Paper 059-2010

Short Course for the Korn® shell "find" Command and Piping the information into SAS®

John Charles Gober, Chuc Phan, Theodore C. Logothetti
U.S. Bureau of the Census, American Community Survey Office

ABSTRACT

The SAS metatables contained within the SASHELP library contain a wealth of information about SAS libraries, datasets, and variables along with other useful information, which can be, used directly in your SAS programs. Sometimes, however, the only way to obtain system-related information on non-SAS files or their attributes is to use one of the utilities in your environment. In the Linux® environment, the “find” command fulfills this need. This paper demonstrates the flexibility of the Korn shell '*find*' command to capture system level information on files and file structures in general. Examples of importing this information into SAS using the PIPE engine will also be presented.

KEYWORDS

filename pipe, find, Linux®, ls, pipe, Redhat®, stdout, UNIX®,macro quoting, %quote, %nrbquote, %nrstr, %superq, KornShell®.

INTRODUCTION

The Census Bureau’s American Community Survey Office (ACSO) is extremely sensitive about who has access to its data, even within the Bureau itself. Mandated by Congress under Title 13, Title 15, and Title 26 this confidentiality is of the utmost importance. To uphold this trust, the ACSO has numerous programs, which run daily to verify the rights and privileges to these files and grant access accordingly. At the ACSO the *find* command is rapidly becoming a replacement for the *ls* command not only for its less restrictive operation but also for its power and flexibility. The *ls* command differs in functionality from the *find* command in that *ls* requires a buffer when searching wildcard strings to store returned results before sending the results to stdout. Depending on the platform, this sometimes causes errors against large queries. The *find* command can return virtually unlimited results. *find* is also more robust when it comes to generating output allowing the user to pick, choose, order, and parse information. This is a useful feature for including results into other applications. This paper will review a few of the most useful features that have enhanced the performance and functionality of many of the applications used for the American Community Survey Office which currently reside on a **Linux 2.6.9-78.013.ElSmp** platform. The functionality of *find* may vary on other platforms.

THE BASIC *find* COMMAND

Even in its simplest form the *find* command can generate a list of files in a directory that can be written to an ASCII file and then imported into SAS. Or, if wished, the files can be read directly into SAS via the pipe engine (*discussed later*). There is no need for extra statements or commands to generate a simple list of files no matter how long that list may be. The following statement is an example of generating a basic list of files and sending that list to stdout. Note that in the first example the *find* command is executed in the directory in which the files are located. The second command can be used outside of the directory to be queried. Because of the whitespace example three might not be the query desired due to the fact that it actually contains three separate queries, a directory query that will return all files in the entire directory, and also two separate file queries.

Example 1 **`find smp???_*.sas`**
Example 2 **`find /user/foobar/smp???_*.sas`**
Example 3 **`find /user/foobar/ smp???_*.sas roll*.sas`**

Test contents of /user/foobar
.profile foobar.sas smp10_foobar.sas smp20_foobar.sas SMP30_foobar.sas
smp40_foobar.sas roll.sas smp000/smp10_foobar.sas

Results Example 1	*Results Example 2*	*Results Example 3*
smp000	/user/foobar/smp10_foobar.sas	/user/.../.profile
smp000/smp10_foobar.sas	/user/foobar/smp20_foobar.sas	/user/.../foobar.sas
smp10_foobar.sas	/user/foobar/smp40_foobar.sas	/user/.../smp10_foobar.sas
smp20_foobar.sas		/user/.../smp20_foobar.sas
smp40_foobar.sas		/user/.../SMP30_foobar.sas
		/user/.../smp40_foobar.sas
		/user/.../smp10_foobar.sas
		/user/.../smp20_foobar.sas
		/user/.../smp40_foobar.sas
		/user/smp000/smp10_foobar.sas
		/user/.../roll.sas

Notice the use of both global and positional wildcards in the file name query and the lack of quotation marks. This will be important later. Wildcards can also be used in the directory portion of the query. Also note the unwanted and duplicate file names for example three is caused by the recursive nature of the find command.

ACTIONS AND TESTS PERFORMED BY *find*

Because of its recursive nature, the results from *find* are sometimes presented in a format other than that which is desired. Typically ACSO users are mostly interested in the file name portion of a returned query without capturing any other information, i.e. directory path, group, owner, permissions, file size, modification date, etc. However, as noted in the examples above, this is not always the case. This is where action parameters can be useful to customize the look and feel of returned results. With the *find* command the user has the option of having as little or as much information returned from a query as needed.

Probably the most familiar (and sometimes abused) output format to users of the *find* command is the *–ls* parameter, which formats results in the same way as its Unix/Linux *ls* command counterpart. In fact *find* actually uses the *ls –dils* command in its routine. Why then should *find* be used over *ls*? On some platforms the *ls* command has a buffer limitation when executing a query using wildcards causing an ”argument too long” condition. This condition doesn't occur with *find –ls* due to the fact that *find* is doing the query and then passing the results to the *-ls* parameter for output.

command: `find /user/foobar/ -name "smp???_*" –maxdepth 1 -ls`

```
results:  5622709  24 -rw--w----   1 foobar acsodp  13131 Jul  6 15:47 /user/foobar/smp010_foobar10.sas
          5622715  20 -rwxrwxrwx          1 foobar acsodp  10301 Jul 10 15:42 /user/foobar/smp040_foobar40.sas
          5622711  24 -rwxrwxrwx 1 foobar acsodp  15249 Jul  7 11:54 /user/foobar/smp020_foobar20.sas
```

Notice the use of the *–name* parameter immediately after the directory path. The *–name* parameter is not necessary if the file pattern to be searched is concatenated to the starting directory path. However the *–name* parameter is case sensitive. To search for file names containing either upper or lower case characters, the *–iname* parameter should be used. To keep *find* from executing recursively a within a directory, use the *–maxdepth* option.

Instead of using the *–ls* parameter of *find,* many users have expanded their knowledge of *find* to include the use of its formatted output capabilities. Analogous to the fairytale of Goldilocks in "The Three Bears," why would a programmer do too much or too little when just enough will suffice? By using formatted output, results can be pre-customized for import into another application such as SAS. After constructing the tests that the *find* command can perform, not only can one pick and choose specific information and exclude everything else, one can also order that information, choose how to delimit the data, select how particular information should be presented, and also inject literal strings into the output. This is where the *-printf* parameter is useful. Below is a fairly complex *find* statement. Hopefully one that will never be used. The *–printf* clause is highlighted in blue and will be discussed below.

```
find /user/foobar(a) -type f(b) -name 'pgm*'(c) -not -name '*.????*'(d) -maxdepth '2'(e)
\( -ctime +10 -o -ctime -100 \)(f) -perm u=rwx,g=rwx,o=r(g)
-printf '%h\054 %p\073 %f Foobar %AD %b %k %s %m\n'
```

Formatting options used by the *–printf* are enclosed in either single or double quotes. The use of single verses double is dictated by where and how the *find* command is used. The %h prints out the directory path. The \054 is the ASCII decimal equivalent of a comma. Notice the space between the 4 and the %p; this space is printed also. To remove the space, either place the %p next to the 4 or insert a \b for a backspace. The %p\073 prints the file name followed by a semi colon. The %f prints the directory and filename (space on either side). "Foobar" is a literal, which is also printed, followed by a space. %AD, %b, %k, and %s will print the date in mm/dd/yy format, file size in blocks, file size in 1k blocks, and file size in bytes, all separated by spaces. The %m\n prints file permissions in octal.

In addition to being able to format the output, *find* has a very large collection of built-in tests that can be performed against files and directories. Like the formatting options, there are too many to present in a single paper so only a few useful ones have been presented. The above *find* command will search the path (a) /user/foobar for (b) files of type f or regular files. The pattern of the file(s) to look for (c) should begin with pgm but (d) with a file extension not contain a pattern containing any three characters followed by a 2. Search only (e) the directory listed and its sub-directories recursively two levels deep. Finally, only find files (f) created between ten and one hundred days ago with (g) permissions of 774. (The owner and group members have read, write, and execute privileges, while everyone else has only read access).

The above *find* will return output similar to standard output the following results:

```
/user/foobar, /user/foobar/pgm.asv3; pgm.asv2 Foobar 12/17/09 8 4 2108 774
```

There are over seventy tests, actions, and formats that can be used with *find* so the possibilities are almost endless. The command even has the capability of executing other commands and programs within itself with the *-exec* parameter without having to direct and redirect results to be used with other commands.

SAS PIPE ENGINE

The ability of SAS to be able to retrieve and process information from the host is well known. A popular way among users is to use the *filename* statement in conjunction with the *pipe* engine. Retrieving a list of filenames is especially simple as shown in the following two examples (both examples will return identical results).

```
filename in1 pipe 'find /user/foobar/pgm*';
filename in1 pipe 'ls  /user/foobar/pgm*';
```

However, more times than not, additional information other than a simple file list must be obtained to fulfill a request. As shown in the previous section, a request can be as simple or as complicated as needed and can spread across several lines of code. There are also times when a file list must be further restricted to eliminate certain files that fall within a pattern. There may be a need to check the age of certain files or file permissions. Ownership and groups might be also needed. Unfortunately, the more complex a request, the longer the statement will take to execute. In addition, the *find* command allows the use of the ampersand, percent, and quote symbols along with other symbols that SAS may try to interpret and resolve according to SAS rules but not the rules and syntax of the *find* command and visa versa. A single ”best solution” to answer the problem of special symbols does not exist. Specific macro quoting functions were found to work well under some conditions, but not under others. Due to the possible long length of a *filename pipe* statement and trying to keep SAS code readable, spacing was also found to be critical, especially when macros are involved to reduce clutter.

Many programmers would give up using a function or routine at the first sign of trouble. They would use the overused excuse of SAS’s inability to function properly under such conditions when in fact it is the programmer’s lack of effort to try to find a viable solution. Below is a SAS *filename pipe* statement that took many frustrating hours to develop. It would have been easy to give up at the first signs of failure but perseverance prevailed and the versatility of SAS was again proven. The snippet of code below is an example of what can be accomplished.

```
%let date=%sysfunc(date(), mmddyy8.);
%let time=%sysfunc(time(), time.);
%let my_sysdate = &date &time;
%let fnd1 = -printf %quote(%') %nrstr(%h\054 %p\054 %f Foobar) &my_sysdate
%nrstr(\073 %t\054 %AD
%b %k %s %m\n) %quote(%');

filename in1 pipe "find /user/foobar -type f -name 'pgm*' -not -name '???2*'
-maxdepth '2' \( -ctime +10 -o -ctime -100 \) -perm u=rwx,g=rwx,o=r
%superq(fnd1)" lrecl=400;

data temp2;
   infile in1 pad truncover;
   input @001 var1 $400.;
   output;
run;
```

The above code returns the same results as the *find* example on page three except now the command has been wrapped inside of the SAS *filename* statement. Only one small addition to the *-printf* parameter has been made, the addition of *&my_sysdate*. Because of the length of the *find* command it was necessary to split the command over several lines of code. It was also decided in writing this paper that it would be interesting to see the effects of using macro variables to aide readability.

In order to use macro variables, in this case *fnd1*, it was necessary to enclose the argument to the *filename* statement inside double quotes. This in turn made it necessary to use single quotes for arguments to the *find* command. However, the difficult part was proving that SAS had the capability of handling the percent and ampersand symbols which were being used as options to the *–printf.* There is probably no good reason to embed the macro variable *&my_sysdate* as a literal to be printed to stdout via *find* when it can be read directly by a datastep, but it did make an interesting problem. Without any macro quoting SAS was able to recognize the symbol and, therefore, resolve *&my_sysdate*. It was being resolved correctly even within the confines of the find and *-printf*.

Also included in the macro variable *fnd1* are percent symbols. These percent symbols, even though inside the *–printf,* were being interpreted by SAS as a macro call and would return the warning message “Apparent invocation of macro *x* not resolved”. Since this was only a warning, it

did not affect the functionality of the code but needed to be resolved. Leaving warning messages in a program violates due diligence and can often lead to questions and panic to a programmer who just took over responsibility of the code. These percent symbols needed to be masked. However, masking the percent symbols yet not masking the ampersand, was a big problem. The original solution to this problem was to quote each individual percent symbol. As one can see even though it worked it made the statement too wordy:

```
%let fnd1 = -printf '%quote(%)h\054 %quote(%)p\054 %quote(%)f … %quote(%)t etc.
```

After experimenting with several other quoting techniques the *%nrstr* function was decided on as being the cleanest appearing option. The *%nrstr* function could have included the *–printf* as part of the argument, but this was only a style issue. The downfall of the *%nrstr* quoting is that the macro variables contained within *&fnd1* would also be quoted and not resolve properly when used elsewhere due to this quoting and again had to be dealt with by using even more quoting functions. The final solution was to create a statement with a unique placement of quoting functions and quotes:

```
%let fnd1 = -printf %quote(%') %nrstr(%h\054 %p\054 %f Foobar) &my_sysdate
%nrstr(\073 %t\054 %AD %b %k %s %m\n) %quote(%');
```

The %quote prevents any tokens, with the exception of the "%" and "&" signs, from being interpreted as macro calls or macro variables. The percent symbol ("%") must precede the single quotation mark within the parentheses of the %quote macro function. The %nrstr prevents any token from being interpreted as a macro call or macro variable. Notice that the &my_sysdate is outside of any of the macro functions (%quote or %nrstr), since it is a macro variable which should be resolved.

When fully resolved, the macro variable &fnd1 becomes:

```
-printf '%h\054 %p\054 %f Foobar 17FEB10 \073 %t\054 %AD %b %k %s %m\n'
```

The %superq prevents macro variables *within a macro variable* from being resolved. (Notice also that the "&" was removed from the &fnd1 macro variable; this is required when the %superq function is used.) So the "%h," "%f," "%t," "%AD," "%b," "%k," "%s," and "%m" symbols do not get resolved within the &fnd1 variable.

Fully resolved, the final "find" command then becomes:

```
find /user/foobar -type f -name 'pgm*' -not -name '????*'
b-maxdepth '2' \( -ctime +10 -o -ctime -100 \) -perm u=rwx,g=rwx,o=r
b-printf '%h\054 %p\054 %f Foobar 17FEB10 \073 %t\054 %AD %b %k %s %m\n'
```

The blue b symbolizes a blank character, which was found to be necessary in some cases when continuing a command across several lines. Note the resolution of the *fnd1* and *my_sysdate* macro variable. This statement is then piped into the "filename" statement, which can used in a DATA step to read the list of files into a SAS dataset.

CAVEATS

There are some arguments and parameters on our Redhat Linux version that we could not get to work reliably. Some returned error conditions while others just returned unexpected results. This could be caused by the different 'flavors' of *find* between platforms or the authors lack of time to discover a workable solution. When solutions were found often it was usually just by adding whitespace. A little experimenting goes a long way.

CONCLUSION

To be technically correct this paper should be titled '**Short Course for the gnu® "find" Command running in the Linux® kernel using the KornShell® Language and Piping the information into SAS®**'. This paper demonstrated how even the most complex gnu find command can be successfully interpreted and executed by SAS. Even though these complex statements will probably never be used via the SAS filename pipe engine the techniques presented can be adapted to other statements and routines.

REFERENCES

Garfinkel, Simson,Gene Spafford, and Deborah Russell. Practical UNIX Security. United States: O'Reilly & Associates, 1991.

Linux Documentation Project. 2003-2004. Volunteer operated web site. 23 Oct 2009. <http://linux-documentation.com/en/man/man1p/find.html >.

SAS Publishing. 1999. SAS Macro Language Reference, Version 8. SAS Institute Inc., Cary, North Carolina.

Tatar, Maria (2002). The Annotated Classic Fairy Tales. New York: W.W. Norton & Company, Inc.. pp. 245–246. ISBN 0-393-05613-3.

ACKNOWLEDGEMENTS

The Authors would like to acknowledge all of the dedication and hard work of their fellow employees at the American Community Survey Office to which this paper was a direct result. Without their dedication to innovation and insight this paper could not have been envisioned.

AUTHOR CONTACTS

John Charles Gober
Bureau of the Census
4600 Silver Hill Road
ACSO HQ-3H460E
Washington DC 20233
301-763-5964

Chuc Phan
Bureau of the Census
4600 Silver Hill Road
ACSO HQ-3H460A
Washington DC 20233
301-763-1211

Theodore C. Logothetti
Bureau of the Census
4600 Silver Hill Road
ACSO HQ-3K064D
Washington DC 20233
301-763-1901

TRADEMARK INFORMATION