Paper 065-30

Some UNIX Scripts Tips for SAS Coding

Jianming He, Solucient LLC, Berkeley Heights, NJ
Yanhong Huang, UMDNJ, Newark, NJ

ABSTRACT

This paper focuses on using UNIX script for SAS jobs. The authors give tips on handling tasks such as converting SAS log file back to source code if, for some reasons, the source code is unavailable; using UNIX shell script to schedule tasks or email the programmer about the job status; and overcoming the limitation of space or time, etc. With the help of UNIX scripts, SAS programmer gains much more capabilities.

INTRODUCTION

UNIX SAS provides unique flexibility that Windows SAS does not. We might be facing a problem with space, delivery time, UNIX system database availability, etc. in our daily work. UNIX system provides additional tools to help us to trade space for speed, or vice versa. We can also submit our job at a time when the system is not too busy. This paper will show some techniques, using both UNIX scripts and SAS coding, which are rarely mentioned in discussions.

TECHNIQUES

1.To Convert SAS Log Into SAS Source Code

Some consulting companies only disclose the SAS log files to their clients after the project is finished for the reasons of intellectual property. UNIX SAS beginners may accidentally destroy their source code by using UNIX command 'ci' or 'rm'. If the damage is done at the same day as the file is created, the system may not be able to be backed up. Programmers editing their scripts using FTP tools may also find problems like losing or overwriting source codes during transporting.

Fortunately we still have the log file for most cases. However, it is usually an unpleasant experience manually editing of the log file if the source file is lost. UNIX can help us to solve this problem efficiently. In the following discussion, we use a simplified workable script to make the underlying concepts easier.

The following code shows how to recover the SAS source code 'run.sas' based on SAS log file 'run.log'.

```
sed '/^[^[0-9]/d;/SAS/d;/^$/d;/DATA?:?:?/d; /\*[0-9]/d' run.log
|cut -c6-256 | tr ! ' ' |sed '/^[^ ]/d'> run.sas
```

When a SAS code is running, it lists the SAS institute license information and system time and date as a heading. It keeps the original codes and adds the line number in front of them. If the length of line exceeds the length limit, it is broken into multiple lines starting with an exclamation mark following the same line number. We can see the execution result for each statement. The log files also show the results for some procedures. For instance, PROC DATASETS lists its results in the log file starting with numbers as well as with a different set of sequences. In the log file, we observe a lot of empty lines also. In the UNIX scripts we need to address all those issues.

Caveats:

- Never set system options to NOSOURCE. Otherwise, the source codes won't be present in the log file.
- SAS automatically masks out the sensitive information in the log file. For instance, relational databases' pass word etc. We need to change these manually.
- SAS log does not keep data records followed by cards, datalines or datalines4 statement. We need to add this raw data by a different channel.
- There might also be certain exceptions not covered by the above codes. However, with its help we do not need a lot of editing work.

2. To trade speed for space

If we have a space problem in handling large datasets, especially flat files, UNIX is a great help. We may use the following code to break the flat file into smaller ones so that they can be compressed to save space. When we need further steps, we may unzip one file, do the required work, and then re-zip it.

For instance, the file “newfile” is broken into a couple of file names starting with ‘new*’. Each file is 200,000 lines long.

```
split newfile -200000 new
ls new* >a
gzip  infile new*
```

The next SAS code will efficiently save space.

```
data filename;
       infile 'a' ;
       informat filename $6.;
       format filename $6.;
       input filename $;
run;

proc sql;
       select distinct filename into : file separated by ' ' from filename;
       select count(distinct filename) into: ct from filename;
Quit;

%macro countpx;

%do i=1 %to &ct;
       %let file&i=%scan(&file, &i);
       x gunzip "&&file&i";

       data ppx&i ;
              %let _EFIERR_ = 0;
               infile "&&file&i" delimiter = '|' MISSOVER DSD
               lrecl=32767 firstobs=1 ;
                informat px $5. ;
                format   px $5.;
                     input  Patientid
                            pxnum
                            px $
                            ;
                     if _ERROR_ then call symput('_EFIERR_',1);
        run;

       proc sql;
              create table &&file&i as select distinct px, count(*) as sum from
       ppx&i group by px order by px;

       proc append data=&&file&i base=final force;
        run;

       x gzip "&&file&i";

        proc delete data=ppx&I;
       run;

%end;

%mend countpx;

%countpx;

proc sql;
       select distinct px, sum(sum) as sum from final group  by px;
```

The plan is to have a dataset with the entire broken out file names and then do something for each dataset. In the final step we do the aggregation.

Caveats:

- The trick to save space is that we always work on much smaller datasets and aggregate their result later on. This solution is fine if the statistics are additive. If we want to compute mean or median, then the solution does not work. Therefore, if the final report needs to report via a strafed variable, we may break the file down by that variable.

- Also, if we have a large file with ‘X’ bytes, we need at least another ‘X’ bytes to break down this big file. The space saved is the temporary working space needed to handle the large file because we are reading a much smaller data set. It is always good practice to keep the permanent datasets at a moderate size and compress them.
- There is another solution to create several SAS datasets by reading the big file separately. It needs more working space.

3. To trade space / memory usage for speed

When the job needs to be done immediately, we may submit multiple SAS jobs at the same time. An example is a group of databases broken down by different quarters where we want a count for the whole year. Usually we do the data pull sequentially.

```
%macro pullall(db);

libname  pdb informix database=&db server=&server_anet;

proc SQL;
      create table count&db as
            select distinct px, count(*) as count
            from pdb.ppx
            group by px;

%mend pullall;
%pullall(dbq1);
%pullall(dbq2);
%pullall(dbq3);
%pullall(dbq4);

data all;
      set countdbq1 countdbq2 countdbq3 countdbq4;

Proc sql;
Select px, sum(sum) as sum from all group by px;
```

However, in order to save time we may create four different SAS sessions for each different quarter. For instance, q3.sas is something like:

```
%macro pullall(db);

libname  pdb informix database=&db server=&server_anet;
proc SQL;
        create table count&db as
        select distinct px, count(*) as count
        from pdb.ppx
        group by px;
%mend pullall;
%pullall(dbq3);
```

We can then evoke UNIX shell script to submit q1.sas to q4.sas like

```
sas q1.sas &
sas q2.sas &
sas q3.sas &
sas q4.sas &
```

Caveat:

- This approach gains speed in processing by consuming a large amount of space and system memories. Ted Conway investigated parallel algorithms in his 2003 SUGI paper. His conclusion is that if we have excess processor capacity and the data is organized, we can always reduce the running time. The traffic to and from relational databases and UNIX systems are the primary factors determining if this solution works. Our experience is to make sure that the maximum number of concurrent processes are in the system. The risk is that you might compete with yourself for system resources. In certain database settings, there might also be a limit to the maximum number of sessions for relational database access.
- Multiple sessions also require additional temporary working space.

4. To automate a SAS job

A nice feature of UNIX is that it provides programmers with the flexibility of scheduling their job in the system, emailing the result and log file, and reporting the running status. The following code can be a standard UNIX shell script for those functions.

```
#!/bin/ksh
export PATH="$PATH/opt/sas8"
# Initializing variables
date1=`date +%m/%d/%y`

# Remove Previous output files
# Starting code execution
echo 'Starting to run the sas Job' > sas_job.log
echo Start Date: $date1 >> sas_job.log
sas run.sas   -noterminal
echo 'Finished running the sas Job' >> sas_job.log
date2=`date +%m/%d/%y`
echo End Date: $date2 >> sas_job.log

# Sending Log File via email
uuencode run.log run.log | mailx -s "LOG FILES" jhe@solucient.com

# Sending MS files via email
(uuencode stat.xls) | mailx -s "EXCEL FILES" jhe@solucient.com

exit 0
```

CONCLUSION

Compared to Window's SAS, UNIX SAS has additional toolset to help the SAS programmer survive. A good understanding of UNIX systems will tremendously enhance job efficiency.

REFERENCES

SAS institute Inc. (1999), *SAS online Document©,* Cary, NC: SAS institute Inc.

Daniel Gilly (1998), *UNIX in a Nutshell,* O'Reilly & Associates,

Ted Conway (2003), *Parallel Processing on the Cheap: Using Unix Pipes to Run SAS® Programs in Parallel*, Proceedings of the Twenty-Eighth Annual SAS Users Group International Conference, Cary, NC: SAS Institute Inc.

CONTACTS

Your comments and questions are valued and encouraged. Contact the author at:

Jianming He
Solucient LLC
One Connell Drive, Suite 1000
Berkeley Heights, NJ 07922
jhe@solucient.com
(734) 669-7835

Yanhong Huang, Ph.D.
UMDNJ
Newark,NJ 07101
huangya@umdnj.edu