Reprocessing Instructions

P310 FT2 Reprocessing

Location of the scripts and of the reprocessing directory:

```
/nfs/farm/g/glast/u38/Reprocess-tasks/P310-FT2
```

The reprocessing is usually split into three steps (expand for details):

I have a file called "preparelist.py" that helps querying the data catalog and saves the list of runs in an appropriate text file, which will be used in the next step. The file (**/nfs/farm/g/glast/u38/Reprocess-tasks/P310-FT2/preparelist.py**) looks like this:

```
#!/usr/bin/env
python

import sys,
os

def run(cmd,
test=False):


'''

    Simple interface to excecure a system
call


'''

    print
cmd

    if not test: os.system
(cmd)


pass



def extractRunNumber(out_file_list,
out_run_list):


'''

    Extract run number from the name of the
file


'''

    runs=
[]

    out_run_list_file=file
(out_run_list,'w')

    for l in file(out_file_list,'r').readlines():

        run=l.split('_v')[0].split('_r0')
[-1]

        runs.append
```

```
(run)

        out_run_list_file.write
(run+'\n')


pass

    return
runs



################################################################

# REPROCESSED
310:

# MINIMUM RUN NUMEBR TO BE
REPROCESSED

RunMin='239557414'

# MAXIMUM RUN NUMEBR TO BE
REPROCESSED

RunMax='604845703'

RunMax='625881605' #2020-11-01 00:00:
00

################################################################

out_file_list = 'FileList_%(RunMin)s_%(RunMax)s.txt' % locals()

out_run_list  = 'RunsList_%(RunMin)s_%(RunMax)s.txt' % locals()



p310_file_list = 'P310_FileList_%(RunMin)s_%(RunMax)s.txt' % locals()

p310_run_list  = 'P310_RunsList_%(RunMin)s_%(RunMax)s.txt' % locals()



p310_remaining_run_list  = 'P310_Remaining_RunsList_%(RunMin)s_%(RunMax)s.txt' % locals()
\
################################################################

# This is the list of file in the
datacatalog:

cmd="/afs/slac.stanford.edu/u/gl/glast/datacat/prod/datacat find --mode PROD --site SLAC_XROOT --group FT2 --
filter 'RunMin >=%(RunMin)s && RunMin<=%(RunMax)s' --sort nRun --show-non-ok-locations /Da\
ta/Flight/Level1/LPA > %(out_file_list)s" % locals()

# --display 'RunMin' >
$out_list

run(cmd,
test=False)

to_process=extractRunNumber(out_file_list,
out_run_list)

print 'split -l25 ../%(out_run_list)s -a 3' % locals()



################################################################
```

```
# This is the list of files that are already
reprocessed:

cmd="/afs/slac.stanford.edu/u/gl/glast/datacat/prod/datacat find --mode PROD --site SLAC_XROOT --group FT2 --
filter 'RunMin >=%(RunMin)s && RunMin<=%(RunMax)s' --sort nRun --s\
how-non-ok-locations /Data/Flight/Reprocess/P310 > %(p310_file_list)s" % locals()

run(cmd,
test=False)

processed=extractRunNumber(p310_file_list,
p310_run_list)

#wc
$out_list



remaining=
[]

out_run_list_file=file
(p310_remaining_run_list,'w')
\


for x in
to_process:

    if not x in
processed:

        if int(x)>240729801:# We skip the first two
runs

            remaining.append
(x)

            out_run_list_file.write
(x+'\n')


pass


pass

out_run_list_file.close()

print 'To procrss:%d, processed:%d, remaining: %d' % (len(to_process),len(processed),len
(remaining))
print 'split -l25 ../%(p310_remaining_run_list)s -a 3' % locals()
```

It basically does two calls to the data catalog. The first retrieves the list of runs to reprocess, and the second retrieves the list of runs already reprocessed. Files are created to keep track of these files, and the names of the files contain the minimum and the maximum run number.
As the last print statement suggests, I split the list of files into files containing 25 runs. First, I create two directories, and I then cd in the todo one. For example:

```
mkdir todo-2020-11/
mkdir done-2020-11/
cd todo-2020-11/
```

Then, the command I use is simply:

```
split -l25 ../P310_Remaining_RunsList_239557414_625881605 -a 3
```

This will create a series of files containing 25 run each.

There is a simple file (**submitter-prod-2020-11**) containing the sequence of bash commands I submit:

```
#!/bin
/bash



delay=300



while true ;
do

        rf=$(ls todo-2020-11/* | head
-1)

        echo
$rf

        for run in $(<$rf) ;
do

            /afs/slac.stanford.edu/u/gl/glast/pipeline-II/prod/pipeline -m PROD createStream --stream $run --
define RUNID=r0$run P310-FT2

done

        mv $rf done-2020-
11/.


date

        sleep
$delay

done
```

Note that this has to be modified every time I create a backfill (todo-2020-11/ and done-2020-11/).

What this does is read one file in the **todo-2020-11** directory, and submit N streams of the P310-FT2 task. Each task has the input run (RUNID) as an argument. In our case, N=25. Then it will move the input file in the **done-2020-11** directory. Then it sleeps for 5 minutes.
This is very similar to reprocessing runs. Check the BAD time periods page which lists runs with bad time intervals (BTIs). Then, create a list (called **BTI-list.txt** here) with the runs in the reprocessing that contain BTIs. As before, I split the file into subfiles with 25 runs:

```
mkdir todo-bti/
mkdir done-bti/
cd todo-bti/
split -l25 ../BTI-list.txt -a 3
```

and then I run the following script (**submitter-prod-bti**)

```
#!/bin
/bash



delay=600



while true ;
do

        rf=$(ls todo-bti/* | head
-1)

        echo
$rf

        for run in $(<$rf) ;
do

            /afs/slac.stanford.edu/u/gl/glast/pipeline-II/prod/pipeline -m PROD createStream --stream $run --
define RUNID=r0$run flagFT2-P310

done

        mv $rf done-
bti/.


date

        rf=$(ls todo-bti/* | head
-1)

        echo
$rf

        for run in $(<$rf) ;
do

            /afs/slac.stanford.edu/u/gl/glast/pipeline-II/prod/pipeline -m PROD createStream --stream $run --
define RUNID=r0$run flagFT2-P310

done

        mv $rf done-
bti/.


date



        sleep
$delay

done
```

which submits the flagFT2-P310 task. Note that here the script submits 50 files and sleeps for 10 minutes.