Landsat_Preprocessing

May 8, 2014

Landsat Data Preprocessing
Chris Holden
Friday, May 9th, 2014

Overview: The Landsat family of satellites have been in space since 1972 and can provide a rich history of the land surface of the Earth over the last ~40 years. Recent advances in computing hardware, free access to the data, and new innovative cloud screening and atmospheric correction algorithms have enabled researchers to "data mine" this archive. Analyses that were previously only possible at a very coarse resolution (e.g., AVHRR and MODIS) are beginning to be possible at the 30m Landsat resolution.

This workshop intends to teach good practices for acquiring, organizing, preprocessing, and documenting your dense Landsat time series before ingestion into the "data mining" algorithm of your choice.

Topics:

1. Search for and filter Landsat acquisitions using EarthExplorer
2. Submit order for "Landsat Climate Data Record (CDR)" product using ESPA
3. Download completed order onto GEO cluster
4. Organize and extract imagery from archives
5. Combine dataset bands into "stacked" images of common geographic extent
6. Run Fmask using custom parameters on images
7. Create spatial subset using extent or polygon ROI
8. Create "preview" images to help filter through preprocessed imagery
9. Summarize and document your dataset

Notes on workshop

There are many ways to acquire and preprocess Landsat datasets and I will be demonstrating but one of the many approaches. Where applicable, I will demonstrate how to perform a step using a general method, and then by using one of my tools developed to help automate the general method for a specific workflow. The goal of showing both approaches will be to give you access to my specialized scripts or programs while helping you understand what they are automating.

0.1 Step 1. Search for and filter Landsat acquisitions using EarthExplorer

The USGS has many portals for filtering through Landsat data (e.g., GLOVIS, EarthExplorer, COVE tool, etc.), but the EarthExplorer tool is one of the better ones. If you don't have an account with EarthExplorer, you will need to create one before ordering data.

You will also need to know the WRS-2 path and row for the dataset you wish to acquire. Further information on the WRS-2 reference system, including a shapefile of all path and rows (download the descending dataset for Landsat) is available here:

https://landsat.usgs.gov/worldwide_reference_system_WRS.php

0.1.1 EarthExplorer

Navigate to:

http://earthexplorer.usgs.gov/

On the left side of the user interface you will see four tabs representing components of the search workflow. Begin by skipping the "Search Criteria" tab and continuing to the "Data Sets" tab.

0.1.2 Landsat CDR Datasets

We will be submitting an order for the Landsat Climate Data Record (CDR) product (http://landsat.usgs.gov/CDR_ECV.php) so that the USGS can perform the atmospheric correction and cloud masking preprocessing steps for us before we download our data.

In the list of available datasets below, find "Landsat CDR", expand the tab, and select both Landsat 7 ETM+ and Landsat 4-5 TM datasets.

0.1.3 Filter

Proceed by clicking the "Additional Cirteria >>" button:

The next window of this tabbed interface will allow you to customize your search result by various criteria for the Landsat 7 ETM+ and the Landsat 4-5 TM datasets.

- Input your WRS-2 path and row into the "WRS Path" and "WRS Row" forms
- Select "Less than 80%" under the "Cloud Cover" option (this cloud cover is estimated from ACCA, not Fmask)
- Ensure that the "Station Identifier" is set to "All"
- Select "Processing Required" and the "L1T" data type option for both sensors (e.g., ETM+ L1T and TM L1T). We select the "L1T" product type because it is the only orthorectified product from the Landsat Level 1 Product Generation System (LPGS). Orthorectification is important to assure consistent geolocation of pixels on the land surface. We include images that have the "Processing Required" status to include images that can be orthorectified to a L1T image, but have not yet been processed (and thus don't have the L1T metadata status in their database). For more information on LPGS, visit this URL: http://landsat.usgs.gov/Landsat_Processing_Details.php

Repeat this last step of adding filters for each Landsat sensor dataset – Landsat 7 ETM+ and Landsat 4-5 TM.

0.1.4 Results

Finally, click the "Results" button to submit your query.

0.1.5 Export

The last tab – "Results" – shows the Landsat imagery that met your search query.

Click the button that says "Click here to export your results" to export your results. Select "Non-Limited Results" under "Export Type". I also recommend selecting the option "Comma (,) Delimited" for the "Format".

Remember to export your results from Landsat 4-5 TM and Landsat 7 ETM+. The results will be retrievable via a URL sent to the email account you registered with. In my case, my results are available at the following URLs:

http://earthexplorer.usgs.gov/export/retrieve/?exportId=23721&exportKey=1f457c3397af85c4c8b755355eb4eb52

http://earthexplorer.usgs.gov/export/retrieve/?exportId=23722&exportKey=cc5e80832dca85903a510f4ac4ddb311

Manually click the links USGS provided you and download the ZIP files from their website. Unzip these files to find two comma separated text files.

0.2 Step 2: Submit order for "Landsat Climate Data Record (CDR)" product using ESPA

The two zip files containing our search results from the previous step contain the metadata required to further filter through the Landsat acquisitions. This includes information on the sun and sensor geometries, the goodness of fit for the terrain correction, the date and time of the acquisition, and more:

In [52]:
```
%%bash
tm=/projectnb/landsat/users/ceholden/2014_Landsat_Preprocess/1_EarthExplorer/LSR_LANDSAT_TM_23

echo -e "Landsat Metadata Fields:\n"

head -1 $tm | awk -F ',' '{ max=30; \
    for (i=1; i<=NF; i = i + 3) { \
        l1=(max - length($i)); l2=(max - length($(i + 1))); \
        print $i sprintf("%*s", l1, " ") $(i + 1) sprintf("%*s", l2, " ") $(i + 2)} \
    }'
```

```
Landsat Metadata Fields:

Result Number                 EE_DISPLAY_ID                 EE_ORDERING_ID
DOWNLOADABLE                  ON_DEMAND                     Landsat Scene Identifier
Spacecraft Identifier         Sensor Mode                   Station Identifier
Day/Night                     WRS Path                      WRS Row
WRS Type                      Date Acquired                 Start Time
Stop Time                     Sensor Anomalies              Acquisition Quality
Quality Band 1                Quality Band 2                Quality Band 3
Quality Band 4                Quality Band 5                Quality Band 6
Quality Band 7                Cloud Cover                   Cloud Cover Quad Upper Left
Cloud Cover Quad Upper Right  Cloud Cover Quad Lower Left   Cloud Cover Quad Lower Right
Sun Elevation                 Sun Azimuth                   Scene Center Latitude
Scene Center Longitude        Corner Upper Left Latitude    Corner Upper Left Longitude
Corner Upper Right Latitude   Corner Upper Right Longitude  Corner Lower Left Latitude
Corner Lower Left Longitude   Corner Lower Right Latitude   Corner Lower Right Longitude
Browse  Exists                Data Category                 Map Projection LORa
Data Type LORp                Data Type Level 1             Elevation Source
Output Format                 Ephemeris Type                Corner UL Latitude Product
Corner UL Longitude Product   Corner UR Latitude Product    Corner UR Longitude Product
Corner LL Latitude Product    Corner LL Longitude Product   Corner LR Latitude Product
Corner LR Longitude Product   Reflective Samples            Reflective Lines
Thermal Lines                 Thermal Samples               Ground Control Points Model
Geometric RMSE Model          Geometric RMSE Model X        Geometric RMSE Model Y
Geometric RMSE Verify         Map Projection Level 1        Datum
Ellipsoid                     UTM Zone                      Vertical Long from Pole
True Scale Latitude           False Easting                 False Northing
Grid Cell Size Reflective     Grid Cell Size Thermal        Orientation
Resampling Option             Scene Center Latitude dec     Scene Center Longitude dec
Corner Upper Left Lat dec     Corner Upper Left Long dec    Corner Upper Right Lat dec
Corner Upper Right Long dec   Corner Lower Left Lat dec     Corner Lower Left Long dec
Corner Lower Right Lat dec    Corner Lower Right Long dec
```

0.2.1 Extract IDs

For right now, let's just extract the Landsat IDs from our search result. We want to create a text file that only contains the Landsat IDs we want to order with each ID separated by row.

You could do this step in a variety of ways - Excel, a text editor, R, C, etc... - but let's just use some utilities in Bash:

```
In [53]: %%bash
         # First let's move to our working directory - USE YOUR OWN DIRECTORY
         cd /projectnb/landsat/users/ceholden/2014_Landsat_Preprocess/

         # Let's test our command and only print the first few lines using "head"
         cat 1_EarthExplorer/LSR_LANDSAT_ETM_COMBINED_23722.txt | awk -F ',' 'NR > 1 { print $2 }' | hea
```

```
LE70200472014109ASN00
LE70200472014093ASN00
LE70200472014077ASN00
LE70200472014061EDC00
LE70200472014045EDC00
```

```
In [54]: %%bash
         # Now let's save our IDs to our output file

         # My results are located within "1_EarthExplorer" and I will extract the IDs to a file named "
         output=2_ESPA/p020r047_submit.txt

         # First I create the text file using the output from AWK for Landsat 7 ETM+ using ">" to redir
         cat 1_EarthExplorer/LSR_LANDSAT_ETM_COMBINED_23722.txt | awk -F ',' 'NR > 1 { print $2 }' > $o
         # Now I append this text file with the IDs from Landsat 4-5 TM using ">>" to append
         cat 1_EarthExplorer/LSR_LANDSAT_TM_23721.txt | awk -F ',' 'NR > 1 { print $2 }' >> $output
```

```
In [57]: %%bash
         # We can check to see how many images we have by concatenating our output file and piping it t
         n=$(cat 2_ESPA/p020r047_submit.txt | wc -l)

         echo "You have $n Landsat images in your text file"
```

```
You have 438 Landsat images in your text file
```

0.2.2 Submit text file of Landsat IDs to EROS Science Processing Architecture (ESPA)

In a web browser, navigate to:

https://espa.cr.usgs.gov

and login using your username and password (they're all defaults, so you can also ask me for now, but it is important to register for tracking purposes).

Once you log in, you'll be taken to the order page:

Enter your email address, click "Browse", and navigate to your list of Landsat IDs.

You have many dataset options for your download, but it is always a good idea to download the "Source Metadata" because this product contains the Landsat "MTL" metadata file.

As pictured, I suggest downloading the following in addition to the metadata:

- "Source Products"
  - original TIF imagery in units of DNs
- "Surface Reflectance"
  - "lndsr*hdf" LEDAPS output, and the "CFmask" cloud mask
- "Band 6 Brightness Temperature"
  - "lndth*hdf" LEDAPS thermal brightness temperature

I recommend downloading the "Source Products" even though the "Surface Reflectance" imagery has a Fmask cloud mask band because the Landsat CDR version of Fmask is ran using default parameters (cloud probability = 22.5, and cloud, shadow, and snow buffering of 3 pixels), but you may require a more conservative mask depending on your scene location or usage scenarios.

You also have the potential of having your output datasets resized, reprojected, or subset.

When you have submitted your order, you will receive a confirmation email and will be taken to your order status page:

0.3 Step 3. Download completed order onto GEO cluster

When you receive an email update notifying you that your images have been processed, you may access them here:

http://espa.cr.usgs.gov/status/

Simply input the email address associated with your order, select your order, and you'll find a page with each image and it's status:

The ESPA interface provides you with a URL for downloading the image archive and a checksum useful for assuring your download completed without error (see http://en.wikipedia.org/wiki/Checksum).

Download onto the cluster

The quickest and easiest way of downloading these datasets onto the cluster would be using Firefox's "DownThemAll" addon because it easily allows for parallel downloads (https://addons.mozilla.org/en-US/firefox/addon/downthemall/).

One simple way of downloading the datasets would be using "wget" (enter "man wget" for more information).

In this example, we use "wget" with several options: + "-nH" specifies no host directories will be downloaded + "-nc" specifies "no clobber", or no overwrite + "–cut-dirs=2" works with "-nH" to remove directories containing our files + "-r" specifies recursion, so we can find our files beneath the download URL + "-A L*2014*-*tar.gz,L*2014*-*cksum" is an accept list that specifies a pattern for files we will download. Files not matching the pattern are ignored + "-nv" will turn off verbose

```
In [16]: %%bash
         cd /projectnb/landsat/users/ceholden/2014_Landsat_Preprocess/3_Download

         url=http://espa.cr.usgs.gov/status/ceholden%40bu.edu-572014-17955/

         wget -nH -nc --cut-dirs=2 -r -A L*2014*-*tar.gz,L*2014*-*cksum -nv $url

         echo "Done!"

Process is terminated.
```

Validate checksums of downloads

It is important, but not always necessary, to assert that the files stored on the USGS's servers are fully intact once downloaded to our cluster. We can assert that the files are in tact with high confidence by checking to make sure the "checksum" of the file matches once downloaded to BU.

For each archive you download, there will be a corresponding "*.cksum" file. The checksum provided is output from the Unix "cksum" command (http://en.wikipedia.org/wiki/Cksum). Unfortuantely, there isn't a built-in mechanism in this program to validate checksums, so we'll do this in a Bash loop and if statement.

```
In [18]: %%bash
         cd /projectnb/landsat/users/ceholden/2014_Landsat_Preprocess/3_Download

         # Loop over cksum files
         for checksum in $(find ./ -name 'L*.cksum'); do
             # Find basename of file and remove extension to match up with archive
             bn=$(basename $checksum | awk -F '.' '{ print $1 }')
```

```
        # Test to see if archive exists
        archive=${bn}.tar.gz
        if [ ! -f $archive ]; then
            echo "$bn has no matching archive"
            continue
        fi

        # If archive exists, then validate checksum
        test=$(cksum $archive)
        if [ "$test" != "$(cat $checksum)" ]; then
            echo "!!!!! WARNING !!!!!"
            echo "$bn may be corrupted"
            echo "!!!!! WARNING !!!!!"
        else
            echo "$bn is OK"
        fi
    done
```

```
LE70200472014093-SC20140507171414 is OK
LE70200472014109-SC20140507171414 is OK
LE70200472014077-SC20140507171413 is OK
LE70200472014061-SC20140507171414 is OK
```

0.4 Step 4. Organize and extract imagery from archives

If you have any prior experience with Landsat imagery, you'll notice that the archives we downloaded have an unusual filename:

LE70200472011261-SC20140507171414.tar.gz

The first half of this string is the Landsat ID, minus the receiving station information. The second half of this string details the timing of when the Landsat image was processed through ESPA.

Since we're not very interested in the processing detail, we will extract each of these archives to a folder based on the reference ID of the Landsat image.

In [21]:
```
%%bash
cd /projectnb/landsat/users/ceholden/2014_Landsat_Preprocess/4_Organize

# We want to find things ONLY in our current directory, not in any subfolders
#     So, we use -maxdepth 1 option
#     You could also just use "ls *tar.gz",
#     but find is good to know because it gives you a lot of control
n=$(find ./ -maxdepth 1 -name '*tar.gz' | wc -l)
i=1

for archive in $(find ./ -maxdepth 1 -name '*tar.gz'); do
    echo "<----- $i / $n: $(basename $img)"

    # Create temporary folder for storage
    mkdir temp

    # Extract archive to temporary folder
    tar -xzvf $archive -C temp/
```

```
    # Find ID based on MTL file's filename
    mtl=$(find temp/ -name 'L*MTL.txt')

    # Test to make sure we found it
    if [ ! -f $mtl ]; then
        echo "Could not find MTL file for $archive"
        break
    fi

    # Use AWK to remove _MTL.txt
    id=$(basename $mtl | awk -F '_' '{ print $1 }')

    # Move archive into temporary folder
    mv $archive temp/

    # Rename archive
    mv temp $id

    # Iterate count
    let i+=1
done

echo "Done!"
```

```
Done!
```

Filter L1G images

If you recall when we filtered our dataset from the Landsat archive, we had selected the "Processing Required" option. This means that some of the images we downloaded that had this status will be L1T, but some of them will be L1G (not orthorectified).

In order to remove the L1G images, we can run through our image stacks and do a quick "grep" on each MTL file:

```
In [83]: %%bash
         cd /projectnb/landsat/users/ceholden/2014_Landsat_Preprocess/4_Organize

         if [ ! -d L1G ]; then
             mkdir L1G/
         fi

         for mtl in $(find ./ -name 'L*MTL.txt'); do
             id=$(basename $(dirname $mtl))

             l1t=$(grep "L1T" $mtl)

             if [ "$l1t" == "" ]; then
                 echo "$id is not L1T"
                 mv $(dirname $mtl) L1G
             else
                 echo "$id is L1T"
             fi

         done
```

```
LE70200472014061EDC00 is L1T
LE70200472014077ASN00 is L1T
LE70200472014093ASN00 is L1T
LE70200472014109ASN00 is L1T

mkdir: cannot create directory ‘L1G/’: File exists
```

0.5 Step 5. Combine dataset bands into "stacked" images of common geographic extent

We now have the images organized by their Landsat ID and extracted from the archives. The next step in our workflow is to create "layer stacks" - or multiband images - that contain the bands we want to use in our analysis. For example, we might want to create a multiband image for each Landsat ID that contains all Landsat bands, ordered by wavelength, and the Fmask result.

This workflow can be accomplished using a variety of software - ENVI, ArcGIS, ERDAS, R, and QGIS, for example - but it would be enormously time consuming to use these programs to accomplish our goal for hundreds of images. Instead, we will use the free program ***gdal_merge.py*** from the GDAL command line set of programs (http://www.gdal.org/ or http://www.gdal.org/gdal_merge.html).

One complicating factor in this workflow is that our LEDAPS output are currently stored as HDF files which stores bands as "subdatasets" instead of bands, making referencing them a little difficult (see http://www.gdal.org/frmt_hdf4.html).

Sometime soon, the ESPA processing system will allow you to download your datasets as either HDF, GeoTIFF, or binary files (http://landsat.usgs.gov/sr_samples.php).

First we will begin my loading the module for GDAL and inspecting one of the LEDAPS outputs.

```
In [45]: %%bash
         module purge
         module load gdal/1.10.0

         cd /projectnb/landsat/users/ceholden/2014_Landsat_Preprocess/5_Stack/LE70200472014061EDC00

         gdalinfo lndsr.LE70200472014061EDC00.hdf
```

```
Driver: HDF4/Hierarchical Data Format Release 4
Files: lndsr.LE70200472014061EDC00.hdf
Size is 512, 512
Coordinate System is ‘’
Metadata:
  AcquisitionDate=2014-03-02T16:19:41.029800Z
  CFmaskVersion=1.3.0
  Cloud Mask Algo Version=CMReflectanceBasedv1.0
  DataProvider=USGS/EROS
  EastBoundingCoordinate=-88.6776995141039
  HDFEOSVersion=4.2
  HDFVersion=4.2.5
  Instrument=ETM
  LEDAPSVersion=1.3.1
  Level1ProductionDate=2014-03-02T00:00:00.000000Z
  LocalGranuleID=L7ESR.a2014061.w2p020r047.020.2014127222256.hdf
  LowerRightCornerLatLong=17.8410695070442, -88.6778422897102
  LPGSMetadataFile=LE70200472014061EDC00_MTL.txt
  NorthBoundingCoordinate=19.7654335783953
  OrientationAngle=0
  PixelSize=30
  ProductionDate=2014-05-07T22:22:56Z
```

```
  ReflBias=-6.97874021530151, -7.19881916046143, -5.62165403366089, -6.06929111480713, -1.1262199878692
  ReflGains=0.778739988803864, 0.798819005489349, 0.621653020381927, 0.969290971755981, 0.1262200027704
  Satellite=LANDSAT_7
  ShortName=L7ESR
  SolarAzimuth=130.5961761
  SolarZenith=37.7718544
  SouthBoundingCoordinate=17.8077011670647
  ThermalBias=-0.0670870020985603
  ThermalGain=0.0670870020985603
  UpperLeftCornerLatLong=19.7282076307976, -91.0190580192774
  WestBoundingCoordinate=-91.0192042045475
  WRS_Path=20
  WRS_Row=47
  WRS_System=2
Subdatasets:
  SUBDATASET_1_NAME=HDF4_EOS:EOS_GRID:"lndsr.LE70200472014061EDC00.hdf":Grid:band1
  SUBDATASET_1_DESC=[7101x8121] band1 Grid (16-bit integer)
  SUBDATASET_2_NAME=HDF4_EOS:EOS_GRID:"lndsr.LE70200472014061EDC00.hdf":Grid:band2
  SUBDATASET_2_DESC=[7101x8121] band2 Grid (16-bit integer)
  SUBDATASET_3_NAME=HDF4_EOS:EOS_GRID:"lndsr.LE70200472014061EDC00.hdf":Grid:band3
  SUBDATASET_3_DESC=[7101x8121] band3 Grid (16-bit integer)
  SUBDATASET_4_NAME=HDF4_EOS:EOS_GRID:"lndsr.LE70200472014061EDC00.hdf":Grid:band4
  SUBDATASET_4_DESC=[7101x8121] band4 Grid (16-bit integer)
  SUBDATASET_5_NAME=HDF4_EOS:EOS_GRID:"lndsr.LE70200472014061EDC00.hdf":Grid:band5
  SUBDATASET_5_DESC=[7101x8121] band5 Grid (16-bit integer)
  SUBDATASET_6_NAME=HDF4_EOS:EOS_GRID:"lndsr.LE70200472014061EDC00.hdf":Grid:band7
  SUBDATASET_6_DESC=[7101x8121] band7 Grid (16-bit integer)
  SUBDATASET_7_NAME=HDF4_EOS:EOS_GRID:"lndsr.LE70200472014061EDC00.hdf":Grid:atmos_opacity
  SUBDATASET_7_DESC=[7101x8121] atmos_opacity Grid (16-bit integer)
  SUBDATASET_8_NAME=HDF4_EOS:EOS_GRID:"lndsr.LE70200472014061EDC00.hdf":Grid:fill_QA
  SUBDATASET_8_DESC=[7101x8121] fill_QA Grid (8-bit unsigned integer)
  SUBDATASET_9_NAME=HDF4_EOS:EOS_GRID:"lndsr.LE70200472014061EDC00.hdf":Grid:DDV_QA
  SUBDATASET_9_DESC=[7101x8121] DDV_QA Grid (8-bit unsigned integer)
  SUBDATASET_10_NAME=HDF4_EOS:EOS_GRID:"lndsr.LE70200472014061EDC00.hdf":Grid:cloud_QA
  SUBDATASET_10_DESC=[7101x8121] cloud_QA Grid (8-bit unsigned integer)
  SUBDATASET_11_NAME=HDF4_EOS:EOS_GRID:"lndsr.LE70200472014061EDC00.hdf":Grid:cloud_shadow_QA
  SUBDATASET_11_DESC=[7101x8121] cloud_shadow_QA Grid (8-bit unsigned integer)
  SUBDATASET_12_NAME=HDF4_EOS:EOS_GRID:"lndsr.LE70200472014061EDC00.hdf":Grid:snow_QA
  SUBDATASET_12_DESC=[7101x8121] snow_QA Grid (8-bit unsigned integer)
  SUBDATASET_13_NAME=HDF4_EOS:EOS_GRID:"lndsr.LE70200472014061EDC00.hdf":Grid:land_water_QA
  SUBDATASET_13_DESC=[7101x8121] land_water_QA Grid (8-bit unsigned integer)
  SUBDATASET_14_NAME=HDF4_EOS:EOS_GRID:"lndsr.LE70200472014061EDC00.hdf":Grid:adjacent_cloud_QA
  SUBDATASET_14_DESC=[7101x8121] adjacent_cloud_QA Grid (8-bit unsigned integer)
  SUBDATASET_15_NAME=HDF4_EOS:EOS_GRID:"lndsr.LE70200472014061EDC00.hdf":Grid:band6
  SUBDATASET_15_DESC=[7101x8121] band6 Grid (16-bit integer)
  SUBDATASET_16_NAME=HDF4_EOS:EOS_GRID:"lndsr.LE70200472014061EDC00.hdf":Grid:band6_fill_QA
  SUBDATASET_16_DESC=[7101x8121] band6_fill_QA Grid (8-bit unsigned integer)
  SUBDATASET_17_NAME=HDF4_EOS:EOS_GRID:"lndsr.LE70200472014061EDC00.hdf":Grid:fmask_band
  SUBDATASET_17_DESC=[7101x8121] fmask_band Grid (8-bit unsigned integer)
Corner Coordinates:
Upper Left  (    0.0,    0.0)
Lower Left  (    0.0,  512.0)
Upper Right (  512.0,    0.0)
Lower Right (  512.0,  512.0)
```

```
Center      (  256.0,  256.0)
```

The LEDAPS surface reflectance HDF file contains 17 subdatasets which store the surface reflectance bands, a variety of QA/QC bands, the thermal brightness temperature band, and the Fmask band.

In order to reference an individual subdataset from this file, we have to reference the name of the subdataset directly. For example, "HDF4_EOS:EOS_GRID:"lndsr.LE70200472014061EDC00.hdf":Grid:fmask_band".

```
In [46]: %%bash
         cd /projectnb/landsat/users/ceholden/2014_Landsat_Preprocess/5_Stack/LE70200472014061EDC00

         gdalinfo HDF4_EOS:EOS_GRID:"lndsr.LE70200472014061EDC00.hdf":Grid:fmask_band
```

```
Driver: HDF4Image/HDF4 Dataset
Files: lndsr.LE70200472014061EDC00.hdf
Size is 8121, 7101
Coordinate System is:
PROJCS["UTM Zone 16, Northern Hemisphere",
    GEOGCS["Unknown datum based upon the WGS 84 ellipsoid",
        DATUM["Not specified (based on WGS 84 spheroid)",
            SPHEROID["WGS 84",6378137,298.257223563,
                AUTHORITY["EPSG","7030"]]],
        PRIMEM["Greenwich",0],
        UNIT["degree",0.0174532925199433]],
    PROJECTION["Transverse_Mercator"],
    PARAMETER["latitude_of_origin",0],
    PARAMETER["central_meridian",-87],
    PARAMETER["scale_factor",0.9996],
    PARAMETER["false_easting",500000],
    PARAMETER["false_northing",0],
    UNIT["Meter",1]]
Origin = (78585.000000000000000,2186415.000000000000000)
Pixel Size = (30.000000000000000,-30.000000000000000)
Metadata:
  _FillValue=255
  AcquisitionDate=2014-03-02T16:19:41.029800Z
  CFmaskVersion=1.3.0
  Cloud Mask Algo Version=CMReflectanceBasedv1.0
  DataProvider=USGS/EROS
  EastBoundingCoordinate=-88.6776995141039
  HDFEOSVersion=4.2
  HDFVersion=4.2.5
  Instrument=ETM
  LEDAPSVersion=1.3.1
  Level1ProductionDate=2014-03-02T00:00:00.000000Z
  LocalGranuleID=L7ESR.a2014061.w2p020r047.020.2014127222256.hdf
  long_name=fmask_band
  LowerRightCornerLatLong=17.8410695070442, -88.6778422897102
  LPGSMetadataFile=LE70200472014061EDC00_MTL.txt
  mask_index=0 clear; 1 water; 2 cloud_shadow; 3 snow; 4 cloud
  NorthBoundingCoordinate=19.7654335783953
  OrientationAngle=0
  PixelSize=30
  ProductionDate=2014-05-07T22:22:56Z
  ReflBias=-6.97874021530151, -7.19881916046143, -5.62165403366089, -6.06929111480713, -1.1262199878692
```

```
  ReflGains=0.778739988803864, 0.798819005489349, 0.621653020381927, 0.969290971755981, 0.12622000277041
  Satellite=LANDSAT_7
  ShortName=L7ESR
  SolarAzimuth=130.5961761
  SolarZenith=37.7718544
  SouthBoundingCoordinate=17.8077011670647
  ThermalBias=-0.0670870020985603
  ThermalGain=0.0670870020985603
  UpperLeftCornerLatLong=19.7282076307976, -91.0190580192774
  valid_range=0, 4
  WestBoundingCoordinate=-91.0192042045475
  WRS_Path=20
  WRS_Row=47
  WRS_System=2
Corner Coordinates:
Upper Left  (   78585.000, 2186415.000) ( 91d 1' 9.14"W, 19d43'42.02"N)
Lower Left  (   78585.000, 1973385.000) ( 90d58'25.95"W, 17d48'27.72"N)
Upper Right (  322215.000, 2186415.000) ( 88d41'48.80"W, 19d45'55.56"N)
Lower Right (  322215.000, 1973385.000) ( 88d40'39.72"W, 17d50'27.37"N)
Center      (  200400.000, 2079900.000) ( 89d50'31.44"W, 18d47'21.19"N)
Band 1 Block=8121x123 Type=Byte, ColorInterp=Gray
  Description = fmask_band
  NoData Value=255
```

One thing to note is that the HDF files the USGS provides do not properly describe the datum of the datasets.

```
GEOGCS["Unknown datum based upon the WGS 84 ellipsoid",
```

From the "Landsat Dictionary" (https://lta.cr.usgs.gov/landsat_dictionary.html#ellipsoid), we know that Landsat data is using the WGS-84 ellipsoid.

Fortunately, we can overwrite the projection with the correct one by either using the "proj4" string from one of the TIF files:

```
In [48]: %%bash
         cd /projectnb/landsat/users/ceholden/2014_Landsat_Preprocess/5_Stack/LE70200472014061EDC00

         gdalinfo -proj4 LE70200472014061EDC00_B1.TIF | grep "+proj"
```

```
'+proj=utm +zone=16 +datum=WGS84 +units=m +no_defs '
```

Stacked images

To create a "layer stack" from this LEDAPS output, we first need to identify which bands we would like to use and which subdatasets the bands are contained in.

For a "layer stack" of the optical bands, the thermal band, and the Fmask band, we would reference the first 6 subdatasets, the 15th, and the 17th. Since we know "gdalinfo" will provide us with the correct references to the subdatasets, we can loop over these subdataset indices and store the names:

```
In [38]: %%bash
         cd /projectnb/landsat/users/ceholden/2014_Landsat_Preprocess/5_Stack/LE70200472014061EDC00

         image=lndsr.LE70200472014061EDC00.hdf

         bands="1 2 3 4 5 6 15 17"
```

```
sds_names=""

for b in $bands; do
    gdalinfo $image | grep "SUBDATASET_${b}_NAME"
    sds=$(gdalinfo $image | grep "SUBDATASET_${b}_NAME" | awk -F '=' '{ print $2 }')
    sds_names="$sds_names $sds"
done

echo "All subdatasets:"
echo $sds_names
```

```
SUBDATASET_1_NAME=HDF4_EOS:EOS_GRID:"lndsr.LE70200472014061EDC00.hdf":Grid:band1
  SUBDATASET_2_NAME=HDF4_EOS:EOS_GRID:"lndsr.LE70200472014061EDC00.hdf":Grid:band2
  SUBDATASET_3_NAME=HDF4_EOS:EOS_GRID:"lndsr.LE70200472014061EDC00.hdf":Grid:band3
  SUBDATASET_4_NAME=HDF4_EOS:EOS_GRID:"lndsr.LE70200472014061EDC00.hdf":Grid:band4
  SUBDATASET_5_NAME=HDF4_EOS:EOS_GRID:"lndsr.LE70200472014061EDC00.hdf":Grid:band5
  SUBDATASET_6_NAME=HDF4_EOS:EOS_GRID:"lndsr.LE70200472014061EDC00.hdf":Grid:band7
  SUBDATASET_15_NAME=HDF4_EOS:EOS_GRID:"lndsr.LE70200472014061EDC00.hdf":Grid:band6
  SUBDATASET_17_NAME=HDF4_EOS:EOS_GRID:"lndsr.LE70200472014061EDC00.hdf":Grid:fmask_band
All subdatasets:
HDF4_EOS:EOS_GRID:"lndsr.LE70200472014061EDC00.hdf":Grid:band1 HDF4_EOS:EOS_GRID:"lndsr.LE70200472014061
```

gdal_merge.py

Now that we know which subdatasets we would like to include, we can create our "layer stack" using ***gdal_merge.py*** (see http://www.gdal.org/gdal_merge.html)

```
In [32]: %%bash
         gdal_merge.py --help
```

```
Unrecognised command option: --help
Usage: gdal_merge.py [-o out_filename] [-of out_format] [-co NAME=VALUE]*
                     [-ps pixelsize_x pixelsize_y] [-tap] [-separate] [-q] [-v] [-pct]
                     [-ul_lr ulx uly lrx lry] [-init "value [value...]"]
                     [-n nodata_value] [-a_nodata output_nodata_value]
                     [-ot datatype] [-createonly] input_files
                     [--help-general]
```

To create an "ENVI" formatted binary image, with band interleave by pixel interleave, we can modify our previous code:

```
In [49]: %%bash
         cd /projectnb/landsat/users/ceholden/2014_Landsat_Preprocess/5_Stack/LE70200472014061EDC00

         image=lndsr.LE70200472014061EDC00.hdf
         output=LE70200472014061EDC00_test_stack.bip

         bands="1 2 3 4 5 6 15 17"

         sds_names=""

         for b in $bands; do
             sds=$(gdalinfo $image | grep "SUBDATASET_${b}_NAME" | awk -F '=' '{ print $2 }')
             sds_names="$sds_names $sds"
         done
```

```
        # Create the stack
        gdal_merge.py -o temp.gtif -separate -a_nodata -9999 $sds_names

        # Correct the projection using gdal_translate
        gdal_translate -of ENVI -co "INTERLEAVE=BIP" -a_srs '+proj=utm +zone=16 +datum=WGS84 +units=m

        # Delete the temporary GeoTIFF
        rm temp.*

        # Query the new stack for metadata
        gdalinfo $output
```

```
0...10...20...30...40...50...60...70...80...90...100 - done.
Input file size is 8121, 7101
0...10...20...30...40...50...60...70...80...90...100 - done.
Driver: ENVI/ENVI .hdr Labelled
Files: LE70200472014061EDC00_test_stack.bip
       LE70200472014061EDC00_test_stack.bip.aux.xml
       LE70200472014061EDC00_test_stack.hdr
Size is 8121, 7101
Coordinate System is:
PROJCS["WGS_1984_UTM_Zone_16N",
    GEOGCS["GCS_WGS_1984",
        DATUM["WGS_1984",
            SPHEROID["WGS_84",6378137,298.257223563]],
        PRIMEM["Greenwich",0],
        UNIT["Degree",0.017453292519943295]],
    PROJECTION["Transverse_Mercator"],
    PARAMETER["latitude_of_origin",0],
    PARAMETER["central_meridian",-87],
    PARAMETER["scale_factor",0.9996],
    PARAMETER["false_easting",500000],
    PARAMETER["false_northing",0],
    UNIT["Meter",1]]
Origin = (78585.000000000000000,2186415.000000000000000)
Pixel Size = (30.000000000000000,-30.000000000000000)
Metadata:
  AREA_OR_POINT=Area
  Band_1=Band 1
  Band_2=Band 2
  Band_3=Band 3
  Band_4=Band 4
  Band_5=Band 5
  Band_6=Band 6
  Band_7=Band 7
  Band_8=Band 8
Image Structure Metadata:
  INTERLEAVE=PIXEL
Corner Coordinates:
Upper Left  (   78585.000, 2186415.000) ( 91d 1' 9.14"W, 19d43'42.02"N)
Lower Left  (   78585.000, 1973385.000) ( 90d58'25.95"W, 17d48'27.72"N)
Upper Right (  322215.000, 2186415.000) ( 88d41'48.80"W, 19d45'55.56"N)
Lower Right (  322215.000, 1973385.000) ( 88d40'39.72"W, 17d50'27.37"N)
Center      (  200400.000, 2079900.000) ( 89d50'31.44"W, 18d47'21.19"N)
```

```
Band 1 Block=8121x1 Type=Int16, ColorInterp=Undefined
  Description = Band 1
Band 2 Block=8121x1 Type=Int16, ColorInterp=Undefined
  Description = Band 2
Band 3 Block=8121x1 Type=Int16, ColorInterp=Undefined
  Description = Band 3
Band 4 Block=8121x1 Type=Int16, ColorInterp=Undefined
  Description = Band 4
Band 5 Block=8121x1 Type=Int16, ColorInterp=Undefined
  Description = Band 5
Band 6 Block=8121x1 Type=Int16, ColorInterp=Undefined
  Description = Band 6
Band 7 Block=8121x1 Type=Int16, ColorInterp=Undefined
  Description = Band 7
Band 8 Block=8121x1 Type=Int16, ColorInterp=Undefined
  Description = Band 8
```

Output extent

Since we will be utilizing a large number of images in whatever analysis we perform, it would be convenient if all of the "layer stack" images share a common image size and geographic extent. If they did not, then our algorithm would have to perform some arithmetic to determine which pixel it should read in from every image. Instead, it is much easier to know that every row and column corresponds to the same extent on the ground.

In order to do this, the simplest approach would be to define an output extent based on one image. A more sophisticated approach would be to define the output extent as the maximum extent required to fit all images, but this is too difficult in Bash to demonstrate.

In [53]:
```
%%bash
cd /projectnb/landsat/users/ceholden/2014_Landsat_Preprocess/5_Stack/

# Define "extent" function (from https://github.com/dwtkns/gdal-cheat-sheet)
function gdal_extent() {
    if [ -z "$1" ]; then
        echo "Missing arguments. Syntax:"
        echo "  gdal_extent <input_raster>"
        return
    fi
    EXTENT=$(gdalinfo $1 |\
        grep "Upper Left\|Lower Right" |\
        sed "s/Upper Left  //g;s/Lower Right //g;s/).*//g" |\
        tr "\n" " " |\
        sed 's/ *$//g' |\
        tr -d "[(,]")
    echo -n "$EXTENT"
}

# Define "subdataset" function
function get_sds() {
    if [ -z "$1" ] || [ -z "$2" ]; then
        echo "Missing arguments. Usage:"
        echo "    get_sds <raster> <bands>"
        return
    fi

    sds_names=""
```

```
    for b in $2; do
        sds=$(gdalinfo $1 | grep "SUBDATASET_${b}_NAME" | awk -F '=' '{ print $2 }')
        sds_names="$sds_names $sds"
    done
    echo "$sds_names"
}

# Define index so we only grab extent from first
i=0
# Bands of interest
bands="1 2 3 4 5 6 15 17"
# Projection "proj4" string to use
proj4="+proj=utm +zone=16 +datum=WGS84 +units=m +no_defs "

# Loop through all image directories
for img in $(find ./ -maxdepth 1 -name 'L*' -type d); do
    echo "Creating layer stack for $img"

    # Find LEDAPS file
    lndsr=$(find $img -name 'lndsr*hdf')
    # Get subdatasets - quote bands so they're all sent as £2
    subdatasets=$(get_sds $lndsr "$bands")

    # Get extent of first image we find
    if [ $i -eq 0 ]; then
        # Extent
        sds=$(echo $subdatasets | awk '{ print $1 }')
        extent=$(gdal_extent $sds)
    fi

    # Find image Landsat ID for filename
    id=$(basename $img)

    output=$img/${id}_stack

    # Create layer stack
    gdal_merge.py -o temp.gtif -ul_lr $extent -separate -a_nodata -9999 $subdatasets

    # Now translate this into ENVI BIP with correct projection
    gdal_translate -of ENVI -co "INTERLEAVE=BIP" -a_srs "$proj4" temp.gtif $output

    # Iterate i and delete temp file
    let i+=1
    rm temp.*
done
```

```
Creating layer stack for ./LE70200472014061EDC00
0...10...20...30...40...50...60...70...80...90...100 - done.
Input file size is 8121, 7101
0...10...20...30...40...50...60...70...80...90...100 - done.
Creating layer stack for ./LE70200472014077ASN00
0...10...20...30...40...50...60...70...80...90...100 - done.
Input file size is 8121, 7101
0...10...20...30...40...50...60...70...80...90...100 - done.
```

```
Creating layer stack for ./LE70200472014093ASN00
0...10...20...30...40...50...60...70...80...90...100 - done.
Input file size is 8121, 7101
0...10...20...30...40...50...60...70...80...90...100 - done.
Creating layer stack for ./LE70200472014109ASN00
0...10...20...30...40...50...60...70...80...90...100 - done.
Input file size is 8121, 7101
0...10...20...30...40...50...60...70...80...90...100 - done.
```

We can check out output to make sure they're all the same size and upper left coordinate:

```
In [55]: %%bash
         cd /projectnb/landsat/users/ceholden/2014_Landsat_Preprocess/5_Stack/

         for stack in $(find ./ -name '*stack'); do
             gdalinfo $stack | grep "Size is"
             gdalinfo $stack | grep "Origin ="
             du -hs $stack
         done
```

```
Size is 8121, 7101
Origin = (78585.000000000000000,2186415.000000000000000)
880M        ./LE70200472014061EDC00/LE70200472014061EDC00_stack
Size is 8121, 7101
Origin = (78585.000000000000000,2186415.000000000000000)
880M        ./LE70200472014077ASN00/LE70200472014077ASN00_stack
Size is 8121, 7101
Origin = (78585.000000000000000,2186415.000000000000000)
880M        ./LE70200472014093ASN00/LE70200472014093ASN00_stack
Size is 8121, 7101
Origin = (78585.000000000000000,2186415.000000000000000)
880M        ./LE70200472014109ASN00/LE70200472014109ASN00_stack
```

Complicated stuff!

Wow - that's a lot of Bash work! You could formalize these lines of code into your own Bash script, of course, but it probably isn't the best language for the job.

In order to create these layer stacks myself, I ended up writing a heavily modified version of ***gdal_merge.py*** that does the following: - Determines the output extent of your stack by: - the maximum extent of all datasets - the minimum extent of all datasets - a specified extent - a percentile of the extents (e.g., 1% for minimum and 99% for maximum) - Treats HDF subdatasets as bands, removing necessity for referencing them directly - Preserves band name metadata (i.e., so your stack has the name "band 1 reflectance" from the LEDAPS HDF file) - Applies NDV separately per band - Can overwrite UTM zone to fix datum - Can "resume" if interrupted

If you'd like to use this solution, you may do so as follows:

```
In [65]: %%bash
         module load gdal/1.10.0
         module load batch_landsat

         landsat_stack.py --help
```

```
Stack Landsat Data

Usage: landsat_stack.py [options] (--max_extent | --min_extent |
    --extent=<extent> | --percentile=<pct>) <location>
```

```
Options:
    -f --files=<files>...       Files to stack [default: lndsr.*.hdf *Fmask]
    -b --bands=<bands>...       Bands from files to stack [default: all]
    -d --dirs=<pattern>         Directory name pattern to search [default: L*]
    -o --output=<pattern>       Output filename pattern [default: *stack]
    -p --pickup                 Pickup / resume where left off
    -n --ndv=<ndv>              No data value [default: 0]
    -u --utm=<zone>             Force a UTM zone (in WGS84)
    -e --exit-on-warn           Exit on warning messages
    --format=<format>           GDAL format [default: ENVI]
    --co=<creation options>     GDAL creation options [default: None]
    -v --verbose                Show verbose debugging messages
    -q --quiet                  Be quiet by not showing warnings
    --dry-run                   Dry run - don't actually stack
    -h --help                   Show help

Examples:
    landsat_stack.py -vq -n "-9999; 255" -b "1 2 3 4 5 6 15; 1" --min_extent ./
```

Example:

Let's say we want the same bands, but want to specify that the extent of the images be the maximum of all images in the stack so nothing gets clipped. We can also specify NoDataValues for each band:

```
In [64]: %%bash
         cd /projectnb/landsat/users/ceholden/2014_Landsat_Preprocess/5_Stack/

         module load gdal/1.10.0
         module load batch_landsat

         landsat_stack.py -q -p --files "lndsr*hdf; *Fmask" \
             -b "1 2 3 4 5 6 15; 1" \
             -n "-9999 -9999 -9999 -9999 -9999 -9999; 255" \
             --utm 16 -o "*_stack_chris" \
             --format "ENVI" --co "INTERLEAVE=BIP" --max_extent ./

Found 4 Landsat images to stack.
Finding maximum extent
Landsat ID: LE70200472014061EDC00 updated maximum extent
Landsat ID: LE70200472014077ASN00 updated maximum extent
Landsat ID: LE70200472014093ASN00 updated maximum extent
Landsat ID: LE70200472014109ASN00 updated maximum extent

Stacking to extent:
        Upper Left: 77385.0,2187015.0
        Lower Right: 323115.0,1973385.0

Stacking images:
<--------------- 1 / 4
Stacking:
 ./LE70200472014061EDC00/LE70200472014061EDC00_stack_chris

Target extent: [77385.0, 2187015.0, 323115.0, 1973385.0]
Output size: x=8191, y=7121
Output projection:
```

```
                PROJCS["UTM Zone 16, Northern Hemisphere",GEOGCS["WGS 84",DATUM["WGS_1984",SPHEROID["WG

<--------------- 2 / 4
Stacking:
 ./LE70200472014077ASN00/LE70200472014077ASN00_stack_chris

Target extent: [77385.0, 2187015.0, 323115.0, 1973385.0]
Output size: x=8191, y=7121
Output projection:
                PROJCS["UTM Zone 16, Northern Hemisphere",GEOGCS["WGS 84",DATUM["WGS_1984",SPHEROID["WG

<--------------- 3 / 4
Stacking:
 ./LE70200472014093ASN00/LE70200472014093ASN00_stack_chris

Target extent: [77385.0, 2187015.0, 323115.0, 1973385.0]
Output size: x=8191, y=7121
Output projection:
                PROJCS["UTM Zone 16, Northern Hemisphere",GEOGCS["WGS 84",DATUM["WGS_1984",SPHEROID["WG

<--------------- 4 / 4
Stacking:
 ./LE70200472014109ASN00/LE70200472014109ASN00_stack_chris

Target extent: [77385.0, 2187015.0, 323115.0, 1973385.0]
Output size: x=8191, y=7121
Output projection:
                PROJCS["UTM Zone 16, Northern Hemisphere",GEOGCS["WGS 84",DATUM["WGS_1984",SPHEROID["WG



 --------------- REPORT ---------------


Stacking completed successfully
```

Now let's check the "landsat_stack.py" result:

```
In [86]: %%bash
         cd /projectnb/landsat/users/ceholden/2014_Landsat_Preprocess/5_Stack/

         for stack in $(find ./ -name '*stack_chris'); do
             gdalinfo $stack | grep "Size is"
             gdalinfo $stack | grep "Origin ="
             du -hs $stack
         done

Size is 8191, 7121
Origin = (77385.000000000000000,2187015.000000000000000)
891M        ./LE70200472014061EDC00/LE70200472014061EDC00_stack_chris
Size is 8191, 7121
Origin = (77385.000000000000000,2187015.000000000000000)
891M        ./LE70200472014077ASN00/LE70200472014077ASN00_stack_chris
Size is 8191, 7121
```

```
Origin = (77385.000000000000000,2187015.000000000000000)
891M        ./LE70200472014093ASN00/LE70200472014093ASN00_stack_chris
Size is 8191, 7121
Origin = (77385.000000000000000,2187015.000000000000000)
891M        ./LE70200472014109ASN00/LE70200472014109ASN00_stack_chris
```

0.6 6. Run Fmask using custom parameters on images

Depending on your scene location or tolerance for noise, you may wish to run Fmask with a more conservative threshold.

Since this example is in the Yucatan Penninsula, my guess is that the default Fmask cloud probability of 22.5 will miss a lot of small clouds. Furthermore, we have a relatively large amount of data to work with, so comission is not much of an issue.

To run Fmask on our cluster, you can run MATLAB as a single threaded "batch style" job using some command line arguments to MATLAB:

In [70]:
```
%%bash
module load gdal/1.10.0

cd /projectnb/landsat/users/ceholden/2014_Landsat_Preprocess/6_Fmask

# Define MATLAB single threaded "batch" runtime
ML="/usr/local/bin/matlab -nodisplay -nojvm -singleCompThread -r "

# Define MATLAB command to run - let's use 12.5 as cloud probabilty and dilate clouds and shad
CMD="addpath('/usr3/graduate/zhuzhe/Algorithms/Fmask');clr_pct=autoFmask(5,4,3,12.5);fprintf('

# Loop over all images running Fmask
here=$(pwd)

for img in $(find ./ -name 'L*' -type d); do
    cd $img

    # Let's not just run Fmask, but also capture clear percentage
    clear=$($ML $CMD | grep "CLEAR=")

    # Find Fmask
    fmask=$(find ./ -name '*Fmask')
    # Add clear percent to metadata
    gdal_edit.py -mo "$clear" $fmask

    cd $here
done

echo "Done!"
```

```
Done!
```

In [71]:
```
%%bash
cd /projectnb/landsat/users/ceholden/2014_Landsat_Preprocess/6_Fmask

for fmask in $(find ./ -name '*Fmask'); do
    echo $(basename $fmask)
    gdalinfo $fmask | grep "CLEAR"
done
```

```
LE70200472014061EDC00_MTLFmask
  CLEAR=36.869076
LE70200472014077ASN00_MTLFmask
  CLEAR=20.763808
LE70200472014093ASN00_MTLFmask
  CLEAR=36.600765
LE70200472014109ASN00_MTLFmask
  CLEAR=21.032762
```

Batch Fmask

Prior to the USGS providing surface reflectance results from LEDAPS, we had to run LEDAPS ourselves here at BU.

To help facilitate the preprocessing of Landsat data, I created a script that works on a directory full of Landsat image archives and intelligently submits "qsub" jobs to preprocess each archive. Features of the script include:

- Stop preprocessing routine and create error log file if image is L1G
- Delete TIF images after completed
- Extract Landsat archives and organize by Path-Row/LandsatID
- Perform LEDAPS and/or Fmask
- Customize Fmask parameters
- Set number of batch jobs running at one time on the cluster (useful since MATLAB runs out of licenses)

You can access this script by loading the module "batch_landsat":

```
In [88]: %%bash
         module load batch_landsat

         landsatPrepSubmit.sh -h
```

```
usage: /project/earth/packages/batch_landsat/bin/landsatPrepSubmit.sh [options] image_directory

    Author: Chris Holden (ceholden@bu.edu)

    Purpose:
    This script generates and manages the submissions of Landsat
    pre-processing (LEDAPS/Fmask) jobs to the Sun Grid Engine on BU's
    Katana cluster.

    Note: requires script "landsatPrepQsub.sh" to also be in your path

    Options:
        -h  help
        -m  maximum SGE jobs at one time (default 2)
        -n  base for job names (default - landsat; cannot begin with a #)
        -w  wait between qstat checks (default 60s)
        -d  delete TIF files?
        -c  cloud dilation parameter for FMASK (default 3)
        -s  shadow dilation parameter for FMASK (default 3)
        -p  cloud probability parameter for FMASK (default 22.5)
        -l  do LEDAPS? 1 - yes, 0 - no (default 1)
        -f  do FMask? 1 - yes, 0 - no (default 1)
        -x  do directory structure organization? (default 1)
            0 - no, just find tar.gz and move to their locations
            1 - yes, creates directory structure
```

```
        -g  check for L1G images and exit if found? (default 0)
        -e  send email? (default 1)
        -u  do unzipping? useful if data already extracted (default 1)
        -8  (BETA) use Qingsong's LC8 compatible version of LEDAPS (default 0)
        -2  Use Fmask 2.1 (default 0)

    Examples:
        Run LEDAPS and Fmask (3.2) using 10 jobs maximum and use custom Fmask
        options on the folder "images":
        > landsatPrepSubmit.sh -m 10 -n myjob -w 60 -d -c 5 -s 4 -p 12.5 images/

        Run ONLY Fmask (3.2) on the files processed above, but this time with a
        different Fmask cloud probability. Note the "-x" option:
        > landsatPrepSubmit.sh -m 10 -n myjob -c 5 -s 4 -p 22.5 images/P012-R031

        Run ONLY Fmask (3.2) on the files processed above, but with a different
        set of Fmask dilation values. Note that because we did not ask for the
        TIF files to be deleted in the last run, they still exist and we will
        not extract them ("-u 0").
        > landsatPrepSubmit.sh -m 10 -n myjob -c 3 -s 3 -p 22.5 images/P012-R031
```

While we could run LEDAPS with this tool, it is not longer required and our version of LEDAPS is not as recently updated as the one running at the USGS.

Instead, let's pretend we just downloaded new data and we'll unzip, organize, and then run Fmask with custom parameters for each image:

> landsatPrepSubmit.sh -m 2 -d -c 5 -s 4 -p 12.5 -l 0

This command will: - -m maximum of 2 qsub jobs - -d delete TIF files when done - -c 5 dilate clouds by 5 pixels - -s 4 dilate shadows by 4 pixels - -p 12.5 cloud probability = 12.5 - -l 0 do not do LEDAPS

```
In [89]: %%bash
         cd /projectnb/landsat/users/ceholden/2014_Landsat_Preprocess/7_BatchPrep

         module load batch_landsat

         landsatPrepSubmit.sh -m 2 -d -c 5 -s 4 -p 12.5 -l 0 -e 0 -g 1 ./
```

```
Scene directory:  /projectnb/landsat/users/ceholden/2014_Landsat_Preprocess/7_BatchPrep
Maximum jobs:  2
Job basename:  landsat
Time between qstat checks:  60
Cloud dilation:  5
Shadow dilation:  4
Cloud probability:  12.5
Remove unneccessary files?:  1
Do LEDAPS?:  0
Do Fmask?:  1
Check for L1G?:  1
Send email?:  0
Do unzip?:  1
LC8 LEDAPS:  0
Use FMask 2.1sav:  0
Found 4 Landsat archives
Wrote jobs out...
```

```
Jobs running: 0
<------------------------>
Your job 5773156 ("landsat.LE70200472014077-SC20140507171413") has been submitted
Submitted job #1
<------------------------>
Jobs submitted:  1
Jobs on SGE:  0
Jobs running:  0
Jobs waiting:  0
<------------------------>
Your job 5773157 ("landsat.LE70200472014093-SC20140507171414") has been submitted
Submitted job #2
<------------------------>
Jobs submitted:  2
Jobs on SGE:  1
Jobs running:  0
Jobs waiting:  1
<------------------------>
Jobs submitted:  2
Jobs on SGE:  2
Jobs running:  0
Jobs waiting:  2
<------------------------>
Jobs submitted:  2
Jobs on SGE:  2
Jobs running:  2
Jobs waiting:  0
<------------------------>
Jobs submitted:  2
Jobs on SGE:  2
Jobs running:  2
Jobs waiting:  0
<------------------------>
Your job 5773183 ("landsat.LE70200472014061-SC20140507171414") has been submitted
Submitted job #3
<------------------------>
Jobs submitted:  3
Jobs on SGE:  1
Jobs running:  1
Jobs waiting:  0
<------------------------>
Jobs submitted:  3
Jobs on SGE:  2
Jobs running:  1
Jobs waiting:  1
<------------------------>
Your job 5773191 ("landsat.LE70200472014109-SC20140507171414") has been submitted
Submitted job #4
<------------------------>
Jobs submitted:  4
Jobs on SGE:  1
Jobs running:  1
Jobs waiting:  0
All jobs submitted to SGE. Exiting.
```

Now that the job is done, let's look at our results:

```
In [94]: %%bash
         cd /projectnb/landsat/users/ceholden/2014_Landsat_Preprocess/7_BatchPrep

         # Remember, this script organizes Landsat images by path-row (e.g., P020-R047) AND Landsat ID
         ls -l P020-R047/

         n=$(find P020-R047 -name '*Fmask' | wc -l)

         echo ""
         echo "We have $n Fmask images"
```

```
total 128
drwxr-sr-x 3 ceholden landsat 32768 May  8 14:04 LE70200472014061-SC20140507171414
drwxr-sr-x 3 ceholden landsat 32768 May  8 14:01 LE70200472014077-SC20140507171413
drwxr-sr-x 3 ceholden landsat 32768 May  8 14:01 LE70200472014093-SC20140507171414
drwxr-sr-x 3 ceholden landsat 32768 May  8 14:06 LE70200472014109-SC20140507171414

We have 4 Fmask images
```

Our results are organized by the filename of the archives without the extension (i.e., no tar.gz).
To rename our files, we can write a simple Bash loop:

```
In [98]: %%bash
         cd /projectnb/landsat/users/ceholden/2014_Landsat_Preprocess/7_BatchPrep/P020-R047

         for d in $(find ./ -name 'L*-S*' -type d); do
             id=$(find $d -name '*MTL.txt' -exec basename {} \;)
             id=$(echo $id | awk -F '_' '{ print $1 }')

             mv $d $id
         done

         echo "Done!"

         ls -l ./
```

```
Done!
total 128
drwxr-sr-x 3 ceholden landsat 32768 May  8 14:04 LE70200472014061EDC00
drwxr-sr-x 3 ceholden landsat 32768 May  8 14:01 LE70200472014077ASN00
drwxr-sr-x 3 ceholden landsat 32768 May  8 14:01 LE70200472014093ASN00
drwxr-sr-x 3 ceholden landsat 32768 May  8 14:06 LE70200472014109ASN00
```

0.7 7. Create spatial subset using extent or polygon ROI

If you want to create a subset of your stacks, you can easily accomplish this using one or two of the GDAL command line utilities. Subsets are often very useful when you are rapidly iterating on your analysis and don't need to extend it to your entire study area, or if your study area is smaller than an entire Landsat image. In addition to just subsetting the size of your raster datasets, you can use GDAL to actually mask certain areas from your analysis.

Simple subsetting using extent

If you already have a bounding box defined (i.e., an extent listing upper left and lower right coordinates), you can use:

to very easily subset your data. You can either define your subset in units of pixels (-srcwin) or in units of actual coordinates (-projwin).

```
In [123]: %%bash
          gdal_translate --help
```

```
Usage: gdal_translate [--help-general] [--long-usage]
       [-ot {Byte/Int16/UInt16/UInt32/Int32/Float32/Float64/
             CInt16/CInt32/CFloat32/CFloat64}] [-strict]
       [-of format] [-b band] [-mask band] [-expand {gray|rgb|rgba}]
       [-outsize xsize[%] ysize[%]]
       [-unscale] [-scale [src_min src_max [dst_min dst_max]]]
       [-srcwin xoff yoff xsize ysize] [-projwin ulx uly lrx lry] [-epo] [-eco]
       [-a_srs srs_def] [-a_ullr ulx uly lrx lry] [-a_nodata value]
       [-gcp pixel line easting northing [elevation]]*
       [-mo "META-TAG=VALUE"]* [-q] [-sds]
       [-co "NAME=VALUE"]* [-stats]
       src_dataset dst_dataset
```

For example, let's say I want to create a subset somewhere in the center of my stacked images:

```
In [119]: %%bash
          cd /projectnb/landsat/users/ceholden/2014_Landsat_Preprocess/8_Subset

          # gdalinfo will list the center coordinate of our raster
          gdalinfo LE70200472014061EDC00/*stack | grep "Center"
```

```
Center      (  200400.000, 2079900.000) ( 89d50'31.44"W, 18d47'21.19"N)
```

Note that 200400.000, 2079900.000 is the center of a pixel closest to our image's center so, we'll need to shift a half pixel to avoid clipping in the middle of a pixel.

Let's make a 500x500 pixel raster centered on our image center:

```
In [133]: %%bash
          cd /projectnb/landsat/users/ceholden/2014_Landsat_Preprocess/8_Subset

          gdalinfo LE70200472014061EDC00/*stack | grep "Center"

          # Center pixel ULx: 200385
          # Center pixel ULy: 2079915

          # So if we resize by 250 pixels in each direction,

          ulx=$(expr 200385 - 30 \* 250)
          uly=$(expr 2079915 + 30 \* 250)
          lrx=$(expr 200385 + 30 \* 250)
          lry=$(expr 2079915 - 30 \* 250)

          echo ""
          echo "Our new extent:"
          extent="$ulx $uly $lrx $lry"
          echo $extent
```

```
        # Perform subsetting on all of our stacked images
        for stack in $(find ./ -name '*stack'); do
            id=$(basename $(dirname $stack))

            echo "Subsetting stack image for: $id"

            output=${stack}_subset.gtif

            gdal_translate -of GTiff -projwin $extent $stack $output

            echo ""
        done
```

```
Center      (  200400.000, 2079900.000) ( 89d50'31.44"W, 18d47'21.19"N)

Our new extent:
192885 2087415 207885 2072415
Subsetting stack image for: LE70200472014061EDC00
Input file size is 8121, 7101
Computed -srcwin 3810 3300 500 500 from projected window.
0...10...20...30...40...50...60...70...80...90...100 - done.

Subsetting stack image for: LE70200472014077ASN00
Input file size is 8121, 7101
Computed -srcwin 3810 3300 500 500 from projected window.
0...10...20...30...40...50...60...70...80...90...100 - done.

Subsetting stack image for: LE70200472014093ASN00
Input file size is 8121, 7101
Computed -srcwin 3810 3300 500 500 from projected window.
0...10...20...30...40...50...60...70...80...90...100 - done.

Subsetting stack image for: LE70200472014109ASN00
Input file size is 8121, 7101
Computed -srcwin 3810 3300 500 500 from projected window.
0...10...20...30...40...50...60...70...80...90...100 - done.
```

As you can see from ***gdal_translate***'s output, it took our "projwin" in projected coordinates (meters in our case, but it could just as well be used for geographic data in decimal degrees) and calculated the corresponding offsets and pixel size.

If you already knew a range of rows and columns from your raster, you could specify the window you want to extract using:

-srcwin xoff yoff xsize ysize

- xoff offset from 0 in columns
- yoff offset from 0 in rows
- xsize number of columns to include
- ysize number of rows to include

More complicated subsetting - polygon & masking outside of polygon

Let's suppose instead that you have some area of interest within your Landsat image that you'd like to analyze. In this scenario, the rest of the Landsat image would be a waste of disk space.

You could calculate the spatial extent needed to contain the polygon of interest and simply run ***gdal_translate*** with the upper left and lower right X and Y coordinates as shown above.

However, you could further limit your analysis by not only subsetting your raster images, but by masking out areas that you're not interested in. In order to do this, we will need to use

gdalwarp

gdalwarp is mostly used for going from one projection to another, but it can also be used with a polygon vector file to crop images or to mask images.

NOTE: ***gdalwarp***'s intended goal is to do reprojections. If we just want to clip our rasters, we will not be doing any reprojections. Unfortunately, even if we do not tell the program to reproject the data, it will recompute the pixel sizes and extent of our raster (see ***gdalwarp*** bug ticket https://trac.osgeo.org/gdal/ticket/3947 and ***gdal_translate*** enhancement ticket https://trac.osgeo.org/gdal/ticket/4875)

We can sidestep this potential issue by feeding ***gdalwarp*** the correct extent and pixel size.

First thing to do will be to use GDAL's sister library, OGR, to extract one polygon from a shapefile containing "early action" sites in Mexico. OGR is a very useful library and has many powerful command line utilities for working with vector data (see http://www.gdal.org/ogr/ogr_sql.html for information about OGR's SQL-like queries).

```
In [146]: %%bash
          module load gdal/1.10.0

          cd /projectnb/landsat/users/ceholden/2014_Landsat_Preprocess/8_Subset

          ogrinfo -al -geom=NO -sql 'SELECT * FROM mredd_aatr WHERE (ESTADO  LIKE "%Campeche")' mredd_a
```

```
INFO: Open of `mredd_aatr.shp'
      using driver `ESRI Shapefile' successful.

Layer name: mredd_aatr
Geometry: Polygon
Feature Count: 1
Extent: (-107.898611, 15.472142) - (-88.909351, 27.573289)
Layer SRS WKT:
GEOGCS["WGS 84",
    DATUM["WGS_1984",
        SPHEROID["WGS 84",6378137,298.257223563,
            AUTHORITY["EPSG","7030"]],
        TOWGS84[0,0,0,0,0,0,0],
        AUTHORITY["EPSG","6326"]],
    PRIMEM["Greenwich",0,
        AUTHORITY["EPSG","8901"]],
    UNIT["degree",0.01745329251994328,
        AUTHORITY["EPSG","9122"]],
    AUTHORITY["EPSG","4326"]]
ID: Real (11.0)
SITIO: String (150.0)
POLIGONO: String (50.0)
ESTADO: String (50.0)
OGRFeature(mredd_aatr):0
  ID (Real) = 1
  SITIO (String) = Sierra Pucc- Los Chenes
  POLIGONO (String) = -
  ESTADO (String) = Yucatán - Campeche
```

Now we can use ***ogr2ogr*** to extract this one action site into it's own shapefile. When we do this, we can also reproject our shapefile into the UTM zone we've been using with our Landsat data:

```
In [153]: %%bash
          module load gdal/1.10.0

          cd /projectnb/landsat/users/ceholden/2014_Landsat_Preprocess/8_Subset

          ogr2ogr -f "ESRI Shapefile" -t_srs EPSG:32616 \
              -sql "SELECT * FROM mredd_aatr WHERE (ESTADO  LIKE '%Campeche')" \
              mredd_yucatan.shp mredd_aatr.shp

          ogrinfo -al -geom=NO mredd_yucatan.shp

INFO: Open of `mredd_yucatan.shp'
      using driver `ESRI Shapefile' successful.

Layer name: mredd_yucatan
Geometry: Polygon
Feature Count: 1
Extent: (176449.259014, 2088249.065836) - (300170.064002, 2281342.856148)
Layer SRS WKT:
PROJCS["WGS_1984_UTM_Zone_16N",
    GEOGCS["GCS_WGS_1984",
        DATUM["WGS_1984",
            SPHEROID["WGS_84",6378137,298.257223563]],
        PRIMEM["Greenwich",0],
        UNIT["Degree",0.0174532925199433295]],
    PROJECTION["Transverse_Mercator"],
    PARAMETER["latitude_of_origin",0],
    PARAMETER["central_meridian",-87],
    PARAMETER["scale_factor",0.9996],
    PARAMETER["false_easting",500000],
    PARAMETER["false_northing",0],
    UNIT["Meter",1]]
ID: Real (11.0)
SITIO: String (150.0)
POLIGONO: String (50.0)
ESTADO: String (50.0)
OGRFeature(mredd_yucatan):0
  ID (Real) = 1
  SITIO (String) = Sierra Pucc- Los Chenes
  POLIGONO (String) = -
  ESTADO (String) = Yucatán - Campeche
```

With our polygon shapefile ready to go, we can calculate an output extent that clips to the polygon but preserves the spacing of the Landsat data. Once again, we'll be using a simple Bash function from Derek Watkin's GDAL/OGR cheat sheet (https://github.com/dwtkns/gdal-cheat-sheet) to easily parse ***gdalinfo*** and ***ogrinfo***'s outputs into extent strings:

```
In [211]: %%bash
          function gdal_extent() {
              if [ -z "$1" ]; then
                  echo "Missing arguments. Syntax:"
                  echo "  gdal_extent <input_raster>"
                  return
              fi
              EXTENT=$(gdalinfo $1 |\
```

```
        grep "Upper Left\|Lower Right" |\
        sed "s/Upper Left  //g;s/Lower Right //g;s/).*//g" |\
        tr "\n" " " |\
        sed 's/ *$//g' |\
        tr -d "[(,]")
    echo -n "$EXTENT"
}

function ogr_extent() {
    if [ -z "$1" ]; then
        echo "Missing arguments. Syntax:"
        echo "  ogr_extent <input_vector>"
        return
    fi
    EXTENT=$(ogrinfo -al -so $1 |\
        grep Extent |\
        sed 's/Extent: //g' |\
        sed 's/(//g' |\
        sed 's/)//g' |\
        sed 's/ - /, /g')
    EXTENT=`echo $EXTENT | awk -F ',' '{print $1 " " $4 " " $3 " " $2}'`
    echo -n "$EXTENT"
}

cd /projectnb/landsat/users/ceholden/2014_Landsat_Preprocess/8_Subset

img_ext=$(gdal_extent LE70200472014061EDC00/LE70200472014061EDC00_stack_chris)
shp_ext=$(ogr_extent mredd_yucatan.shp)

pix=$(gdalinfo LE70200472014061EDC00/LE70200472014061EDC00_stack_chris \
    | grep "Pixel Size" | sed "s/Pixel.*(//g;s/,/ /g;s/)//g")
pix_sz="$pix $pix"

echo "Extent of stacked images and extent of shapefile:"
echo $img_ext
echo $shp_ext

new_ext=""

for i in 1 2 3 4; do
    # Get the ith coordinate from sequence
    r=$(echo $img_ext | awk -v i=$i '{ print $i }')
    v=$(echo $shp_ext | awk -v i=$i '{ print $i }')
    pix=$(echo $pix_sz | awk -v i=$i '{ print $i }')

    # Quick snippit of Python
    ext=$(python -c "\\
        offset=int(($r - $v) / $pix); \
        print $r - offset * $pix\
        ")
    new_ext="$new_ext $ext"
done
```

```
        echo "Calculated new extent:"
        echo $new_ext

        # Now, unfortunately, gdalwarp wants us to specify xmin ymin xmax ymax
        # In this case, this corresponds to the upper left X, lower right Y, lower right X, and upper
        warp_ext=$(echo $new_ext | awk '{ print $1 " " $4 " " $3 " " $2 }')
        echo "gdalwarp extent:"
        echo $warp_ext

        # Perform the clip:

        for stack in $(find ./ -name '*stack'); do
            echo "Clipping $(basename $(dirname $stack))"

            output=${stack}_subset

            gdalwarp -of ENVI -co "INTERLEAVE=BIP" -te $warp_ext -tr 30 30 \
                -cutline mredd_yucatan.shp -cl mredd_yucatan -crop_to_cutline \
                -dstnodata -9999 -wm 2000 \
                $stack $output

        done
```

```
Extent of stacked images and extent of shapefile:
77385.000 2187015.000 323115.000 1973385.000
176449.259014 2281342.856148 300170.064002 2088249.065836
Calculated new extent:
176445.0 2281335.0 300195.0 2088225.0
gdalwarp extent:
176445.0 2088225.0 300195.0 2281335.0
Clipping LE70200472014061EDC00
Creating output file that is 4124P x 6436L.
Processing input file ./LE70200472014061EDC00/LE70200472014061EDC00_stack.
Using internal nodata values (eg. -9999) for image ./LE70200472014061EDC00/LE70200472014061EDC00_stack.
0...10...20...30...40...50...60...70...80...90...100 - done.
Clipping LE70200472014077ASN00
Creating output file that is 4124P x 6436L.
Processing input file ./LE70200472014077ASN00/LE70200472014077ASN00_stack.
Using internal nodata values (eg. -9999) for image ./LE70200472014077ASN00/LE70200472014077ASN00_stack.
0...10...20...30...40...50...60...70...80...90...100 - done.
Clipping LE70200472014093ASN00
Creating output file that is 4124P x 6436L.
Processing input file ./LE70200472014093ASN00/LE70200472014093ASN00_stack.
Using internal nodata values (eg. -9999) for image ./LE70200472014093ASN00/LE70200472014093ASN00_stack.
0...10...20...30...40...50...60...70...80...90...100 - done.
Clipping LE70200472014109ASN00
Creating output file that is 4124P x 6436L.
Processing input file ./LE70200472014109ASN00/LE70200472014109ASN00_stack.
Using internal nodata values (eg. -9999) for image ./LE70200472014109ASN00/LE70200472014109ASN00_stack.
0...10...20...30...40...50...60...70...80...90...100 - done.
```

In practice, I don't actually calculate the correct extent that aligns with my raster programatically - I just run the numbers myself. It's a lot easier that way.

What did we get?

Before

Lots of clouds!

After Well, at least it is smaller now

0.8 8. Create "preview" images to help filter through preprocessed imagery

By now you are probably thinking - how am I to actually look at these hundreds of images?

With the right data policies (i.e., free), hardware, and software tools, the challenge of using Landsat data is no longer in getting the datasets together; the challenge is looking at all of your source data and results.

One easy way of assessing whether or not an individual image might be useful or not is to create a "browse" or "preview" image of each image in your time series. By "browse" or "preview", I mean a 3 color composite that is viewable using any number of image browsing softwares.

GDAL tools

Creation of a "browse" image is possible by combining ***gdal_merge.py*** with ***gdal_translate***. ***gdal_merge.py*** can be used to extract band combinations from your stack images (e.g., my favorite 5, 4, and 3) into a new 3 color image. ***gdal_translate*** can then be used to resize the image to a smaller resolution and scale the image numbers to 0-255.

I won't discuss this workflow here because it is very limited for time series applications. For one, when you scale each image in ***gdal_translate***, the scaling is very simplistic and will simply linearly scale the minimum and maximum of each band to fit between 0 - 255. If we want to compare images through time, we would need a constant stretch. Another limitation is that this workflow will not help us identify how Fmask is performing.

If curious, a good description of this workflow is available here:

http://davidvhill.com/article/creating-browse-images-from-landsat-data

Custom tools

To help fulfill the limitations described above, I created my own specific tool that has many possible customizations:

gen_preview.py

```
In [213]: %%bash
          module load gdal/1.10.0
          module load batch_landsat

          gen_preview.py --help
```

```
Generate preview image

Usage:
    gen_preview.py [options] (--linear_pct <pct> | --histeq | --manual <minmax>)
        <input> <output>

Options:
    -b --bands <bands>              Bands for output image [default: 3 2 1]
    --mask <mask>                   Mask band [default: 8]
    --maskval <value>...            Mask band value [default: 2 3 4]
    --maskcol <r, g, b>             Mask color [default: 0, 0, 0]
    --ndv <value>                   No data value [default: 255]
    --threshold <percent>           Min unmasked data for output [default: 0]
    --srcwin <x y xsize ysize>      Window (in pixels) to subset
    --projwin <ulx uly lrx lry>     Window (in projected coordinates) to subset
    --resize_pct <pct>              Image resize percent [default: 100]
    --resize_method <method>        Image resize method [default: antialias]
    --format <format>               Output format [default: JPEG]
    -v --verbose                    Print (verbose) debugging messages
```

```
  -q --quiet                    Do not print except for warnings/errors
  -h --help                     Show help
```

With this program, we can not only create many "browse" images with a consistent stretch, but we can:

- –mask
  specify Fmask band in our stack images
- –maskval
  specify which Fmask values we want to ignore
- –threshold
  specify minimum amount of unmasked pixels required to output preview
- –resize_pct
  resize image by a percentage

For example:

```
In [217]: %%bash
          module load gdal/1.10.0
          module load batch_landsat

          cd /projectnb/landsat/users/ceholden/2014_Landsat_Preprocess/9_PreviewImage

          for stack in $(find ./ -name '*stack'); do
              id=$(basename $(dirname $stack))
              echo "Making preview image for $id"

              gen_preview.py --format PNG -b "5 4 3" --resize_pct 25 --manual "0 8000" $stack ${id}_pre

          done

          echo "Done!"
```

```
Making preview image for LE70200472014061EDC00
Making preview image for LE70200472014077ASN00
Making preview image for LE70200472014093ASN00
Making preview image for LE70200472014109ASN00
Done!
```

Results:

And if you really want to get creative...

```
In [219]: import io
          import base64
          from IPython.display import HTML

          video = io.open('resources/movie.mp4', 'r+b').read()
          encoded = base64.b64encode(video)
          HTML(data='''<video alt="test" controls>
                          <source src="data:video/mp4;base64,{0}" type="video/mp4" />
                       </video>'''.format(encoded.decode('ascii')))
```

```
Out[219]: <IPython.core.display.HTML at 0x1461cd0>
```

0.9 9. Summarize and document your dataset

By now, you've probably seen how useful loops combined with grep, sed, awk, etc. can be.

The following are some examples of how you might summarize your dataset by querying the metadata provided with your Landsat images:

ACCA Cloud Cover

```
In [228]: %%bash
          cd /projectnb/landsat/users/ceholden/2014_Landsat_Preprocess/5_Stack

          for mtl in $(find ./ -name '*MTL.txt'); do
              id=$(basename $(dirname $mtl))
              echo $id - $(grep "CLOUD_COVER" $mtl | tr -d ' ' | awk -F '=' '{ print $2 }')
          done
```

```
LE70200472014061EDC00 - 21.00
LE70200472014077ASN00 - 59.00
LE70200472014093ASN00 - 17.00
LE70200472014109ASN00 - 40.00
```

Sun Elevation, Sun Azimuth

```
In [227]: %%bash
          cd /projectnb/landsat/users/ceholden/2014_Landsat_Preprocess/5_Stack

          for mtl in $(find ./ -name '*MTL.txt'); do
              id=$(basename $(dirname $mtl))
              az=$(grep "SUN_AZIMUTH" $mtl | tr -d ' ' | awk -F '=' '{ print $2 }')
              el=$(grep "SUN_ELEVATION" $mtl | tr -d ' ' | awk -F '=' '{ print $2 }')

              echo "$id, $az, $el"
          done
```

```
LE70200472014061EDC00, 130.59617147, 52.22814557
LE70200472014077ASN00, 123.58852009, 57.11885778
LE70200472014093ASN00, 114.75958139, 61.65047059
LE70200472014109ASN00, 103.95380278, 65.12459343
```

Number of observations per pixel

If you want a map of how many observations there are per pixel, you could loop through all images in your dataset and count the number of times each pixel is not masked by Fmask.

In order to do this, I wrote the following program:

stack_nobs.py

```
In [236]: %%bash
          module load batch_landsat

          stack_nobs.py --help

          cd /projectnb/landsat/users/ceholden/2014_Landsat_Preprocess/5_Stack

          # Create our number of observations stack
          stack_nobs.py --dname 'L*' --format GTiff ./ stack_nobs.gtif 2 3 4
```

```
        # Make it smaller!
        gdal_translate -scale 0 4 0 255 -outsize 25% 25% -of PNG stack_nobs.gtif stack_nobs.png

        # Create class height file for gdaldem
        echo 0, 215, 25, 28 > colors.txt
        echo 1, 253, 174, 97 >> colors.txt
        echo 2, 255, 255, 191 >> colors.txt
        echo 3, 166, 217, 106 >> colors.txt
        echo 4, 26, 150, 65 >> colors.txt

        gdaldem color-relief stack_nobs.gtif colors.txt stack_nobs_colormap.gtif

        gdal_translate -outsize 25% 25% -of PNG stack_nobs_colormap.gtif stack_nobs_colormap.png

Number of Observations in Stacks

Usage:
    stack_nobs.py [options] <location> <output> [<maskvalues>...]

Options:
    -n --name <name>        Pattern of each stack file [default: *stack]
    -d --dname <dname>      Pattern for each stack directory [default: LND*]
    -m --mask <band>        Mask band [default: 8]
    -n --ndv <ndv>          No data value [default: 255]
    -f --format <format>    Output file format [default: GTiff]
    -v --debug              Show (verbose) debugging messages
    -h --help               Show help
Finished image 1/4
Finished image 2/4
Finished image 3/4
Finished image 4/4
Input file size is 8121, 7101
0...10...20...30...40...50...60...70...80...90...100 - done.
0...10...20...30...40...50...60...70...80...90...100 - done.
Input file size is 8121, 7101
0...10...20...30...40...50...60...70...80...90...100 - done.
```

Visualize time series

module load CCDCTools/_beta

Example: