnings - that's ok!

> use "-f" to specify the output driver we want ogr to use: here, you put the output file name first, then the source.
>
> yes, really. it's that fast. open your geojson file in a text editor if you don't believe me.
>
> WARNINGs are because shapefiles don't really support date fields, and coerce into YYYYMMDD strings.

Now try: `ogrinfo zipcodes.geojson`

and: `ogrinfo -so zipcodes.geojson zipcodes`

> notice that OGR gives geojson layer's a default name of "OGRGeoJSON" unless using GDAL 2.2+ - verify with ogrinfo

GeoTIFF → georeferenced PNG

`gdal_translate -of png doqq.tif converted_doqq.png`

> note that gdalwarp uses "-of" for output format. And for some reason, source & destination are reversed from ogr2ogr.

Now try:

`gdalinfo converted_doqq.png`

> here you can verify that your PNG is still spatially aware!

Reprojecting Data

Side note: EPSG Codes spatialreference.org (great resource!)

> Is everyone here familiar with EPSG codes as an easy, standard way of referring to a spatial reference system? Just checking… When reprojecting data with GDAL/OGR, EPSG codes are one of the ways you can specify a target spatial reference (you could also use things like a prj file or a PROJ4 definition – but today we're just going to use EPSG codes for convenience)

Let's make everything WGS84!

```
ogr2ogr -t_srs EPSG:4326 reprojected_pools.shp pools/pools.shp
```

```
gdalwarp -t_srs EPSG:4326 doqq.tif reprojected_doqq.tif
```

> REMEMBER: ogr and gdal put the source & target filenames in reverse order, for some reason…

Let's take a break!

Querying Data

> Asking questions about your data layers OGR allows you to use a dialect of SQL ("OGR SQL") to query layers

Q: "How many swimming pools are in Austin?"

```
ogrinfo pools/pools.shp -sql "SELECT COUNT(*) FROM pools"
```

A: 51,073

Q: "What is the first pool feature?"

```
ogrinfo -q pools/pools.shp -sql "SELECT * FROM pools WHERE fid = 1"
```

> Here I've started using an option "-q" flag, before the input file name, to tell ogrinfo to be a little quieter. You can try this command with and without "-q" to see the difference – basically "quiet" is just supressing some metadata from being displayed.

Q: "How many above ground pools in Austin?"

```
ogrinfo -q pools/pools.shp -sql "SELECT COUNT(*) FROM pools WHERE FEATURE = 'Above ground'"
```

A: 21,550

Saving query results to a new file

Similar to how you might save a "query view" in other GIS software, you can use OGR to create a new file based on a SQL query

"Create a shapefile containing only the zipcode boundaries in Austin city limits"

```
ogr2ogr austin_zips.shp zipcodes/zipcodes.shp -sql "SELECT * FROM zipcodes WHERE name = 'Austin'"
```

> Here, you put the target filename first, then the source, followed by the SQL query. You can ignore any warnings that talk about SHAPEAREA field width – this just means the exact calculated area for SHAPEAREA won't fit in the default precision of this shapefile field, so it's being rounded slightly.

> There is a way to fix (set OGR_TRUNCATE=YES in GDAL>=1.11) this BUT it is GDAL version-dependent so we are going to just ignore the warning for today.

"Create a shapefile containing small zipcodes"

```
ogr2ogr under_10mil.shp zipcodes/zipcodes.shp -sql "SELECT * FROM zipcodes WHERE SHAPE_AREA <10000000"
```

> Here we're just using the calculated area field, in map units (so, decimal degrees squared in WGS84?). We wouldn't really want that in real life, but we're not going to bother adding a calculated area value here for now.

"Create a geoJSON with only zipcode 78756's boundary"

```
ogr2ogr -f geojson 78756.geojson zipcodes/zipcodes.shp -sql "SELECT * FROM zipcodes WHERE ZIPCODE = '78756'"
```

> Here you'll see combining a query with format conversion is basically no extra work at all. Again, ignore the WARNINGs about Date field here

Clipping to create new data

You can clip a target data layer to a given clipping layer to create a new layer

"Create a new layer with only the pools in zipcode 78756"

```
ogr2ogr -clipsrc 78756.geojson 78756_pools.shp reprojected_pools.shp
```

> We're using a new flag here, "-clipsrc", to indicate what we're going to clip our input layer by. In order to clip layers, they need to be in the same projection – so use our WGS84 version of "pools", since zipcodes is already in WGS84.
>
> You don't have to create a shapefile here; you can use any output format you like. Notice how mixing & matching file types is also not a problem.

"Show only the imagery in zipcode 78756"

```
gdalwarp -cutline 78756.geojson doqq.tif 78756_doqq.tif
```

> We use gdalwarp to modify the raster based on an input raster. Here, the clipping tool flag is "-cutline". Remember, on the gdal side of the house, "input" and "destination" ordering is reversed!

"Clip a DEM to a bounding box"

```
gdal_translate -srcwin 0 0 1000 1000 -of USGSDEM dem10m/dem10m.dem clipped_dem.dem
```

> This time, we're ising gdal_translate and specifying a bounding box ("srcwin"). Here, we're starting at x offset=0, y offset=0, and making a box 1000 map units square.