# Using GDAL / OGR for Data Processing and Analysis

Roger Andre
May 2009

# What the heck is "GDAL"?

- GDAL is an open source programming library and set of utilities that allow you to manipulate raster data.
- As a library, GDAL presents a single abstract data model to the calling application for all supported formats.
- OGR is the vector analog, also comes with GDAL
- Main focus is on conversions between formats, but that's just the tip of the iceberg.
- Works in Windows, Linux and MacOS

# GDAL Trivia

- History: Development started by Frank Warmerdam in 1998. Now officially maintained by OSGEO.
- How do you pronounce “GDAL”?

"I pronounce it "goodle".  I had originally thought to call it the "Geospatial Object Oriented Data Abstraction Library" (GOODAL) to make the right sound obvious, but I was too lazy to type GOODAL all the time, so I dropped the OO part.  Most folks pronounce the library "gee-dal" which is ok too but not my preference." - Frank Wamerdam

# gdal_translate -of

VRT (rw+): Virtual Raster
GTiff (rw+): GeoTIFF
NITF (rw+): National Imagery Transmission Format
HFA (rw+): Erdas Imagine Images (.img)
ELAS (rw+): ELAS
AAIGrid (rw): Arc/Info ASCII Grid
DTED (rw): DTED Elevation Raster
PNG (rw): Portable Network Graphics
JPEG (rw): JPEG JFIF
MEM (rw+): In Memory Raster
GIF (rw): Graphics Interchange Format (.gif)
XPM (rw): X11 PixMap Format
BMP (rw+): MS Windows Device Independent Bitmap
PCIDSK (rw+): PCIDSK Database File
PCRaster (rw): PCRaster Raster File
ILWIS (rw+): ILWIS Raster Map
SRTMHGT (rw): SRTMHGT File Format
Leveller (rw+): Leveller heightfield
Terragen (rw+): Terragen heightfield
GMT (rw): GMT NetCDF Grid Format
netCDF (rw): Network Common Data Format
PNM (rw+): Portable Pixmap Format (netpbm)
ENVI (rw+): ENVI .hdr Labelled
EHdr (rw+): ESRI .hdr Labelled
PAux (rw+): PCI .aux Labelled
MFF (rw+): Vexcel MFF Raster
MFF2 (rw+): Vexcel MFF2 (HKV) Raster
BT (rw+): VTP .bt (Binary Terrain) 1.3 Format
IDA (rw+): Image Data and Analysis
ERS (rw+): ERMapper .ers Labelled
JPEG2000 (rw): JPEG-2000 part 1 (ISO/IEC 15444-1)
FIT (rw): FIT Image
RMF (rw+): Raster Matrix Format
RST (rw+): Idrisi Raster A.1
INGR (rw+): Intergraph Raster
GSAG (rw): Golden Software ASCII Grid (.grd)
GSBG (rw+): Golden Software Binary Grid (.grd)
USGSDEM (rw): USGS Optional ASCII DEM (and CDED)
ADRG (rw+): ARC Digitized Raster Graphics

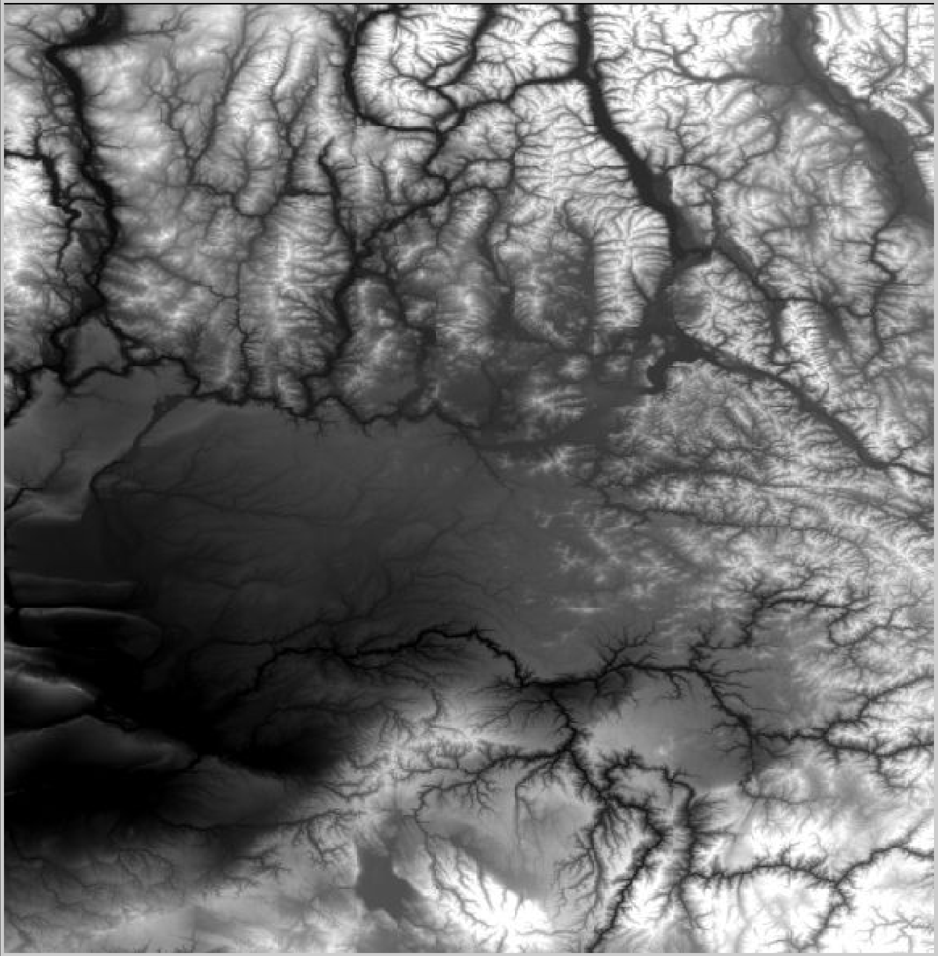
```
$ gdal_translate -of <output_format> <infile> <outfile>
```

# ogr2ogr -f

-> "ESRI Shapefile" (read/write)
-> "MapInfo File" (read/write)
-> "TIGER" (read/write)
-> "S57" (read/write)
-> "DGN" (read/write)
-> "Memory" (read/write)
-> "BNA" (read/write)
-> "CSV" (read/write)
-> "GML" (read/write)
-> "GPX" (read/write)
-> "KML" (read/write)
-> "GeoJSON" (read/write)
-> "Interlis 1" (read/write)
-> "Interlis 2" (read/write)
-> "GMT" (read/write)
-> "SQLite" (read/write)
-> "ODBC" (read/write)
-> "PostgreSQL" (read/write)

**$ ogr2ogr -f <output_format> <output_file> <options> <input_file>**

# gdalinfo



What is it?

$ gdalinfo srtm_13_03.TIF

```
Driver: GTiff/GeoTIFF
Files: srtm_13_03.TIF
Size is 6000, 6000     <== size
Coordinate System is:
GEOGCS["WGS 84", <== projection
    DATUM["WGS_1984",
        SPHEROID["WGS 84",6378137,298.257223563,
            AUTHORITY["EPSG","7030"]],
        AUTHORITY["EPSG","6326"]],
    PRIMEM["Greenwich",0],
    UNIT["degree",0.0174532925199433],
    AUTHORITY["EPSG","4326"]] <== EPSG code
Origin = (-120.000416666676756,49.999583672324661)
Pixel Size = (0.000833333333333,-0.000833333333333) <
Metadata:
  AREA_OR_POINT=Area <== metadata
Image Structure Metadata:
  INTERLEAVE=BAND
Corner Coordinates:          <== corner coords
Upper Left  (-120.0004167,  49.9995837)
Lower Left  (-120.0004167,  44.9995837)
Upper Right (-115.0004167,  49.9995837)
Lower Right (-115.0004167,  44.9995837)
Center      (-117.5004167,  47.4995837)
Band 1 Block=6000x1 Type=Int16, ColorInterp=Gray<=type
NoData Value=-32768        <== nodata value
```

# ogrinfo

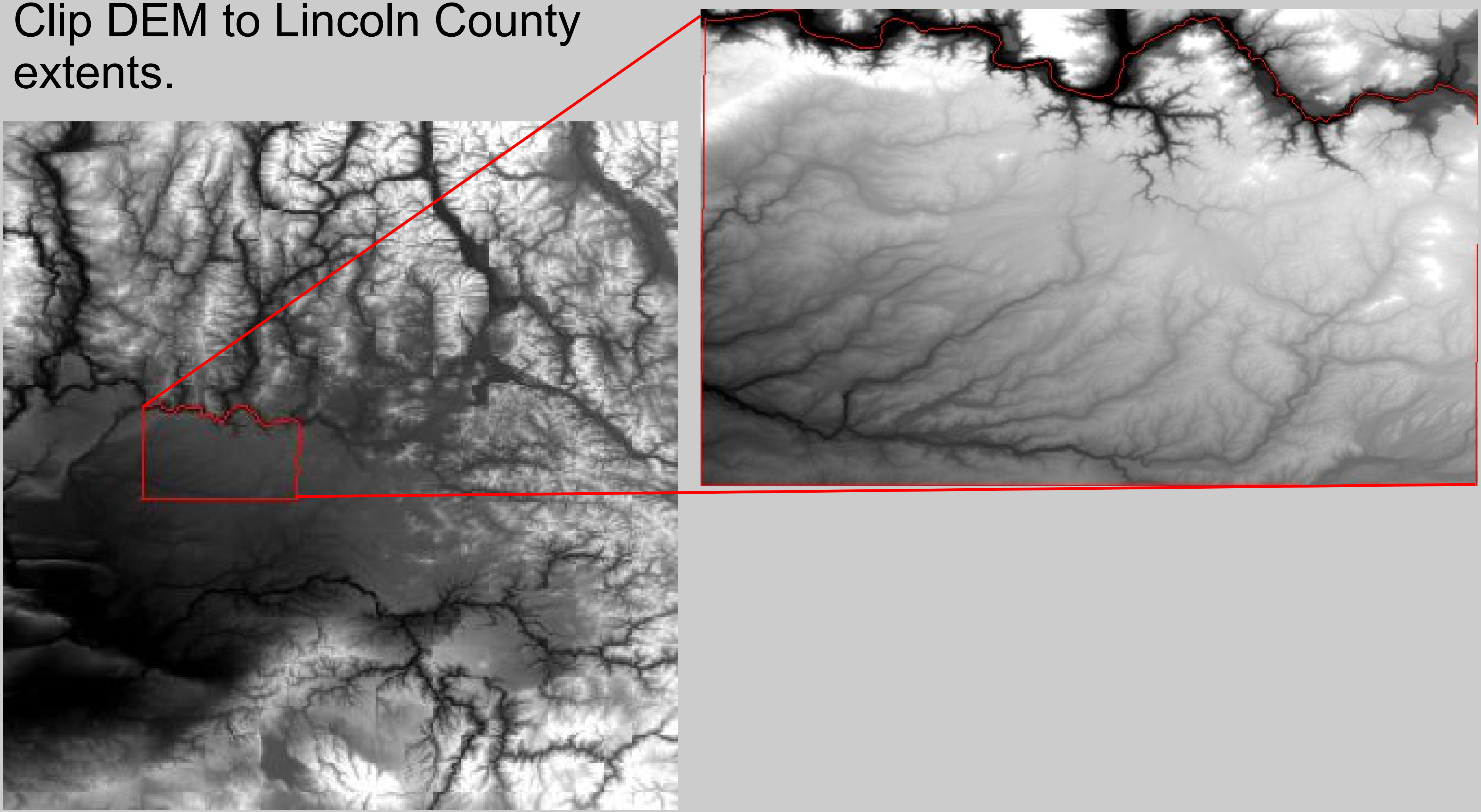

```
Layer name: wgs84_lincoln   <== usually filename
Geometry: Polygon           <== feature type
Feature Count: 1            <== how many features in layer
Extent: (-118.979777, 47.260011) - (-117.819452, 47.957913)
Layer SRS WKT:
GEOGCS["GCS_WGS_1984",<== projection info
    DATUM["WGS_1984",
        SPHEROID["WGS_1984",6378137,298.257223563]],
    PRIMEM["Greenwich",0],
    UNIT["Degree",0.017453292519943295]]
COUNTY_COD: Integer (2.0)  <== attribute field names/types
COUNTY_FIP: String (3.0)
COUNTY_NM: String (15.0)
ECY_REGION: String (4.0)
AIR_REGION: String (46.0)
```

What is it?

```
$ ogrinfo -summary ./ wgs84_lincoln
```

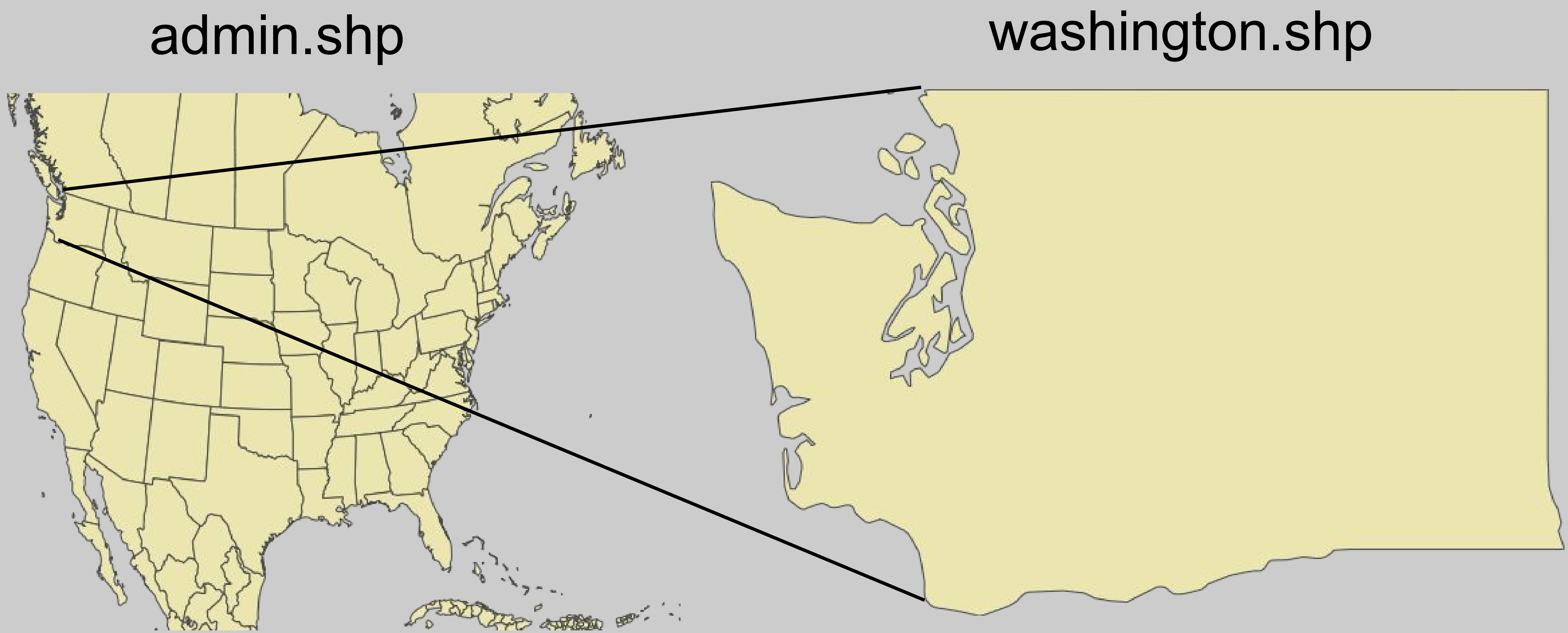
# gdal_translate -projwin

Clip DEM to Lincoln County extents.



**$ gdal_translate -projwin -118.979777 47.957913 -117.819452 47.260011 srtm_13_03.TIF lincoln.tif**

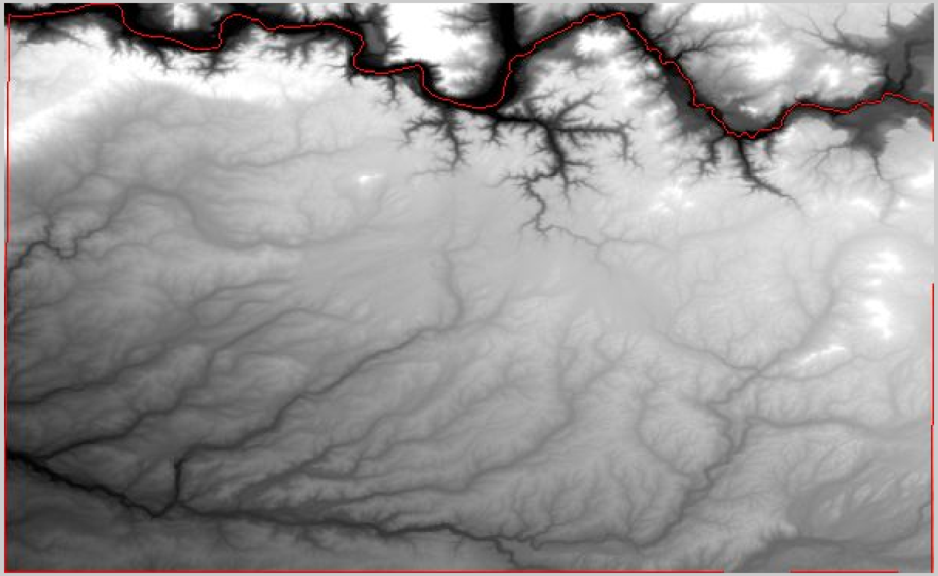
# ogr2ogr -where



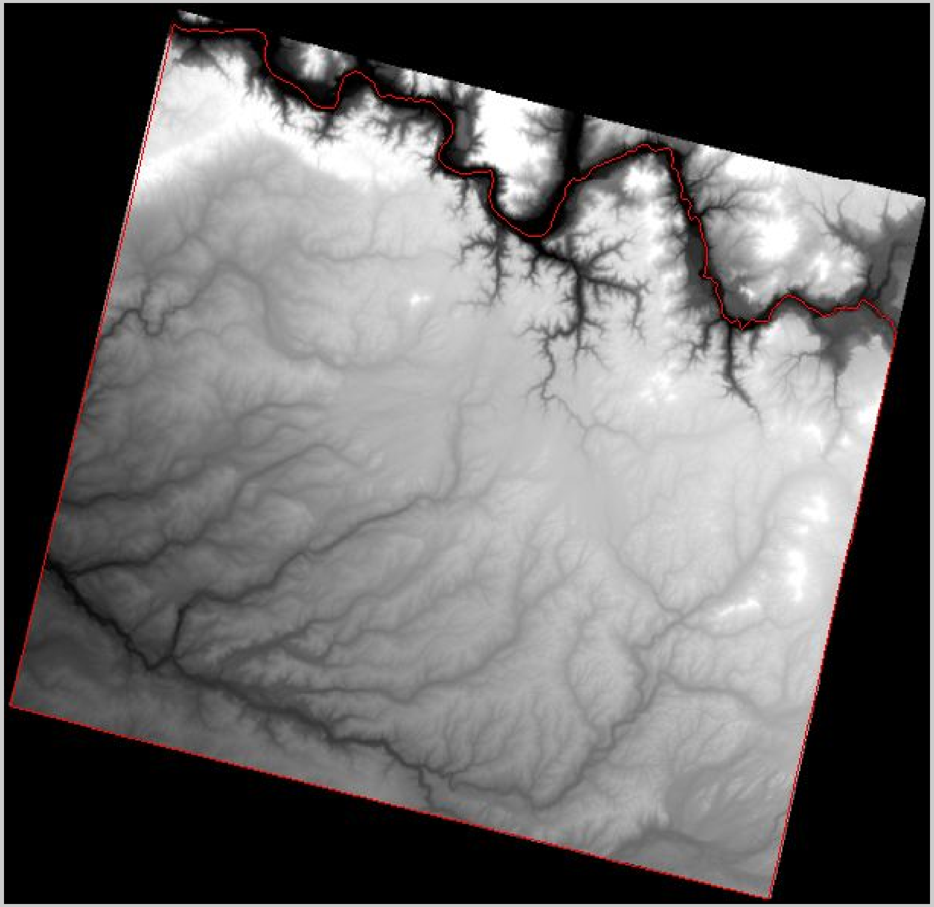

$ ogr2ogr washington.shp -where "ADMIN_NAME = 'Washington'" ./ admin

# gdalwarp

GCS  WGS84

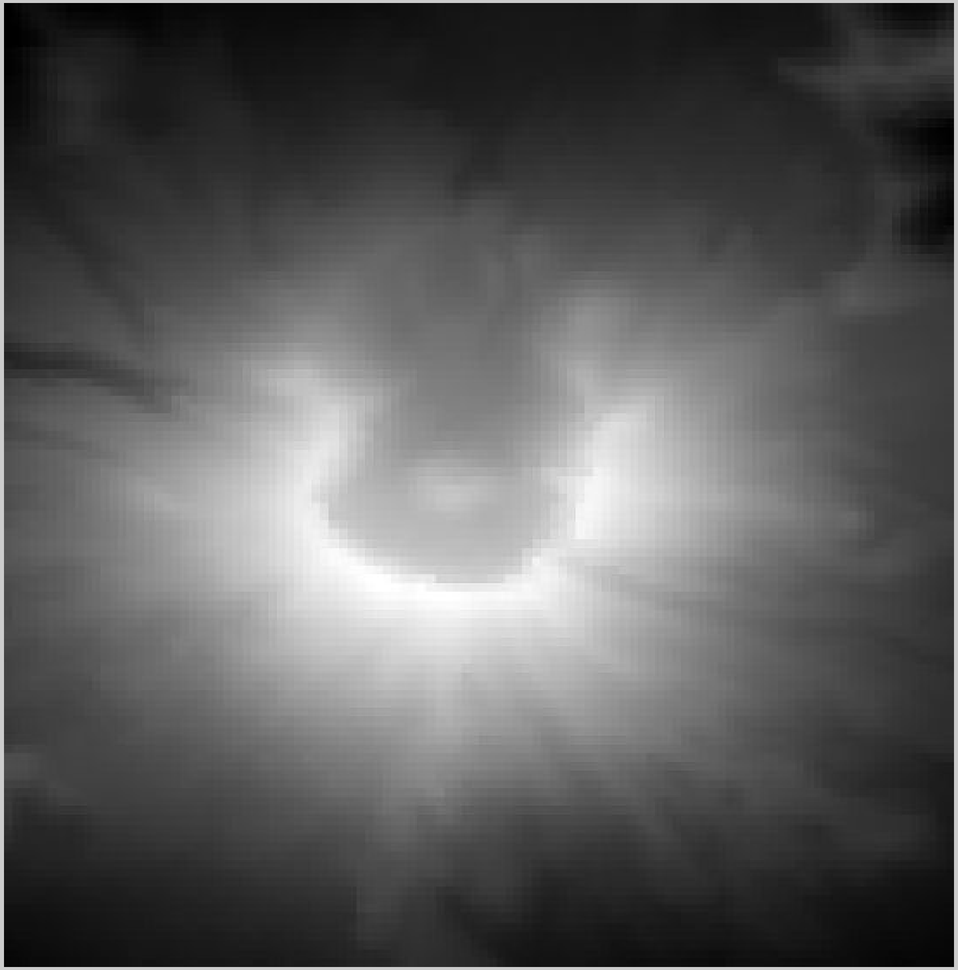


Albers Equal Area

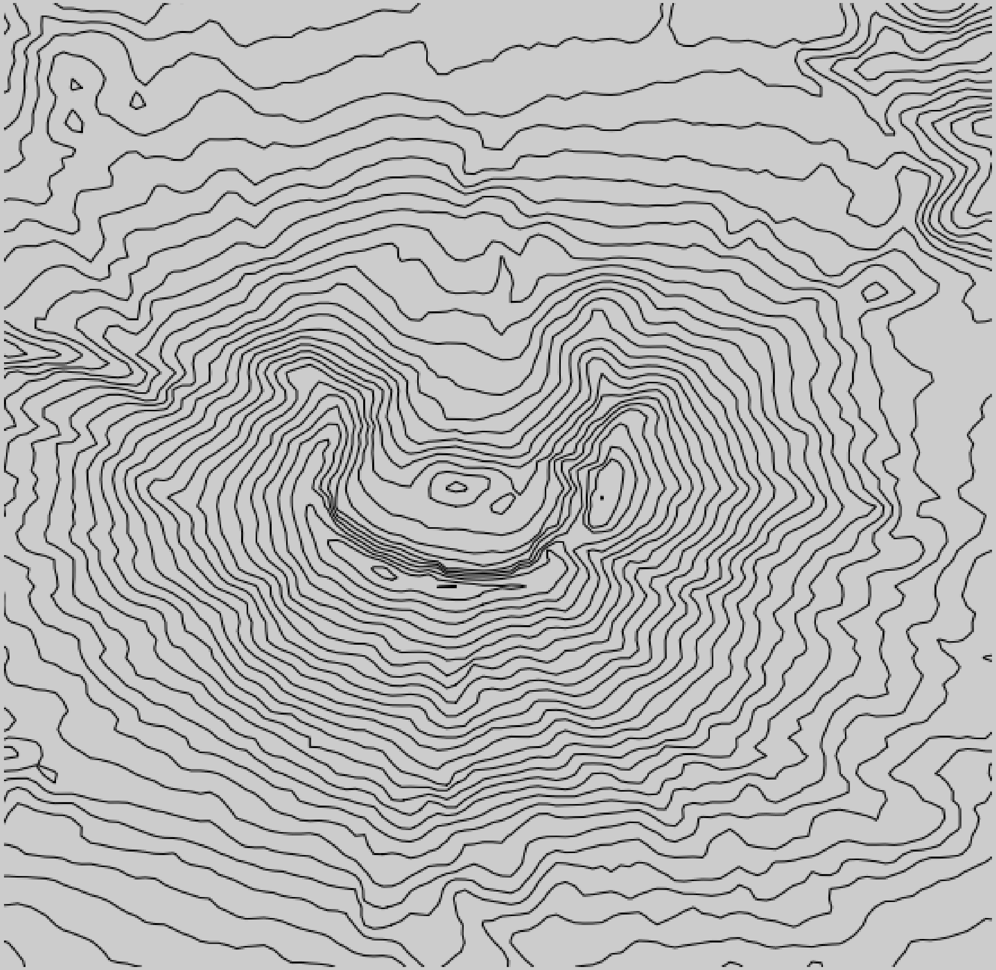


**$ gdalwarp -t_srs "EPSG:102003" lincoln.tif aea_lincoln.tif**

# gdal_contour



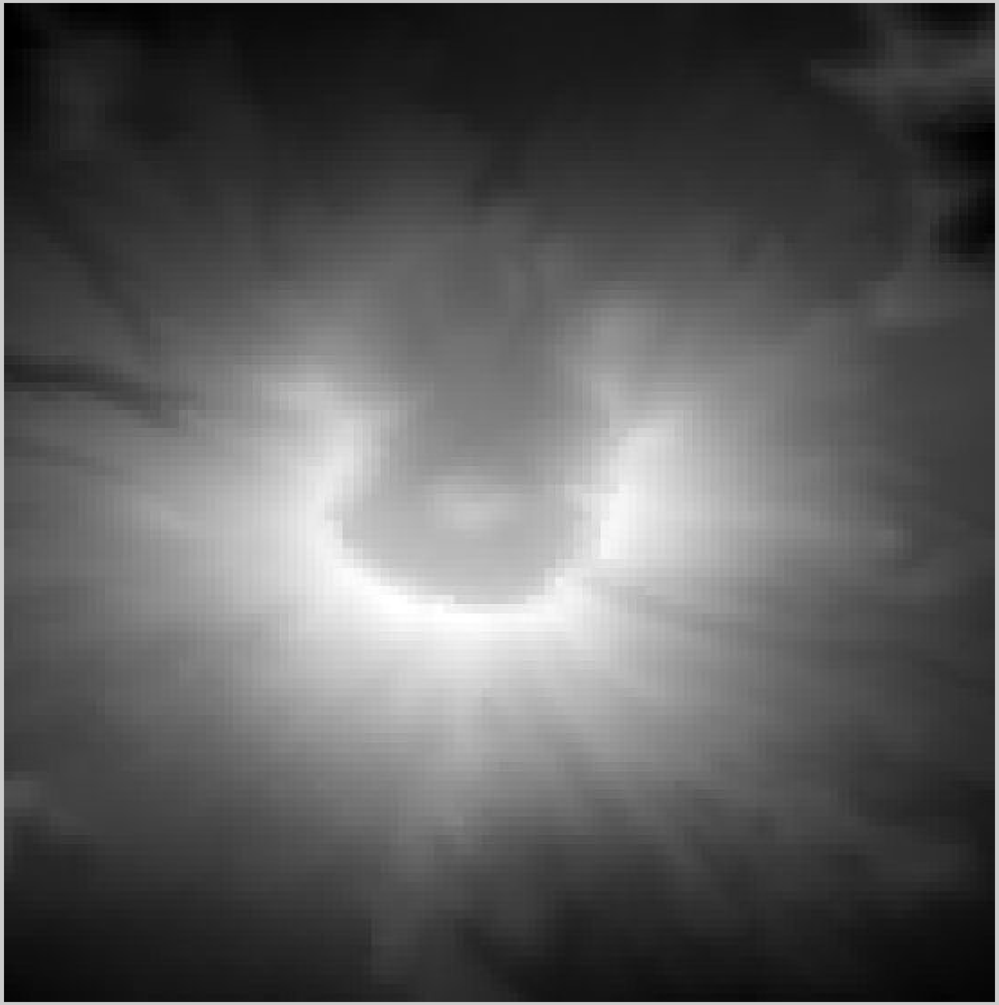
st_helens_dem.tif



st_helens_contours.shp

$ gdal_contour -a elev -i 50 st_helens_dem.tif  st_helens_contours.shp

# gdalwarp resampling

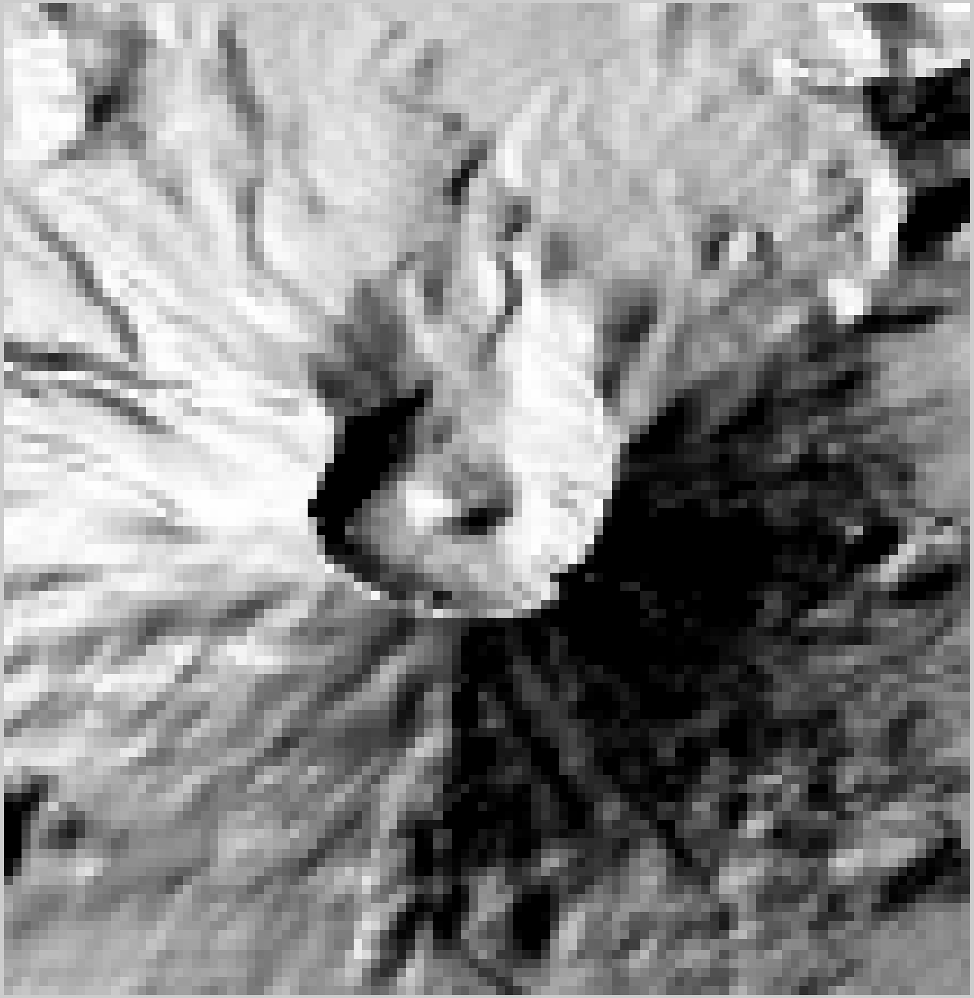


**st_helens_dem.tif**

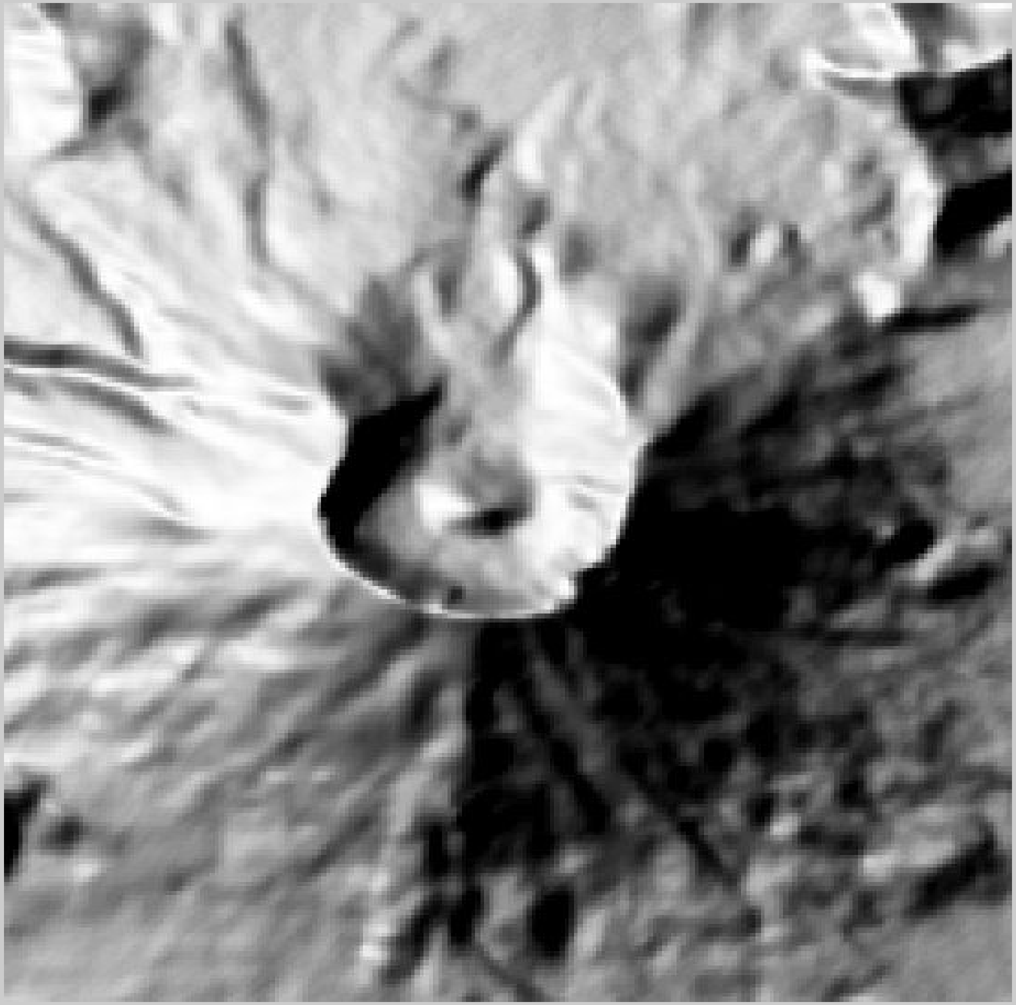
**2x_st_helens_dem.tif**

$ gdalwarp -ts 3672 2520 -r cubicspline st_helens_dem.tif 2x_st_helens_dem.tif

# Hillshades

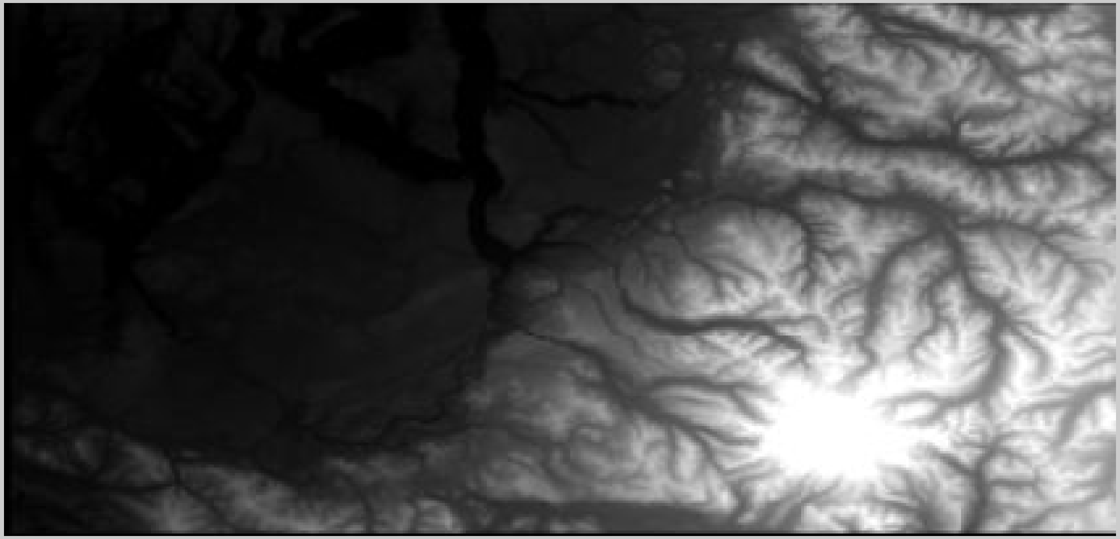


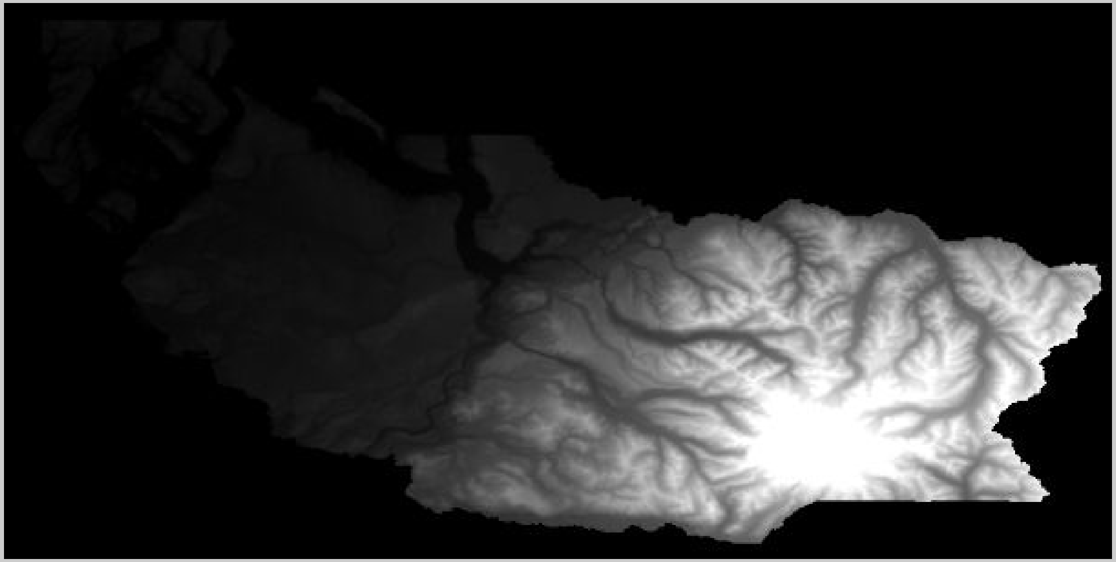
Hillshade from original DEM



Hillshade from resampled DEM

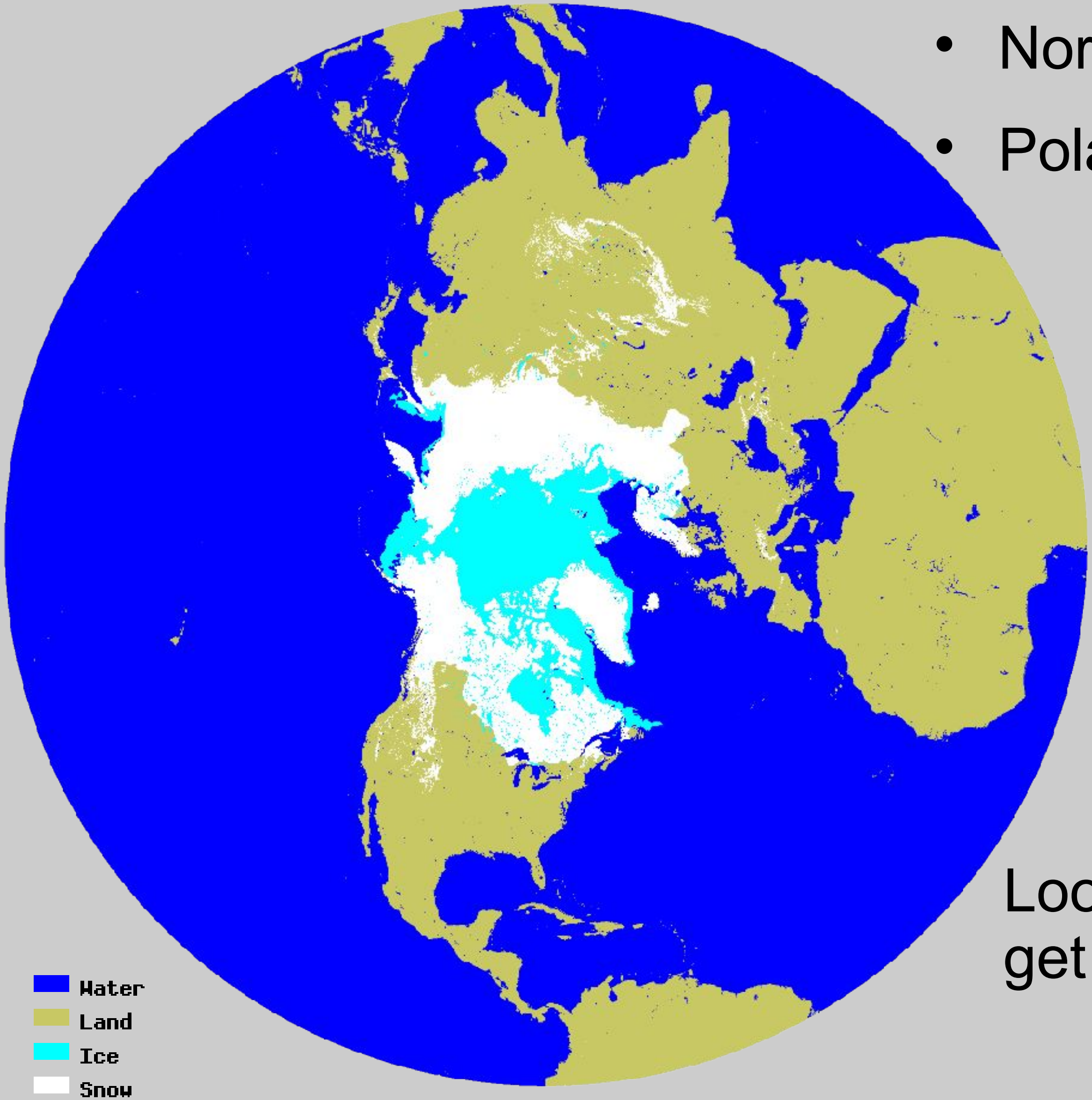
# gdal_rasterize





```
$ gdal_rasterize -b 1  -i  -burn -32678  -l  wgs84_pierce_county \
    wgs84_pierce_county.shp  masked_pierce_dem.tif
```

# Tying it All Together

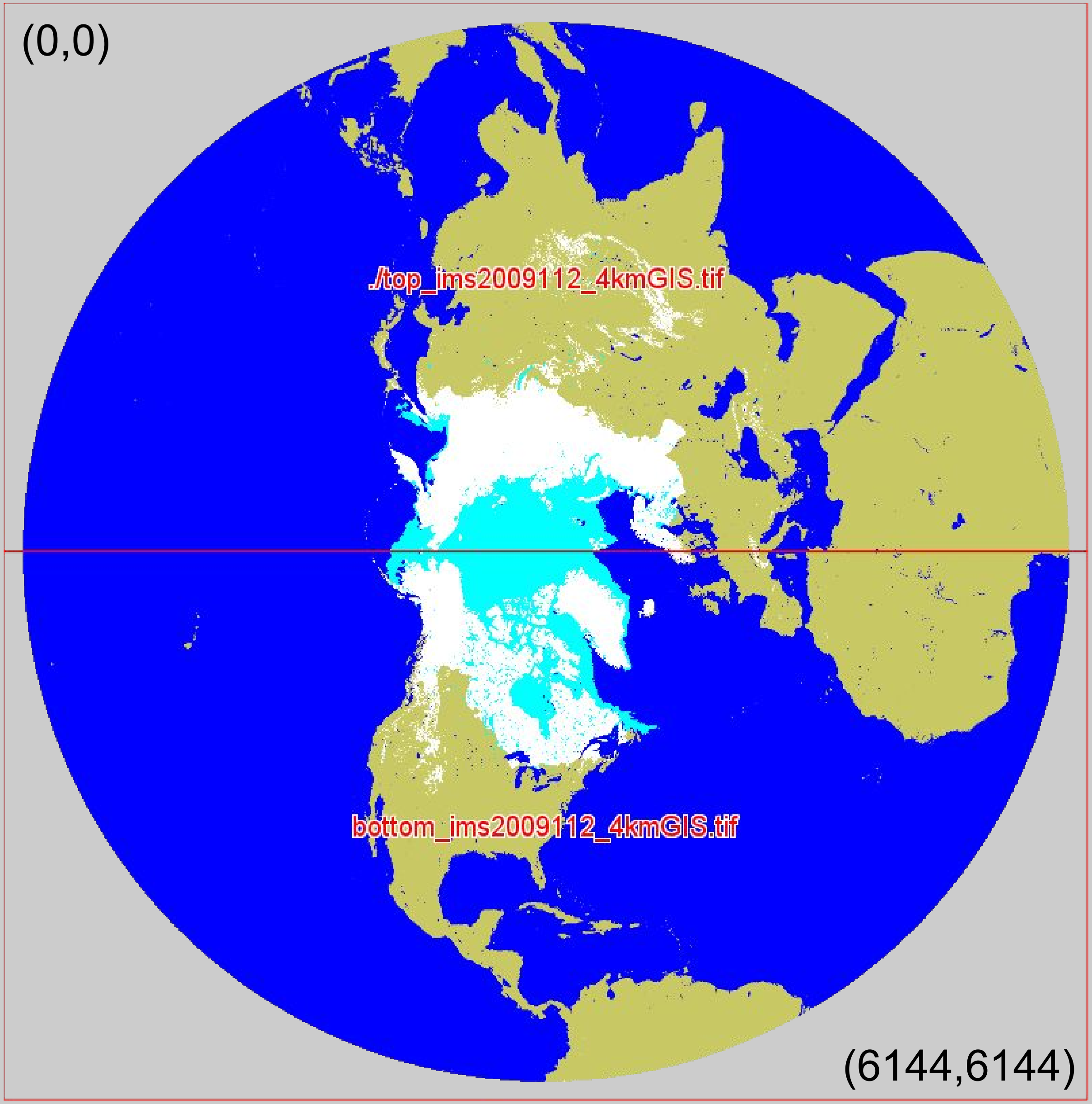



- Northern Hemisphere snow / ice
- Polar Stereographic projection

Looks pretty, but how do we get it to work with other data?

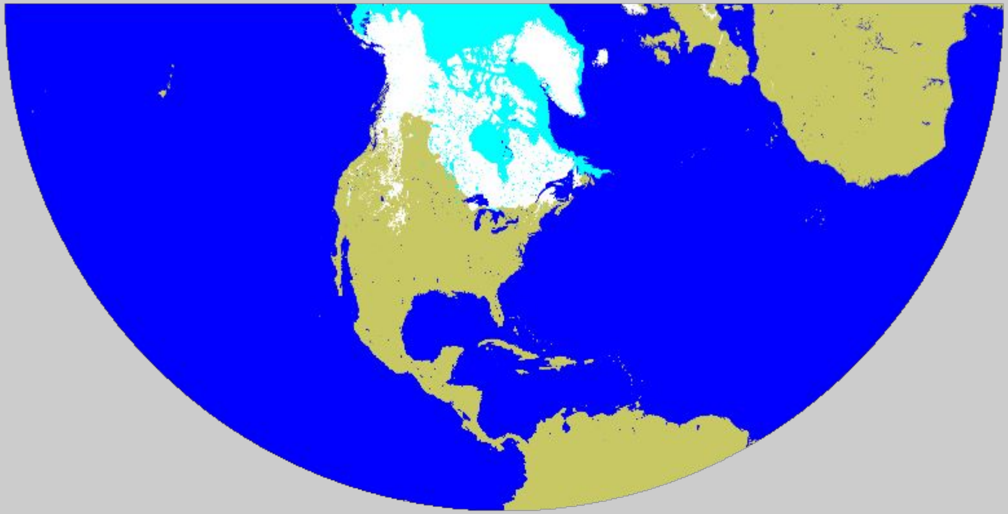
GDAL, of course!

# Step 1 - Clip



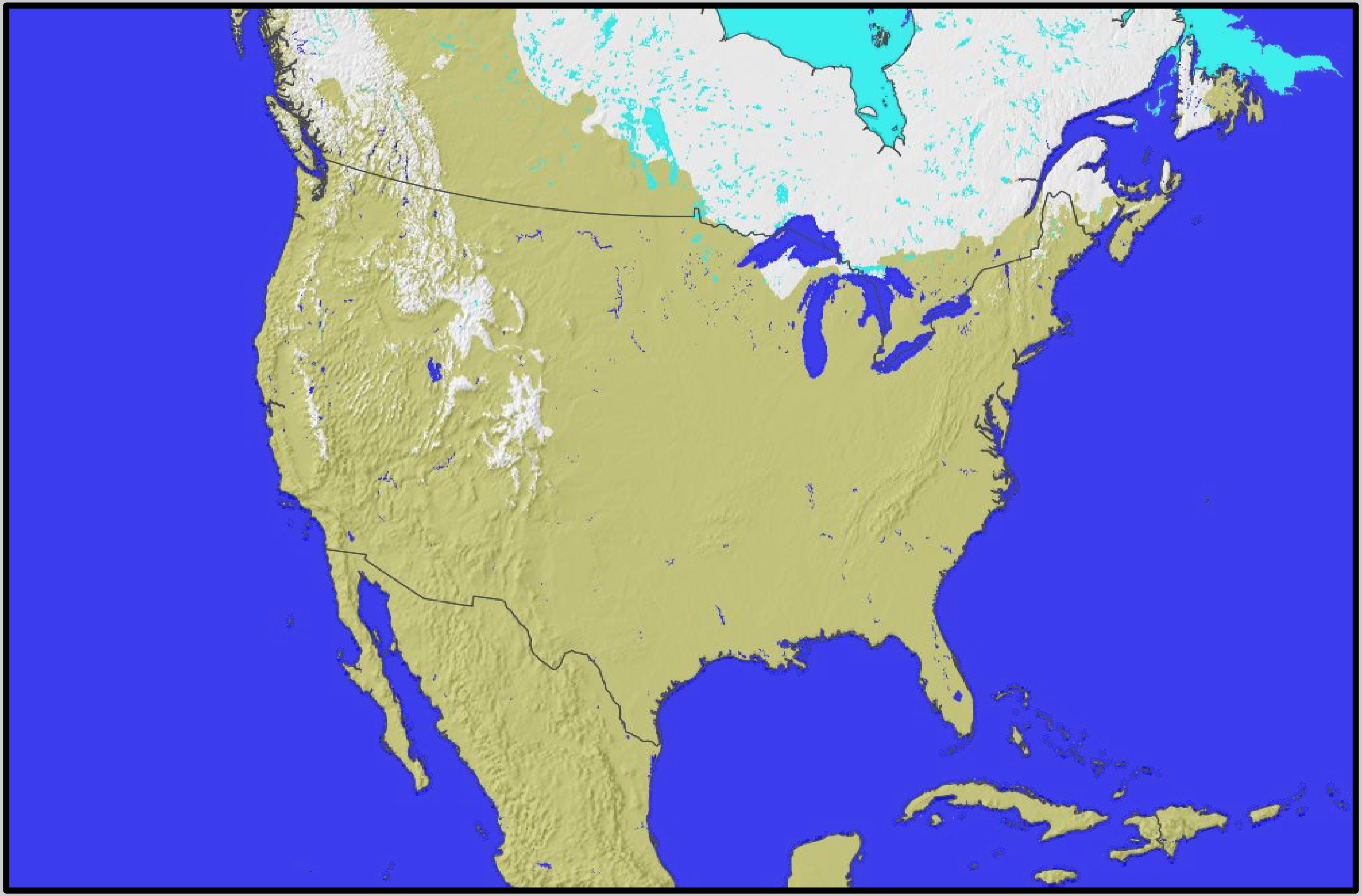

- gdal_translate -srcwin \
  0 0 6144 3073 ...
- gdal_translate -srcwin \
  0 3072 6144 3072 ...

# Step 2 - Reproject



- gdalwarp -t_srs “EPSG:4326”...

# Step 3 - Integrate



# Python API - rasters

```
#! /usr/bin/python

from osgeo import gdal
import sys
import numpy

src_file = sys.argv[1]
dst_file = sys.argv[2]
out_bands = 3

# Open source file
src_ds = gdal.Open( src_file )
src_band = src_ds.GetRasterBand(1)

# create destination file
## driver.Create( outfile, outwidth, outheight, numbands, gdaldatatype)
dst_driver = gdal.GetDriverByName('GTiff')
dst_ds = dst_driver.Create(dst_file, src_ds.RasterXSize,
src_ds.RasterYSize, out_bands, gdal.GDT_Byte)

# create output bands
band1 = numpy.zeros([src_ds.RasterYSize, src_ds.RasterXSize])
band2 = numpy.zeros([src_ds.RasterYSize, src_ds.RasterXSize])
band3 = numpy.zeros([src_ds.RasterYSize, src_ds.RasterXSize])
```

# Python - rasters (cont)

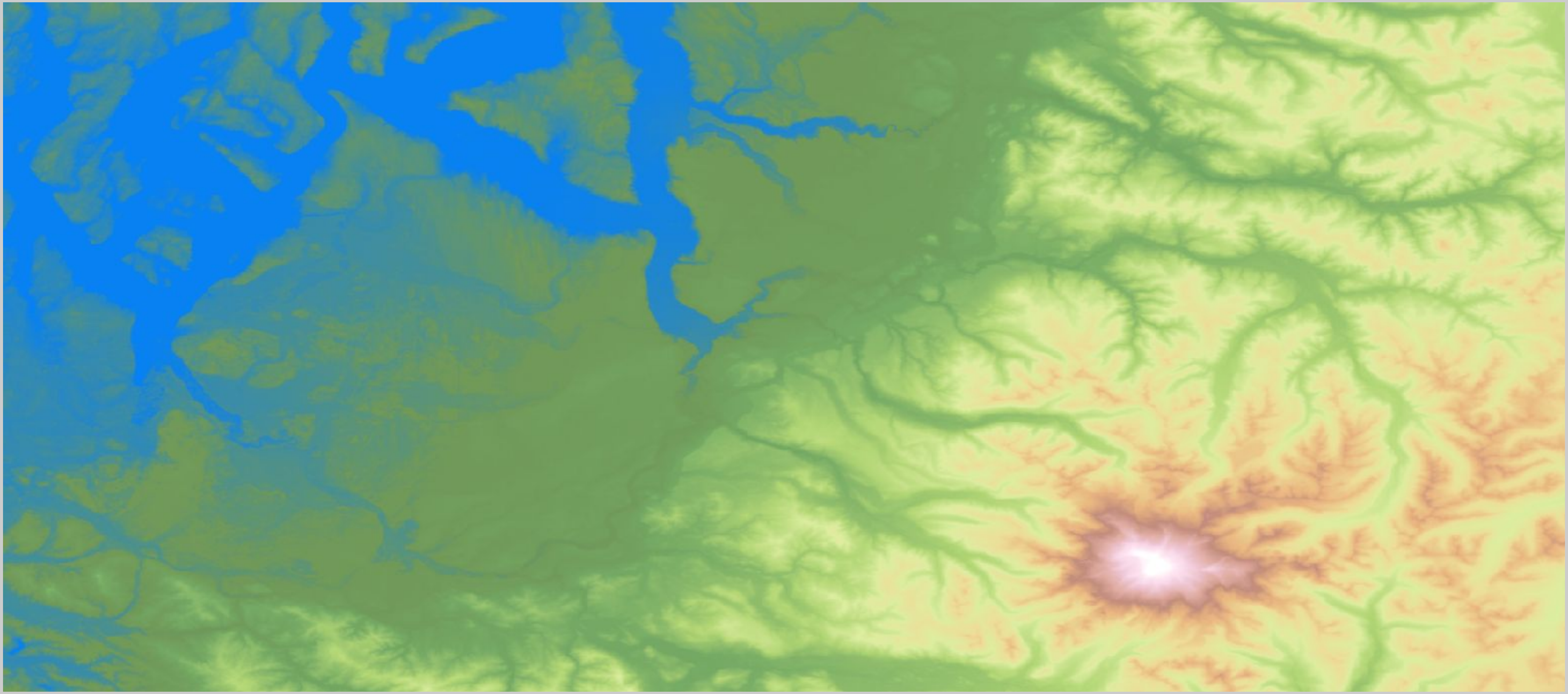
```
# set the projection and georeferencing info
dst_ds.SetProjection( src_ds.GetProjection() )
dst_ds.SetGeoTransform( src_ds.GetGeoTransform() )

# read the source file
for iY in range(src_ds.RasterYSize):
  src_data = src_band.ReadAsArray(0,iY,src_ds.RasterXSize,1)
  col_values = src_data[0] # array of z_values, one per row in source
data
  for iX in range(src_ds.RasterXSize):
    z_value = col_values[iX]
    [R,G,B] = MakeColor(z_value)
    band1[iY][iX] = R
    band2[iY][iX] = G
    band3[iY][iX] = B

# write each band out
dst_ds.GetRasterBand(1).WriteArray(band1)
dst_ds.GetRasterBand(2).WriteArray(band2)
dst_ds.GetRasterBand(3).WriteArray(band3)

dst_ds = None
```

# Hypsometric Tint



SRTM data over extents of Pierce County after gray2color.py

# Python API - vectors

```
"""Merge a group of shapefiles into one new one."""

import sys
import glob
from osgeo import ogr

outfile = "merge.shp"
file_list = glob.glob("*.shp")

# CREATE OUTPUT FILE
out_driver = ogr.GetDriverByName( 'ESRI Shapefile' )
out_ds = out_driver.CreateDataSource(outfile)
out_srs = None
out_layer = out_ds.CreateLayer("trans", out_srs, ogr.wkbLineString)
fd = ogr.FieldDefn('name', ogr.OFTString)
out_layer.CreateField(fd)
fd = ogr.FieldDefn('kV', ogr.OFTInteger)
out_layer.CreateField(fd)
```

cont. on next slide

# Python API - vectors (cont)

```
# READ EACH INPUT FILE AND WRITE FIELDS TO NEW FILE
for shapefile in file_list:
  print shapefile
  [filename, extension] = shapefile.split('.')
  in_drv = ogr.GetDriverByName( 'ESRI Shapefile' )
  in_ds = in_drv.Open(shapefile)
  in_layer = in_ds.GetLayer(0)
  in_feature = in_layer.GetNextFeature()
  kV_field = in_feature.GetFieldIndex('kV')
  counter = 1
  while in_feature is not None:
    name = filename + "_" + `counter`
    kV = in_feature.GetField(kV_field)
    out_feat = ogr.Feature(out_layer.GetLayerDefn())
    out_feat.SetField('name', name)
    out_feat.SetField('kV', kV)
    out_feat.SetGeometry(in_feature.GetGeometryRef().Clone())
    out_layer.CreateFeature(out_feat)
    out_layer.SyncToDisk()
    out_feat.Destroy()
    in_feature.Destroy()
    in_feature = in_layer.GetNextFeature()
    counter += 1
out_ds.Destroy()
```

# Combine the 2 types!

OSGeo Cascadia Chapter

- Because gdal/ogr can read both raster and vectors, it's possible to do some interesting combined processing tasks via the Python API.
- Contour line generation is one example
- Raster feature vectorizer is another
- Be creative, this tool is flexible

# Where to get GDAL / help

- http://www.gdal.org/ - main site, has examples for how to use the API, as well as each utility
- http://fwtools.maptools.org/ - includes support for some formats which are difficult to work with
- http://www.maptools.org/ms4w/ - comes with a ton of other stuff as well
- gdal-dev@lists.osgeo.org - mailing list primarily for developers, but users welcome as well
- #gdal irc channel on irc.freenode.net