Paper 5099-2020

U-can-NIX Leaving Your SAS® Session to Do That

Lorelle Benetti
Division of Biomedical Statistics and Informatics, Mayo Clinic

ABSTRACT

SAS® programmers often play the role of "electronic storage space steward," requiring them to monitor the electronic space their programs and data sets take up as well as the electronic space their colleagues utilize. With tight deadlines, it's easy for programmers to write programs or create data sets without putting a lot of thought into where the data sets are stored or even how large those data sets are. As a result, electronic space and organization can quickly get out of hand. This presentation provides examples of how to utilize UNIX commands to locate files that you may be able to delete in order to free up electronic space. For example, data sets with "old" or "temp" at the end of the file name are typically intermediate data sets before a final data set is created or updated. The presentation also provides examples of reading in files, such as Excel files, that are not SAS files by using the UNIX FILENAME PIPE command and utilizing FIND - MAXDEPTH to monitor electronic space.

INTRODUCTION

SAS is a multi-purpose software with many capabilities including the ability to use a variety of UNIX commands within your SAS session. This eliminates the need to open a UNIX terminal to manually determine how much electronic space is available from one or more directories. This paper will nix the idea of leaving your SAS session to look up that information.

The directory structure within the Division of Biomedical Statistics and Informatics consists of a large electronic space partitioned into sections. Within those sections there are Disease Oriented Groups and further separation for each research project. Each research project folder contains numerous file types from study personnel. The need to meet tight deadlines causes users to start programs and create data sets without considering where those data sets are stored or how large the data sets are. This results in an electronic space and organization that can quickly get out-of-hand.

LS COMMAND

The ls command is one of UNIX's simplest commands and allows you to display what files and folders are in your directories. The ls command produces a data set including one record for each file and/or folder in the directory specified. This data set can be referenced using a FILENAME statement with the PIPE command within SAS, and then set into a SAS data set in order to potentially create new variables, check if folders/files exist, call execute a macro based on a particular condition, etc.

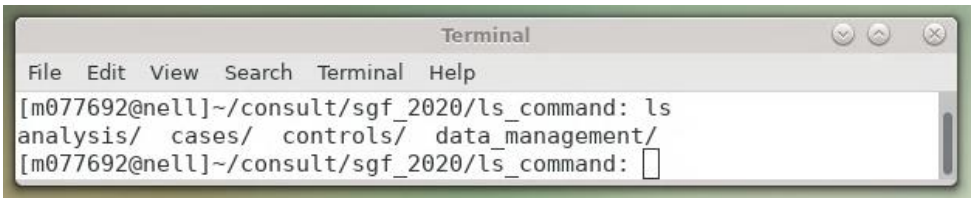
The FILENAME statement has the following syntax:

```
FILENAME fileref PIPE 'UNIX-command' <options>;
```

The *fileref* must be a valid SAS name and eight or less characters long. The PIPE command enables SAS to understand the following string is a UNIX command. The single or double quoted string should be written exactly as it would in the UNIX terminal including quotations and it is case sensitive.

Running the FILENAME PIPE command by itself doesn't print anything extra in the log, but enables you to specify the fileref in an infile statement within a SAS DATA step similarly to setting in a SAS data set.

DISPLAY SUBFOLDERS



Output 1. Screenshot of ls command run in UNIX terminal

The following example is using SAS code with the ls command to list subfolders in a particular location:

```
filename ls_examp pipe "ls ~/consult/sgf_2020/ls_command";

data ls_command_subfolders;
  length subfolders $100;
  infile ls_examp;
  input subfolders $;
run;
```

Table: WORK.LS_COMMAND_SUBFOLDERS

Total rows: 4 Total columns: 1

	subfolders
1	analysis
2	cases
3	controls
4	data_management

Output 2. Output from ls command listing subfolders

DISPLAY FILES

The following output displays the list of Excel files that will be created with the ls command code below:

Output 1. Screenshot of ls command run in UNIX terminal

The following example is using SAS code with the ls command to list files in a particular location:

```
filename ls_cases pipe "ls ~/consult/sgf_2020/ls_command/cases";

/*was today's case file created?*/

data todays_ls_command_file;
  length case_files $100;
  infile ls_cases;
  input case_files $;

  **create variable of file name expecting to find;
  case_file_name='cases identified_'||put(today(),date7.)||'.xlsx';

  if case_files=case_file_name then do;

    **create variable with full path;
    cases_file_path="~/consult/sgf_2020/ls_command/cases/"||case_file_name;

    call execute('%readin(cases_file_path)');**read in today's case file;

   output;
  end;

run;
```

In the SAS DATA step above, the current date is being compared to the dates in the names of the files output from the ls command. If the current date is found, SAS calls the %readin macro and runs the code within the macro to read in the particular file.

Table: WORK.TODAYS_LS_COMMAND_FILE | View: Column names | Filter: (none)

Total rows: 1 Total columns: 3

	case_files	case_file_name	cases_file_path
1	cases_identified_19JAN20.xlsx	cases_identified_19JAN20.xlsx	~/consult/sgf_2020/ls_command/cases/cases_identified_19JAN20.xlsx

Output 2. Output from ls command listing files

FILENAME PIPE COMMAND

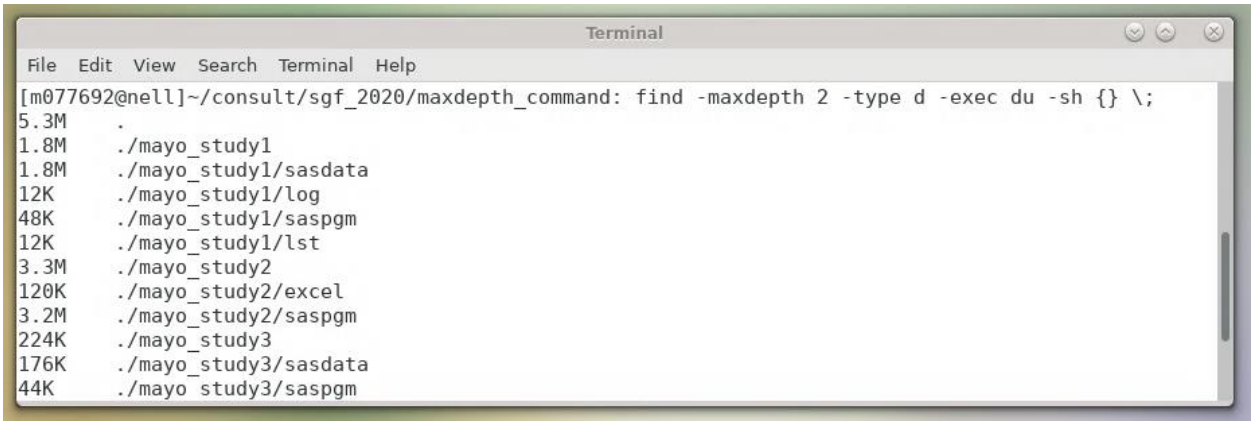
The ls command is useful to identify which folders/files are in your directories, but what is even more helpful is calculating how much space is being used and how much free space is available. That is where using the find command with -maxdepth within the FILENAME PIPE command come in handy.

FIND –MAXDEPTH COMMAND

The UNIX command you can add to the FILENAME PIPE command is:

```
find –maxdepth 2 –type d –exec du –sh {} \;
```

The number specified after –maxdepth specifies the number of levels of subfolders to produce in the output. After –type you specify either d for directory or f for file. Next, SAS tells UNIX to execute the du –sh command which calculates the disk usage for the directory specified, displays it in a human readable format as well as displays the total size of the directory.



Output 2. Screenshot of FIND -MAXDEPTH 2 command run in UNIX terminal

Changing the number specified after –maxdepth to a 1 will display the output and size at the first folder level.

Output 2. Screenshot of FIND -MAXDEPTH 1 command run in UNIX terminal

The following example is using SAS code with the find –maxdepth 2 command:

```
filename maxdepth pipe "find ~/consult/sgf_2020/maxdepth_command –maxdepth
2 –type d –exec du –sh {} \;";

data maxdepth_command_subfolders;
  length subfolders $100;
  infile maxdepth;
  input subfolders $;

  dir_size=scan(subfolders, 1, ' /');

run;
```

Table: WORK.MAXDEPTH_COMMAND_SUBFOLDERS | View: Column names | Filter: (none)

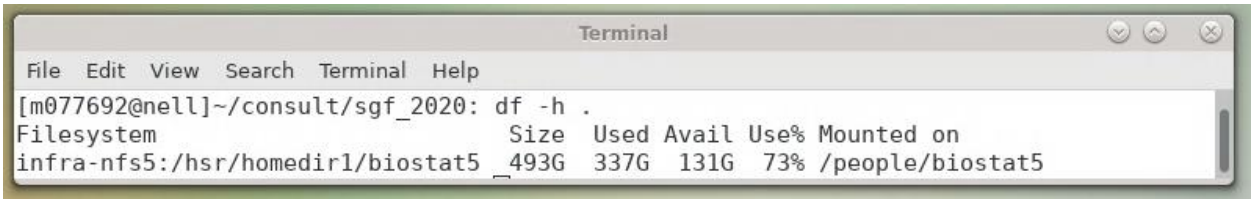
Total rows: 12 Total columns: 2

	subfolders	dir_size
1	5.3M /sgf_2020/maxdepth_command	5.3M
2	1.8M /sgf_2020/maxdepth_command/mayo_study1	1.8M
3	1.8M /sgf_2020/maxdepth_command/mayo_study1/sasdata	1.8M
4	12K sgf_2020/maxdepth_command/mayo_study1/log	12K
5	48K sgf_2020/maxdepth_command/mayo_study1/saspgm	48K
6	12K sgf_2020/maxdepth_command/mayo_study1/lst	12K
7	3.3M /sgf_2020/maxdepth_command/mayo_study2	3.3M
8	120K /sgf_2020/maxdepth_command/mayo_study2/excel	120K
9	3.2M /sgf_2020/maxdepth_command/mayo_study2/saspgm	3.2M
10	224K /sgf_2020/maxdepth_command/mayo_study3	224K
11	176K /sgf_2020/maxdepth_command/mayo_study3/sasdata	176K
12	44K sgf_2020/maxdepth_command/mayo_study3/saspgm	44K

Output 3. Output from find –maxdepth 2 command

DF COMMAND

The df command displays the overall amount of space available, how much space is currently being used as a number and a percentage, and how much space is currently available. This can be used to monitor available space and to inform colleagues that directory clean-up is needed.



Output 3. Screenshot of df command run in UNIX terminal

Running the df command in a UNIX terminal creates six columns of output: file system, total directory size, amount of space used, amount of space available, percentage of space used and directory location. Again, –h has been added after df so the results are in a human readable format. In order to create a SAS data set with the same information you must list six variables in the input statement.

The following example is using SAS code with the df command:

```
filename df_comm pipe "df -h ~/consult/sgf_2020/;";

data df_command_size;
  length file_system size used avail use_perc mounted $100 use_perc_num 8.;
  infile df_comm;
  input file_system $ size $ used $ avail $ use_perc $ mounted $;

  **create numeric variable out of character percentage;
  use_perc_num=scan(use_perc, 1, '%');
  remain_percent=(100-use_perc_num)/100;*decimal to apply percent format;

run;
```

Table: WORK.DF_COMMAND_SIZE | View: Column names | Filter: (none)

Total rows: 2 Total columns: 8

	file_system	size	used	avail	use_perc	mounted	use_perc_num	remain_percent
1	Filesystem	Size	Used	Avail	Use%	Mounted		
2	infra-nfs5:/hsr/homedir1/biostat5	493G	335G	133G	72%	/people/biostat5	72	0.28

Output 2. Output from df command

The output above displays the computed percentage of available space remaining for one file system. When there are multiple file systems included within a report there is a need to easily distinguish between the file systems that are near capacity and those that have enough free space. The COMPUTE statement in PROC REPORT can be used to fill in the background of the row if the remaining percentage of available space is below 25%.

The following example is using SAS code to display the result of the df command with PROC REPORT:

```
title "File system information as of &sysdate";

proc report data=df_command_size2 nowd;
  column file_system size used avail use_perc remain_percent;
  define file_system / display "File System";
  define size / display "Size";
  define used / display "Used";
  define avail / display "Available";
  define use_perc / display "% Used";
  define remain_percent / display "Remaining percent" format=5.2;

compute remain_percent;
  if remain_percent < .25 then call /*value before percent format applied*/
  define(_row_,"style","style={background=red}");
endcomp;

run;
```

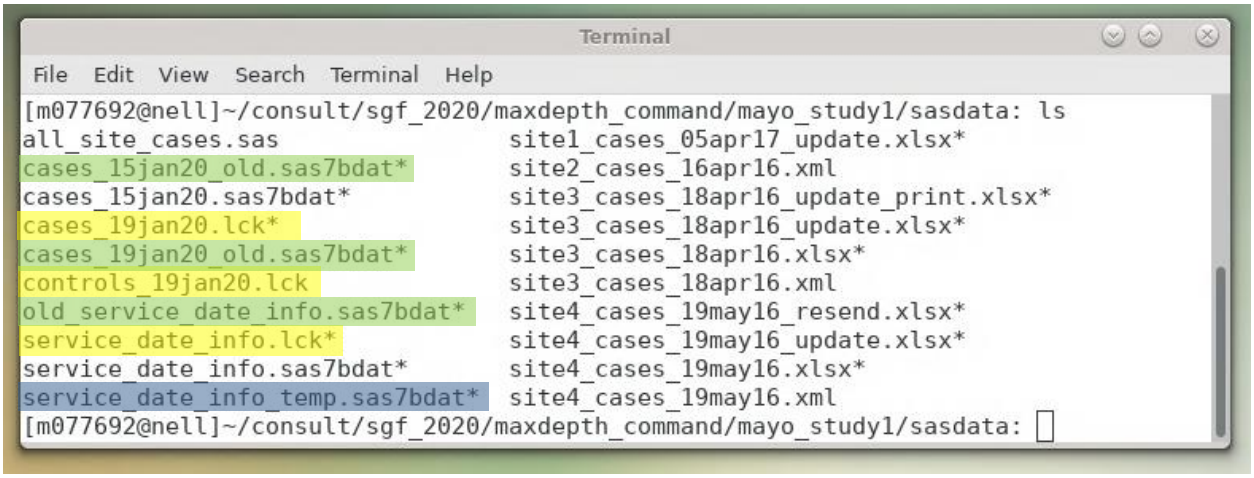
File system information as of 20JAN20

File System	Size	Used	Available	% Used	Remaining percent
infra-nfs5:/hsr/homedir1/biostat5	493G	335G	133G	72%	28%
infra-nfs5:/hsr/homedir1/biostat6	1.7T	1.2T	386G	76%	24%
infra-nfs5:/hsr/homedir1/biostat7	493G	290G	178G	62%	38%
infra-nfs5:/hsr/homedir1/biostat8	493G	190G	278G	41%	59%

Output 3. Output from PROC REPORT displaying df command results and highlighting file location with less than 25% of free space

.LCK FILES

Another way to assist colleagues in identifying files to remove from directories in order to free up space is to identify common files that programmers may not be aware have been created. If a program is interrupted or a SAS session shuts down while a data set is being created a .lck file may be saved with the same name as the data set that was being created and those files can get very large. You can use the FIND command in your SAS program to locate the files that need to be reviewed and possibly deleted. Including the asterisk before .lck allows SAS to search for files with any name.



Output 4. Screenshot of ls command run in UNIX terminal listing all files

The following example is using SAS code with the find command for files with a .lck extension:

```
filename dir_scan pipe "find ~/consult/sgf_2020 -name '*.lck'";

data chk_lck_file;
  length lck_file $100;
  infile dir_scan;
  input lck_file $;
run;
```

Table: WORK.CHK_LCK_FILE | View: Column names | Filter: (none)

Total rows: 3 Total columns: 1

	lck_file
1	/consult/sgf_2020/maxdepth_command/mayo_study1/sasdata/service_date_info.lck
2	/consult/sgf_2020/maxdepth_command/mayo_study1/sasdata/cases_19jan20.lck
3	/consult/sgf_2020/maxdepth_command/mayo_study1/sasdata/controls_19jan20.lck

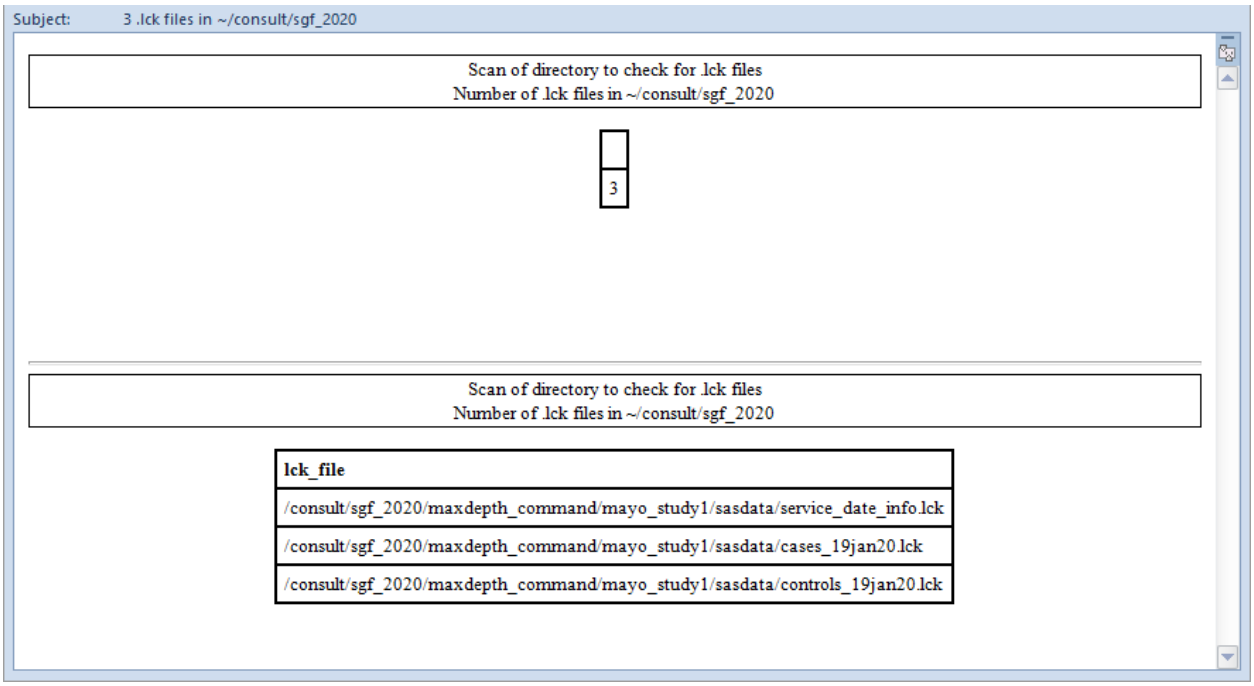
Output 2. Output from find command – .lck file extension

If the program is scheduled to run as a cron job, you can set up an email to be sent with the number of files found.

The following example is using SAS code to generate the file count and send an email:

```
proc sql noprint;
 select count(*)
 into: count_lck /*generate file count to be used in subject of email*/
 from chk_lck_file;
quit;

data _null_;
 filename mymail
   email to=("benetti.lorelle@mayo.edu")
         from=("benetti.lorelle@mayo.edu")
         subject="&count_lck .lck files in ~/consult/sgf_2020"
         content_type='TEXT/HTML';
         ods html style=printer body=mymail;
run;
```

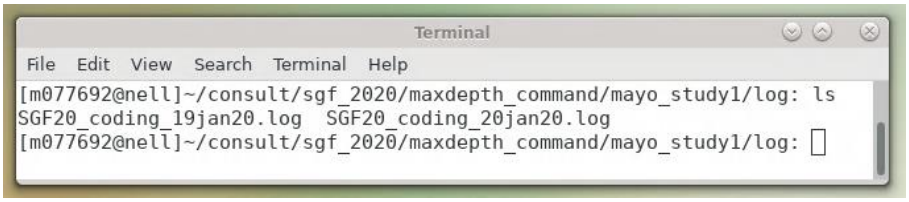
```
title 'Scan of directory to check for .lck files';
title2 'Number of .lck files in ~/consult/sgf_2020';

proc sql;
 select count(*)
 from chk_lck_file;
quit;

proc print data=chk_lck_file noobs;run;

ods html close;
```



Output 3. Screenshot of email sent with results of find command searching for files with .lck extension

.LOG/.LST FILES

When a program is run in Batch the log and output files are automatically saved in the directory where the program is saved unless another location is specified. Having the ability to go back to the log for previous program runs can be tremendously helpful. However, the .log and .lst files can get very large. Again, including the asterisk before .log allows SAS to search for files with any name.



Output 5. Screenshot of ls command run in UNIX terminal listing all files

The following example is using SAS code to find files with a .log extension:

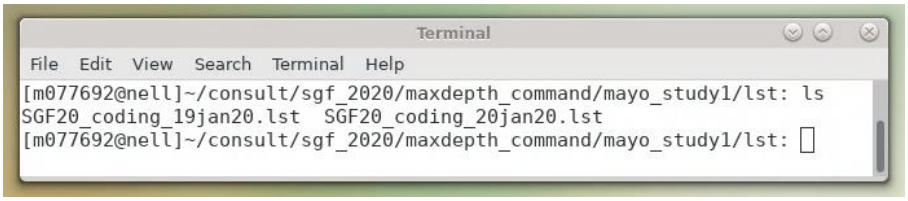
```
filename find_log pipe "find ~/consult/sgf_2020 -name '*.log'";

data chk_log_file;
  length log_file $150;
  infile find_log;
  input log_file $;
run;
```

Table: WORK.CHK_LOG_FILE | View: Column names | Filter: (none)

Total rows: 2 Total columns: 1

	log_file
1	/consult/sgf_2020/maxdepth_command/mayo_study1/log/SGF20_coding_20jan20.log
2	/consult/sgf_2020/maxdepth_command/mayo_study1/log/SGF20_coding_19jan20.log

Output 2. Output from find command – .log extension



Output 3. Screenshot of ls command run in UNIX terminal listing all files

The following example is using SAS code to find files with a .lst extension:

```
filename find_lst pipe "find ~/consult/sgf_2020 -name '*.lst'";

data chk_lst_file;
  length lst_file $150;
  infile find_lst;
  input lst_file $;
run;
```

Table: WORK.CHK_LST_FILE | View: Column names | Filter: (none)

Total rows: 2 Total columns: 1

	lst_file
1	/consult/sgf_2020/maxdepth_command/mayo_study1/lst/SGF20_coding_19jan20.lst
2	/consult/sgf_2020/maxdepth_command/mayo_study1/lst/SGF20_coding_20jan20.lst

Output 4. Output from find command – .lst extension

"OLD" OR "TEMP" IN FILE NAMES

There can be many reasons as to why a data set has the word "old" or "temp" tacked on to the end of the file name. Most commonly these are intermediate data sets created before a data set is finalized or a current data set is updated. There may be a few cases where those files are actually useful and should be kept. Again, including the asterisk before and after "old" allows SAS to search for files with any name including "old".

The following example is using SAS code to find files with "old" in the file name:

```
filename find_old pipe "find ~/consult/sgf_2020 -name '*old*.*'";

data chk_old_file;
  length old_file $150;
  infile find_old;
  input old_file $;
run;
```

Table: WORK.CHK_OLD_FILE | View: Column names | Filter: (none)

Total rows: 3 Total columns: 1

	old_file
1	/consult/sgf_2020/maxdepth_command/mayo_study1/sasdata/cases_15jan20_old.sas7bdat
2	/consult/sgf_2020/maxdepth_command/mayo_study1/sasdata/old_service_date_info.sas7bdat
3	/consult/sgf_2020/maxdepth_command/mayo_study1/sasdata/cases_19jan20_old.sas7bdat

Output 6. Output from find command – "old" in file name

The following example is using SAS code to find files with "temp" in the file name:

```
filename findtemp pipe "find ~/consult/sgf_2020 -name '*temp*.*'";

data chk_temp_file;
  length temp_file $150;
  infile findtemp;
  input temp_file $;
run;
```

Table: WORK.CHK_TEMP_FILE | View: Column names | Filter: (none)

Total rows: 1 Total columns: 1

	temp_file
1	/consult/sgf_2020/maxdepth_command/mayo_study1/sasdata/service_date_info_temp.sas7bdat

Output 2. Output from find command – "temp" in file name

EXAMPLE: READING IN FILES THAT ARE NOT SAS DATA SETS

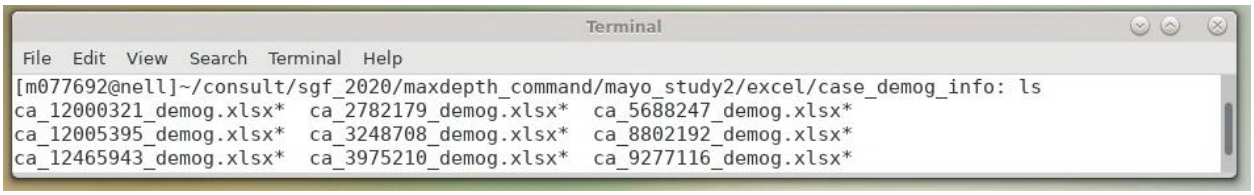
Once the known files are extracted from a FILENAME PIPE command they can be used to efficiently import data from other sources. This is especially useful when the number of files in a directory will be unknown or constantly changing. The following is a real world example of making a project much more efficient when combining UNIX command lines with SAS.

Task scenario:

Read in the case demographic information for a project. However, there are a few caveats:

1. Based on the process of obtaining case demographic information, each patient was logged in their own excel file.
2. The subject variable, which uniquely identifies each case, can be pulled from the file name. However, the subject value is not always the same number of characters.

This can easily be accomplished using the techniques described earlier in this paper. The UNIX ls command is used to create a data set with one record for each file in the directory and then within a DO loop a macro is called to import each file as its own SAS DATA set.



Output 7. Screenshot of case_demog_info directory in UNIX terminal

The following example is using SAS code to use the UNIX ls command to create a data set with one record for each file in the location then within a DO loop call a macro to import each file:

```
** macro to read in files once the name is generated below;
%macro demog_readin(id,subject);

proc import file="&id" dbms=xlsx
 out=&subject replace;
run;

%mend demog_readin;


** create data set of list of variables in directory;

filename indata pipe "ls -1
~/consult/sgf_2020/maxdepth_command/mayo_study2/excel/case_demog_info";

data directory_list;
 infile indata truncover;
 input file_name $25.;**create variable of file name - includes extension;

file2read='~/consult/sgf_2020/maxdepth_command/mayo_study2/excel/case_demog
_info/'||(file_name);
run;


** determine the number of files in directory - need to run loop below;
proc sql noprint;
 select count(*)
 into: file_count
 from directory_list;
quit;


** call macro to create one data set for each file in the directory;
data run_macro;
 set directory_list;
 record=_n_;

 id=scan(file_name, 2, '_');**create variable with subject identifier #;

 file_id='ca_'||scan(file_name, 2, '_.');

 call symput('m_file2read',trim(file2read));
```

```
 call symput('m_file_id',trim(file_id));

 if record le &file_count then do;

  call execute ('%demog_readin(&m_file2read,&m_file_id)');**call macro;
 end;

run;
```

Table: WORK.DIRECTORY_LIST | View: Column names | Filter: (none)

Total rows: 9 Total columns: 2

	file_name	file2read
1	ca_12000321_demog.xlsx	~/consult/sgf_2020/maxdepth_command/mayo_study2/excel/case_demog_info/ca_12000321_demog.xlsx
2	ca_12005395_demog.xlsx	~/consult/sgf_2020/maxdepth_command/mayo_study2/excel/case_demog_info/ca_12005395_demog.xlsx
3	ca_12465943_demog.xlsx	~/consult/sgf_2020/maxdepth_command/mayo_study2/excel/case_demog_info/ca_12465943_demog.xlsx
4	ca_2782179_demog.xlsx	~/consult/sgf_2020/maxdepth_command/mayo_study2/excel/case_demog_info/ca_2782179_demog.xlsx
5	ca_3248708_demog.xlsx	~/consult/sgf_2020/maxdepth_command/mayo_study2/excel/case_demog_info/ca_3248708_demog.xlsx
6	ca_3975210_demog.xlsx	~/consult/sgf_2020/maxdepth_command/mayo_study2/excel/case_demog_info/ca_3975210_demog.xlsx
7	ca_5688247_demog.xlsx	~/consult/sgf_2020/maxdepth_command/mayo_study2/excel/case_demog_info/ca_5688247_demog.xlsx
8	ca_8802192_demog.xlsx	~/consult/sgf_2020/maxdepth_command/mayo_study2/excel/case_demog_info/ca_8802192_demog.xlsx
9	ca_9277116_demog.xlsx	~/consult/sgf_2020/maxdepth_command/mayo_study2/excel/case_demog_info/ca_9277116_demog.xlsx

Output 2. List output data set with file name and full path

The following is the SAS log showing macro code of the first file being imported:

```
MLOGIC(DEMOG_READIN):  Beginning execution.
MLOGIC(DEMOG_READIN):  Parameter ID has value
      ~/consult/…/case_demog_info/ca_12000321_demog.xlsx

MLOGIC(DEMOG_READIN):  Parameter SUBJECT has value ca_12000321
MPRINT(DEMOG_READIN):   proc import
file="~/consult/…/case_demog_info/ca_12000321_demog.xlsx" dbms=xlsx
out=ca_12000321
replace;
MPRINT(DEMOG_READIN):   run;


NOTE: CALL EXECUTE generated line.
1         + proc import
 file="~/consult/…/case_demog_info/ca_12000321_demog.xlsx"
 dbms=xlsx out=ca_12000321 replace;

NOTE: VARCHAR data type is not supported by the V9 engine. Variable
patient_last_name has been converted to CHAR data type.
NOTE: VARCHAR data type is not supported by the V9 engine. Variable
patient_first_name has been converted to CHAR data type.
NOTE: The import data set has 1 observations and 3 variables.
NOTE: WORK.CA_12000321 data set was successfully created.
NOTE: PROCEDURE IMPORT used (Total process time):
      real time           0.02 seconds
      cpu time            0.01 seconds
```

CONCLUSION

In conclusion, incorporating the UNIX commands listed above in different ways can keep you in your SAS session and allow you to retrieve the electronic file information you would typically find using a UNIX terminal. This paper just scratches the surface as to what commands you can use and how to include them in your SAS programs.

REFERENCES

Alexander, Alvin. "Linux file searching: Search for text in files with find and grep commands." Accessed January 20, 2019. Available at https://alvinalexander.com/blog/post/linux-unix/combine-find-grep-search-subdirectories.

Carpenter, Arthur L. 2008. "PROC REPORT: Compute Block Basics – Part I Tutorial." Proceedings of SAS Global Forum 2008. Available at https://support.sas.com/resources/papers/proceedings/pdfs/sgf2008/031-2008.pdf.

SAS Institute. "PERCENT Format." Accessed January 21, 2020. https://documentation.sas.com/?docsetId=ds2ref&docsetTarget=n1sw88r281qsn1n1r77pj0zn40ha.htm&docsetVersion=3.1&locale=en.

SAS Institute. "Reading from and Writing to UNIX Commands (PIPE)" Accessed January 20, 2019. Available at https://documentation.sas.com/?docsetId=hostunx&docsetTarget=n1ceb0xedanuj3n19l3g73awk1wf.htm&docsetVersion=9.4&locale=en.

Virgile, Bob. "MAGIC WITH CALL EXECUTE." Proceedings of SAS Users Group International 22. Available at https://support.sas.com/resources/papers/proceedings/proceedings/sugi22/CODERS/PAPER86.PDF

SAS Institute. "Referencing External Files." Accessed December 2019. http://support.sas.com/documentation/cdl/en/hostwin/69955/HTML/default/viewer.htm#n07buc7sg08fdrn1c1jmmr8hl78r.htm

"Learning Linux Commands: ls". Accessed February 12, 2020. Available at https://linuxconfig.org/learning-linux-commands-ls.

Ashok, Arunlal. "Some useful "du – disk usage" command usages – Unix/Linux". Accessed February 17, 2020. Available at https://www.crybit.com/du-command-usage/.

"mindepth and maxdepth in Linux find() command for limiting search to a specific directory". Accessed February 21, 2020. Available at https://www.geeksforgeeks.org/mindepth-maxdepth-linux-find-command-limiting-search-specific-directory/

ACKNOWLEDGMENTS

Thank you to Jeffrey Meyers for his willingness to provide information about the SAS Global Forum submission process and for his thorough review and evaluation of this paper.

CONTACT INFORMATION

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

Lorelle Benetti
Division of Biomedical Statistics and Informatics
Mayo Clinic
Benetti.lorelle@mayo.edu