SAS® GLOBAL FORUM 2018

USERS PROGRAM

Using UNIX Shell Scripting to Enhance Your SAS Programming Experience

By James Curley

April 8 – 11 | Denver, CO

#SASGF

Using UNIX Shell Scripting to Enhance Your SAS Programming Experience

By James Curley

ELIASSEN GROUP®
Biometrics & Data Solutions

ABSTRACT

This series will address three different approaches to using a combination of UNIX shell-scripting and SAS® programming to dramatically increase programmer productivity by automating repetitive, time-consuming tasks.

Program One embeds an entire SAS® program inside a UNIX shell script, feeds it an input file of source and target data locations, and then writes the SAS® copy program out to each directory with dynamically updated LIBNAME statements. This approach turned a 25 hour job into mere minutes.

Program Two of the series reviews another deploy shell script that creates its own input file and determines whether a standalone SAS® program should to be deployed to each directory. This approach turned an 8 hour job into just a couple minutes.

Program Three consists of a smaller shell script that dynamically creates SAS® code depending on the contents of the directory in which the program is executed.

At the end of these three segments, you will have a better understanding of how to dramatically increase the productivity of your SAS® programs by integrating UNIX shell-scripting into your SAS® programming to automate repetitive, time-consuming tasks. None of these programs requires any input from the user once started, so the only programming required is the few minutes it takes to set up the programs.

PROGRAM 1 - deployDataCopy.ksh

The first program uses a UNIX shell script to remove the human element from the task of copying a simple SAS program into 500 directories, updating the LIBNAME statements, executing the program and checking the logs for errors. A UNIX shell script is the perfect delivery system to do all of that work in a fraction of the time.

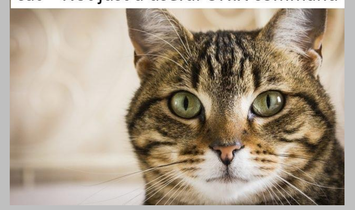
Here is a sample input file, trimmed down to 3 lines for the purposes of this illustration. (We typically have larger files to facilitate the loading of data for hundreds of studies at once). Each line consists of two parts: the protocol name and the path to the source datasets, delimited by a caret:

```
X3351001^/Volumes/app/data/prod/X335/nda1/X3351001/data
X3351002^/Volumes/app/data/prod/X335/nda_final/X3351002/data
X3351007^/Volumes/app/data/prod/X335/csr1/X3351007/data
```

To write out the SAS program the UNIX cat command is used to create a file in the program directory called DataCopy.sas. The cat command will write everything in this program between the two instances of the word “FINAL”.

```
cat > DataCopy.sas << FINAL
/* Put the entire SAS program here */
FINAL
```

cat – Not just a useful UNIX command



Using UNIX Shell Scripting to Enhance Your SAS Programming Experience

By James Curley

ELIASSEN GROUP®
Biometrics & Data Solutions

PROGRAM 1 - CONTINUED

The key advantage to embedding the SAS program inside the UNIX shell script is that the UNIX variables can be referenced in the SAS® program and will resolve to their UNIX variable values when the SAS program is written out to a file:

```
/** Set the location for source & target  **/
%let inpath =${s_path};
%let outpath=${d_path};
/** Set the dataset flags **/
%let demog  = ${de_flag};
%let drug   = ${dg_flag};
%let random = ${ra_flag};
```

As a result, the variable names as they appear in the shell script (above) will be the resolved variable values in the written SAS program (below):

```
/** Set the location for source & target  **/
%let inpath =/Volumes/app/data/prod/X335/nda1/X3351001/data;
%let outpath=/Volumes/app/data/prod/prjX335/pbrer2017/X3351001/data;
/** Set the dataset flags **/
%let demog  = YES;
%let drug   = YES;
%let random = YES;
```

PROGRAM 1 - CONCLUSION

So there you have it: A SAS program embedded in a UNIX shell script, deployed across any number of directories, executed, and then checked for errors. The hours this one utility has saved my team is almost inconceivable. What makes this combination more functional than running either a shell script or SAS program separately is that embedding the SAS program inside the UNIX shell script allows you the use of UNIX variables in the SAS program that will resolve to their UNIX variable values when the SAS program is stored prior to execution. Being able to dynamically update the SAS program with each iteration of the input file loop thus makes this combination especially useful.

The time savings is very easy to calculate for this illustration. Performing the tasks manually takes 25 to 40 hours of work. To set up and run this shell script takes 15 minutes. The program may take an hour to run, but the programmer need only start the program and move on to other tasks.

In program 2 of this series I will show how to deploy a standalone SAS program across the same 500 protocols; however, in this example, the program will create its own input file and then determine on a protocol-by-protocol basis if it needs to be deployed or not.

Using UNIX Shell Scripting to Enhance Your SAS Programming Experience

By James Curley

ELIASSEN GROUP®
Biometrics & Data Solutions

PROGRAM 2 - deployTreatments.ksh

Program two also uses a UNIX shell script to remove the human element from the task of copying a simple SAS program into 500 directories, executing the program and checking the logs for errors. Since the program resides in the same directory as the dataset it is analyzing, there are no LIBNAME statements to update. This means the program doesn't have to be embedded in the UNIX shell script, and can instead be deployed as a standalone program using only the shell script to quickly deliver it where it needs to be.

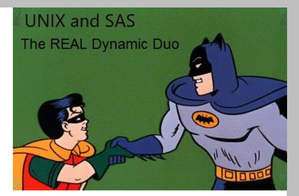
This section of code creates an input file of all the treatment datasets in the submission. This file is then piped into a grep that removes all the symlinks before writing it out to the deploy1.tmp temporary file. The sed statement will turn any number of spaces into a single space and allow us to use a space delimiter when the file is read a little further down in the code:

```
ls -l /Volumes/app/data/prod/prj${proj}/${sub}/*/data/treatment.sas7bdat |
grep ^- > ./deploy1.tmp


sed 's/[ ]\{1,\}/ /g' ./deploy1.tmp > ./deploy.lst
```

The program will now use the cat command and while loop to process the file one line at a time until reaching the end. The first variable assignment is going to remove the entire path and file name from the ls-l and store that value in ${line2}. Next, the protocol name will be cut from the ${line2} and assigned to ${prot}. Lastly, the dirname statement will remove the filename from ${line2}, leaving just the data directory path assigned to ${path}.

PROGRAM 2 – CONTINUED

```
cat ./deploy.lst |
  while read line
  do

  line2=`echo ${line} | cut -d' ' -f9`
  prot=`echo ${line2} | cut -d'/' -f8`
  path=$(dirname ${line2})
```



Next the program will determine if the Listing from the previous deployment is still "current". It would be wasteful to deploy a fresh program in each study where the underlying data hasn't even changed. This is accomplished by doing an ls -1 –t into a temporary file, which will produce a reverse chronological listing of the directory contents. Reading this list will allow us to set a flag telling us whether the listing is out-of-date and only regenerate the listing when the data has actually changed:

Using UNIX Shell Scripting to Enhance Your SAS Programming Experience

By James Curley

ELIASSEN GROUP®
Biometrics & Data Solutions

PROGRAM 2 - CONTINUED

```
 ls -1 -t > ./dir_lst.txt

  flag=no
  cat ./dir_lst.txt |
    while read line1
    do

    if [[ ${flag} = "yes" ]]; then
      continue
    elif [[ ${line1} = "treatment.sas7bdat" ]]; then
      echo -e "\n${file_name} is either out of date or does not exist....deploying"

      #  Check for and remove any existing log, listing and SAS® files.
      [[ -e ./${file_name}.lst ]] && rm ./${file_name}.lst
      [[ -e ./${file_name}.log ]] && rm ./${file_name}.log
      [[ -e ./${file_name}.sas ]] && rm ./${file_name}.sas
      #  Copy the program to the target directory and execute SAS® program.
      cp ${src_file} ${file_name}.sas

      echo -e "Running ${file_name}.sas for protocol ${prot}"
      sas94 -log ./${file_name}.log -print ./${file_name}.lst -nodms
./${file_name}.sas
      flag=yes
    elif [[ ${line1} = "${file_name}.lst" ]]; then
      echo -e "\n${file_name} listing is current for protocol ${prot}"
      flag=yes
    fi
    done
done
```

PROGRAM 2 – CONCLUSION

As you can see, it is just as simple to deploy a standalone SAS® program using a UNIX shell script as it is to embed one in the program. The two advantages to using the shell script are:

- Programmer setup time is reduced since the utility creates its own input file.
- Total run-time is reduced since the program only regenerates listings which are out-of-date.

The time savings is very easy to calculate for this illustration: Performing the tasks manually takes 8 hours of work versus a 15 second setup time for the shell script. The program may take an hour to run, but the programmer need only start the program and move on to other tasks.

Using UNIX Shell Scripting to Enhance Your SAS Programming Experience

By James Curley

ELIASSEN GROUP®
Biometrics & Data Solutions

PROGRAM 3 - unpack_xpt.ksh

Clinical trial data doesn't always come in SAS® datasets, it can come in a wide range of forms from delimited text files, to excel spreadsheets to SAS® export files. In all of these cases, these raw data files need to be turned into SAS® datasets. This program will dynamically write out the LIBNAME statements for each export file in the directory so the SAS® program can iterate through them and unpack each export file.

This line of code creates an input file of all xpt files in the current directory:

```
# Create the input list of all export files
ls -1 ./*.xpt > ./setup.lst
```

The program is to iterate through the setup file it just created and write out to a temp file, a LIBNAME statement for each xpt file on the list. The loop counter will iterate 1 more time than we need so the ((x=x-1)) line will bring the counter back to the correct number of LIBNAME statements:

```
cat ./setup.lst |
  while read line
  do
   echo -e "libname xpt${x} xport \"${line}\";" >> ./temp.txt
    ((x=x+1))
done
((x=x-1))
```

UNIX and SAS working together

PROGRAM 3 – CONCLUSION

Here we see how UNIX shell scripts can write customized SAS® programs with information gathered on its own. This program takes just a few seconds to get setup and run, saving the programmer the time of manually writing out the LIBNAME statements for each file in the directory.

This three parts of this series demonstrated just a few different ways that UNIX shell scripting and SAS® can be used together to dramatically increase the productivity of SAS® programs by automating time-consuming, repetitive tasks.

CONTACT INFO

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

Name: Jim Curley

E-mail: JCurley@eliassen.com

http://www.jimcurley.net/sas-global-forum-2018

SAS® GLOBAL FORUM 2018

April 8 – 11 | Denver, CO
Colorado Convention Center

#SASGF