SESUG Paper 021-2021

Some Linux Shell Scripts for SAS® Programmers

Hengwei Liu, Hengrui USA

ABSTRACT

Many pharmaceutical companies use SAS on Linux server. Linux shell scripting is a power tool for the SAS programmers. It can be used to read text files and extract information. It can be used to do some file operations. In this paper some Linux shell scripts of interest to SAS programmers are discussed.

INTRODUCTION

Many SAS programmers run some Linux commands in their daily work, such as ls, cd, pwd and grep. For some tasks, it is worth the effort to set up a small shell script. It can be used later when such tasks arise again. A collection of shell scripts for some routine jobs improves efficiency and makes life easier for the programmers.

In this paper some Linux shell scripts are discussed. For more information about shell scripts see references [1] and [2].

To understand the scripts in this article, the readers are expected to have basic knowledge of shell scripting, including sed, awk, pipe and command substitution.

DETAILS FOR THE SHELL SCRIPTS

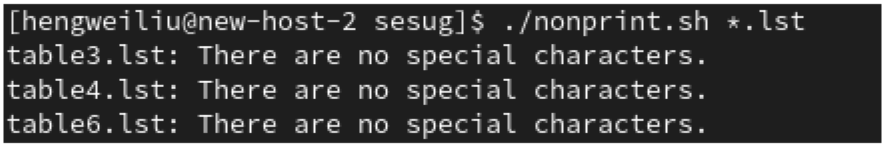
NONPRINT.SH

Hundreds of LST files may be generated for the tables and listings in a clinical study. As non-printable characters in the LST files cause problems, it is necessary to check those files for non-printable characters. This can be done with a script.

This script is run in the folder where the LST files are located. The command line arguments are the names of the files to be checked. Wildcard is allowed. Command substitution is used to find all the non-printable characters. The regular expression for printable characters is [[:print:]]. In the script the negation of this expression is used. The form feed with octal code o14 is not of interest for the purpose of the script. It is removed by the sed command. The user is given a notice whether there are special characters in the files.

```
#!/usr/bin/bash

for file in $*
do
  nonprint=$(cat  $file | \
              grep [^[:print:]] | \
              sed '/\o14/d' )

  if [[ $nonprint = "" ]]
  then
      echo "$file: There are no special characters."
  else
      echo "$file: There are special characters: $nonprint"
  fi
done
```

The Display 1 shows a sample output.



```
[hengweiliu@new-host-2 sesug]$ ./nonprint.sh *.lst
table3.lst: There are no special characters.
table4.lst: There are no special characters.
table6.lst: There are no special characters.
```

Display 1. Sample Output of the Script nonprint.sh

REPLACE.SH

It often happens that some SAS programs from one study can be used with minor modification in another clinical study. There are situations when the programmer wants to make some global change, e.g. change the word RANDOMIZATION to ENROLLMENT in multiple SAS programs. Instead of changing the SAS programs one by one, a script can be used to make the change in one fell swoop.

The script is run in the folder where the SAS programs are located. The command line arguments are the names of the SAS programs. Wildcard is allowed. The sed command is used to replace the word RANDOMIZATION by ENROLLMENT in all the SAS programs. A new SAS program with the prefix “new_” is created for each SAS program in the argument.

```
#!/usr/bin/bash
PREFIX="new_"
OLD_WORD="RANDOMIZATION"
NEW_WORD="ENROLLMENT"

for file in $*
do
 sed "s/$OLD_WORD/$NEW_WORD/g" $file > $PREFIX$file
done
```

DATASIZE.SH

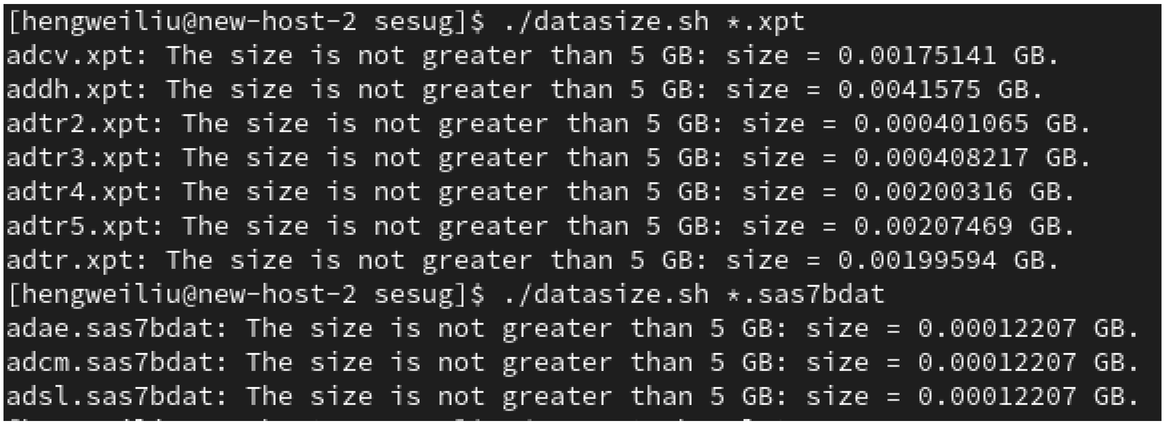
In the FDA submission, the sponsor needs to submit datasets. The maximum size allowed for a dataset is 5 GB. If the dataset is larger than 5 GB, it must be split into smaller datasets. See reference [3]. In a study there can be many SDTM and ADaM datasets. We can use a script to check the size of the datasets.

The script is run in the folder where the SAS datasets are located. The command line arguments are the dataset names. Wildcard is allowed. The size of the dataset in GB is calculated from the size in byte. The user is notified whether the dataset size is greater than 5 GB, and the size is printed out.

```
#!/usr/bin/bash


for file in $*

do

SIZE=$(ls -l $file | awk '{print $5}')

SIZE_GB=$( awk -v BYTE_TO_GB=1073741824 -v SIZE="$SIZE" \

'BEGIN {print (SIZE/BYTE_TO_GB)}' )


  if [[ $SIZE_GB > 5 ]]

  then
```

```
      echo "$file: The size is greater than 5 GB: \
size = $SIZE_GB GB."
  else
      echo "$file: The size is not greater than 5 GB: \
size = $SIZE_GB GB."
  fi
done
```

The Display 2 shows a sample output.



```
[hengweiliu@new-host-2 sesug]$ ./datasize.sh *.xpt
adcv.xpt: The size is not greater than 5 GB: size = 0.00175141 GB.
addh.xpt: The size is not greater than 5 GB: size = 0.0041575 GB.
adtr2.xpt: The size is not greater than 5 GB: size = 0.000401065 GB.
adtr3.xpt: The size is not greater than 5 GB: size = 0.000408217 GB.
adtr4.xpt: The size is not greater than 5 GB: size = 0.00200316 GB.
adtr5.xpt: The size is not greater than 5 GB: size = 0.00207469 GB.
adtr.xpt: The size is not greater than 5 GB: size = 0.00199594 GB.
[hengweiliu@new-host-2 sesug]$ ./datasize.sh *.sas7bdat
adae.sas7bdat: The size is not greater than 5 GB: size = 0.00012207 GB.
adcm.sas7bdat: The size is not greater than 5 GB: size = 0.00012207 GB.
adsl.sas7bdat: The size is not greater than 5 GB: size = 0.00012207 GB.
```

Display 2. Sample Output of the Script datasize.sh

LOG2SAS.SH

Programmers work with many SAS programs. Occasionally a programmer may delete a SAS program by mistake. Don’t panic, as all is not lost. The SAS program can be recovered from the log file through a script. Let’s review a sample log file:

```
2 The SAS System                             12:43 Wednesday, March 3, 2021

11         where sex='F';
12         run;

NOTE: There were 2 observations read from the data set WORK.A.
      WHERE sex='F';
NOTE: The PROCEDURE PRINT printed page 1.
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.23 seconds
      cpu time            0.04 seconds


NOTE: SAS Institute Inc., SAS Campus Drive, Cary, NC USA 27513-2414
NOTE: The SAS System used:
      real time           1.08 seconds
      cpu time            0.18 second
```

Output 1. Sample Log File

In this log file you can see there is a title that contains the word “The SAS System”. The SAS codes are preceded by a number in the line. The other contents in the log file are not preceded by a number. Regular expression can be used to extract the SAS codes.

This script is run in the folder where the log files are located. The command line arguments are the log files. Wildcard is allowed. A pipe is used to read the file, find the lines that start with numbers, remove the lines that contain the words “The SAS System’, and remove the leading numbers. A SAS program is created that contains the output from the pipe.

```
#!/usr/bin/bash
for file in $*
do
NAME=$(echo "$file" | cut -f 1 -d '.')
EXT=.sas
    cat $file | grep  ^[[:digit:]]  \
       | sed '/The SAS System/d' \
       | sed 's/^[0-9]*//g' > $NAME$EXT
done
```

CHKXPT.SH

When you see a folder of SAS transport files, the first question that may come to mind is: how were these transport files generated? There are different ways to create transport files in SAS. A paper about this topic is in reference [4].

If the file is a V5 transport file generated with the XPORT engine, the first line of the file starts with the following characters:
HEADER RECORD*******LIBRARY HEADER RECORD!!

If the file is a V8 transport file generated with the SAS autocall macro loc2xpt.sas, the first line of the file starts with the following characters:
HEADER RECORD*******LIBV8   LIBRARY HEADER RECORD!!

If the file is generated with PROC CPORT without the option nocompress, the first line of the file starts with the following characters:
COMPRESSED **COMPRESSED** **COMPRESSED** **COMPRESSED** **COMPRESSED

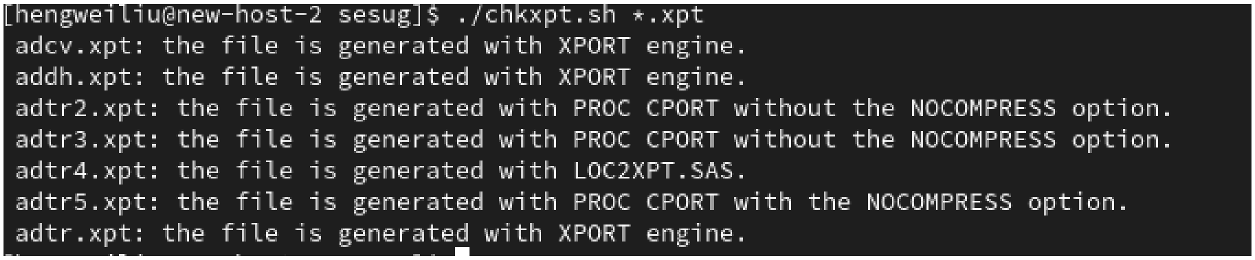
If the file is generated with PROC CPORT with the option nocompress, the first line of the file starts with the following characters:
LIB CONTROL

A shell script can be set up to read the transport files and let the user know how the transport files were created. The user can then proceed to convert the transport files into SAS datasets accordingly. This script is run in the folder where the transport files are located. The command line arguments are the transport files. Wildcard is allowed. The command head -c is used to get the first few bytes from the file. These bytes are compared with certain strings to determine how the transport files were generated.

```
#!/usr/bin/ksh
for file in $*
do
BYTE27=$(head -c 27 $file)
BYTE25=$(head -c 25 $file)
BYTE14=$(head -c 14 $file)
```

```
BYTE11=$(head -c 11 $file)

if [[ $(echo $BYTE27)  == "HEADER RECORD*******LIBRARY" ]]
then
    echo " $file: the file is generated with XPORT engine."
elif [[ $(echo $BYTE25) == 'HEADER RECORD*******LIBV8' ]]
then
    echo " $file: the file is generated with LOC2XPT.SAS."
elif [[ $(echo $BYTE14) == '**COMPRESSED**' ]]
then
    echo " $file: the file is generated with PROC CPORT \
without the NOCOMPRESS option."
elif [[ $(echo $BYTE11) == 'LIB CONTROL' ]]
then
    echo " $file: the file is generated with PROC CPORT \
with the NOCOMPRESS option."
else
    echo " $file: I cannot decide how the file is generated."
fi
done
```

The Display 3 shows a sample output.



```
[hengweiliu@new-host-2 sesug]$ ./chkxpt.sh *.xpt
 adcv.xpt: the file is generated with XPORT engine.
 addh.xpt: the file is generated with XPORT engine.
 adtr2.xpt: the file is generated with PROC CPORT without the NOCOMPRESS option.
 adtr3.xpt: the file is generated with PROC CPORT without the NOCOMPRESS option.
 adtr4.xpt: the file is generated with LOC2XPT.SAS.
 adtr5.xpt: the file is generated with PROC CPORT with the NOCOMPRESS option.
 adtr.xpt: the file is generated with XPORT engine.
```

Display 3. Sample Output of the Script chkxpt.sh

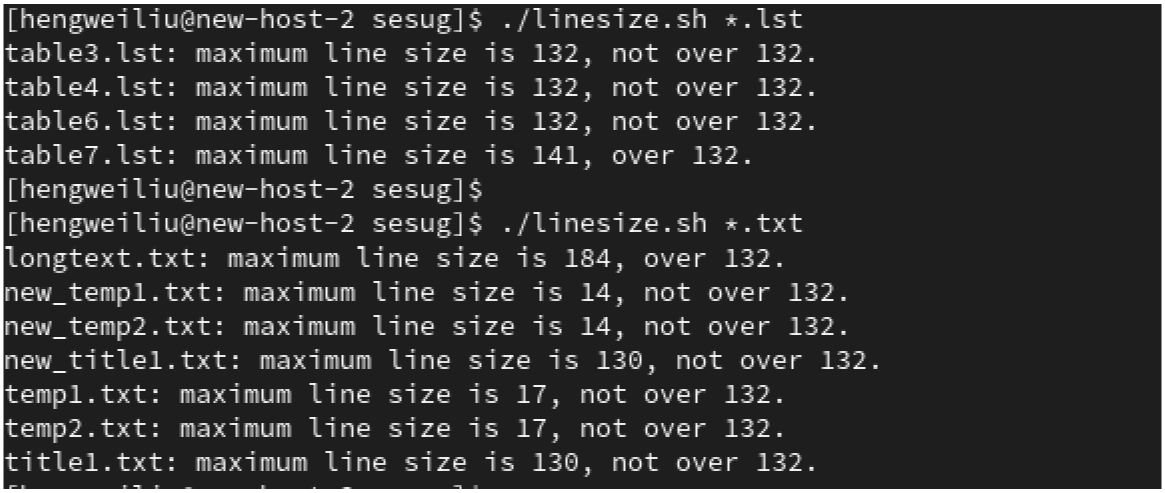
LINESIZE.SH

Some SAS programs read a text file with extension lst or txt. Suppose in the SAS options we have LINESIZE=132. If the input text file has some lines with length greater than 132, that line will be truncated when read in by SAS. It is important to check the input text file to see if there is any line with length greater than 132. This can be done through a script.

This script is run in the folder where the text files are located. The command line arguments are the names of text files. Wildcard is allowed. The script uses awk ‘{print length}’ to find the length of all the lines in the input file and uses sort and tail command to find the maximum length. This maximum length is compared with the value of the SAS option LINESIZE.

```
#!/usr/bin/bash

linesize=132
for file in $*
do
  maximum_length=$( awk '{print length}' $file  | sort -n | tail -1 )
  if [[ $maximum_length -gt $linesize ]]
  then
    echo "$file: maximum line size is $maximum_length, over $linesize."
  else
    echo "$file: maximum line size is $maximum_length, not over $linesize."
  fi
done
```

The Display 4 shows a sample output.



```
[hengweiliu@new-host-2 sesug]$ ./linesize.sh *.lst
table3.lst: maximum line size is 132, not over 132.
table4.lst: maximum line size is 132, not over 132.
table6.lst: maximum line size is 132, not over 132.
table7.lst: maximum line size is 141, over 132.
[hengweiliu@new-host-2 sesug]$
[hengweiliu@new-host-2 sesug]$ ./linesize.sh *.txt
longtext.txt: maximum line size is 184, over 132.
new_temp1.txt: maximum line size is 14, not over 132.
new_temp2.txt: maximum line size is 14, not over 132.
new_title1.txt: maximum line size is 130, not over 132.
temp1.txt: maximum line size is 17, not over 132.
temp2.txt: maximum line size is 17, not over 132.
title1.txt: maximum line size is 130, not over 132.
```

Display 4. Sample Output of the Script linesize.sh

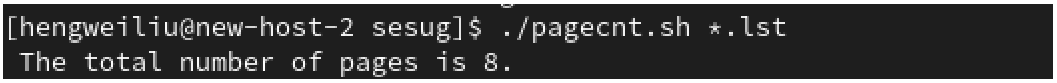
PAGECNT.SH

In a clinical study there can be hundreds of LST files generated for tables and listings. Sometimes these files are printed out in hard copy. It is of interest before printing to know how many pages there are in all those files combined. This can be quickly calculated by a script.

This script is run the folder where the LST files are located. The command line arguments are the names of the LST files. Wildcard is allowed. We assume that the LST files are generated with the SAS option PAGESIZE=52. We use the awk to get the total number of rows in each file. The number of rows is divided by 52. The result is rounded up to get the number of pages in each file. The total number of pages is the sum of the number of pages in each individual file.

```
#!/usr/bin/ksh
PAGE_COUNT=0

for file in $*
do
  NUM_ROWS=$(cat $file | awk 'BEGIN {i=0} {i++;} END {print i}')
  NUM_PAGES=$( awk -v pagesize=52 -v num_rows="$NUM_ROWS" \
'BEGIN {print (num_rows/pagesize)}' )
  ROUND_UP=$( echo "$NUM_PAGES" | awk 'function ceiling(val)
       {
         return (val==int(val))?val:int(val)+1
       }
       {
         printf "%d", ceiling($1)
       }')

 PAGE_COUNT=$(awk -v page_count="$PAGE_COUNT" -v round_up="$ROUND_UP" \
'BEGIN {print (page_count+round_up)}')
done

echo " The total number of pages is $PAGE_COUNT."
```

The Display 5 shows a sample output.



```
[hengweiliu@new-host-2 sesug]$ ./pagecnt.sh *.lst
 The total number of pages is 8.
```

Display 5. Sample Output of the Script pagecnt.sh

TOC.SH

In this script we would like to show how to use the trap command.
Suppose there are some tables in LST format in a folder. We would like to read the title lines from those LST files and create a table of contents (TOC). The title line is the word “Table”, followed by the table number and the table title. We assume that the table number can have four levels separated by a dot, each level being a number from 1 to 99, e.g., 14.1.2, 14.2.1.3.

We set up a script for this task. It is run in the folder where the LST files are located. There is no command line argument. We use regular expression “Table [1-9]” to find the title lines and put them in a file called toc_temp1. A sample of toc_temp1 is shown in output 2. From the file toc_temp1, we extract the table number and add a padding zero if the number in between the dots is less than 10. These new table numbers are put into a file called toc_temp2. A sample of toc_temp2 is shown in output 3.

Now we paste the toc_temp2 and toc_temp1 together and create a file called toc_temp3. A sample of toc_temp3 is shown in output 4. The reason we create this toc_temp3 is that we can sort this file to get the correct order of the tables. Without this step, the order of the tables may not be correct, e.g., the table 14.1.11 will be displayed before the table 14.1.2. After we sort the file toc_temp3, we remove the table numbers with padding zeros and print out a TOC. A sample of the TOC is shown in output 5.

```
Table 14.1.2 Demographics and Baseline Characteristics
Table 14.1.1 Disposition
Table 14.1.11 Vital Signs for Cohort 1
Table 14.1.12 Vital Signs for Cohort 2
```

Output 2. A Sample of toc_temp1

```
14010200
14010100
14011100
14011200
```

Output 3. A Sample of toc_temp2

```
14010200 Table 14.1.2 Demographics and Baseline Characteristics
14010100 Table 14.1.1 Disposition
14011100 Table 14.1.11 Vital Signs for Cohort 1
14011200 Table 14.1.12 Vital Signs for Cohort 2
```

Output 4. A Sample of toc_temp3

```
 Table 14.1.1 Disposition
 Table 14.1.2 Demographics and Baseline Characteristics
 Table 14.1.11 Vital Signs for Cohort 1
 Table 14.1.12 Vital Signs for Cohort 2
```

Output 5. A Sample of toc_final

In this script we create quite a few temporary files, the toc_temp1, toc_temp2 and toc_temp3. These files need to be removed after we run the script. This is where we need the trap command. The trap command allows you to catch signals and execute code when they occur. In our script the signal is EXIT. The script will delete the files toc_temp* on exit.

```
#!/bin/ksh
trap "rm -f toc_temp*" EXIT
for filename in $(ls *.lst); do
print $(grep "Table [1-9]" $filename) >> toc_temp1
done

awk '{print  $2}' toc_temp1 \
| awk -F '.' '{ printf "%02i%02i%02i%02i\n", $1,$2,$3,$4}' > toc_temp2

paste -d ' ' toc_temp2 toc_temp1 > toc_temp3
sort toc_temp3 | awk '{$1=""}1' > toc_final
```

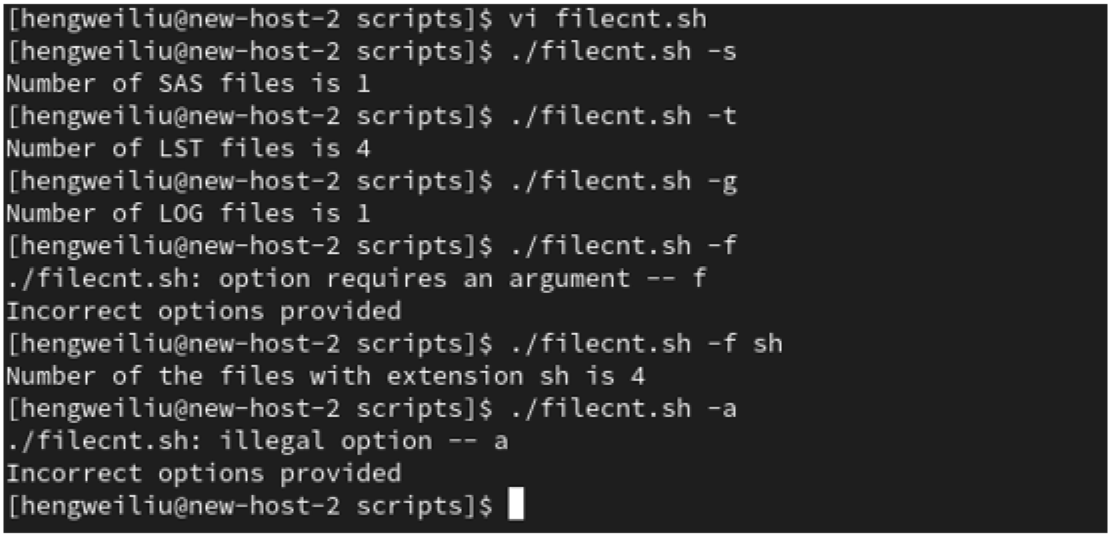
FILECNT.SH

In this script we will show how to use the bash builtin getopts. This builtin can parse the command line arguments to your script. Suppose you want to count the number of files with a certain extension in a folder. You are especially interested in the files with extension sas, lst and log. But you would like the script to be able to count the files with other extensions.

We use the getopts builtin to do this. The command line argument -s, -t and -g are used to count the number of SAS, LST and LOG files respectively. To count files of other extensions, the user needs to use the argument -f followed by the extension.

```
#!/usr/bin/bash
while getopts "stgf:" OPTION; do
    case $OPTION in
    s)
       echo "Number of SAS files is $(ls *.sas | wc -l)"
       ;;
    t)
       echo "Number of LST files is $(ls *.lst | wc -l)"
       ;;
    g)
       echo "Number of LOG files is $(ls *.log | wc -l)"
       ;;
    f)
       EXT=$OPTARG
       echo "Number of the files with extension $EXT is $(ls *.$EXT | wc -l)"
       ;;
    *)
       echo "Incorrect options provided"
       exit 1
       ;;
     esac
done
```

The display 6 shows sample output.



```
[hengweiliu@new-host-2 scripts]$ vi filecnt.sh
[hengweiliu@new-host-2 scripts]$ ./filecnt.sh -s
Number of SAS files is 1
[hengweiliu@new-host-2 scripts]$ ./filecnt.sh -t
Number of LST files is 4
[hengweiliu@new-host-2 scripts]$ ./filecnt.sh -g
Number of LOG files is 1
[hengweiliu@new-host-2 scripts]$ ./filecnt.sh -f
./filecnt.sh: option requires an argument -- f
Incorrect options provided
[hengweiliu@new-host-2 scripts]$ ./filecnt.sh -f sh
Number of the files with extension sh is 4
[hengweiliu@new-host-2 scripts]$ ./filecnt.sh -a
./filecnt.sh: illegal option -- a
Incorrect options provided
[hengweiliu@new-host-2 scripts]$ █
```

Display 6. Sample Output of the Script filecnt.sh

CONCLUSION

The shell scripting is a wonderful tool. SAS programmers working in Unix/Linux environment can utilize the tool to find interesting and efficient ways of doing things and make the work more enjoyable.

REFERENCES

[1] Newbam, Cameron; Rosenblatt, Bill. Learning the Bash Shell. Second edition. 1998. O'Reilly.
[2] Rosenblatt, Bill; Robbins, Arnold. Learning the Korn Shell. Second Edition. 2002, O'Reilly.
[3] Study Data Technical Conformance Guide, FDA, June 2021, available at
Study Data Technical Conformance Guide_Working Draft_4.6 (fda.gov)
[4] Adams, John. Those pesky SAS v5 transport files, what's inside? Paper AD08, PharmaSUG 2009

ACKNOWLEDGEMENT

The author thanks the management at Hengrui USA for support.

CONTACT INFORMATION

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

Hengwei Liu
Hengrui USA
400 Alexander Park
Princeton, NJ 08540
Hengwei_liu@yahoo.com