AWK! Was it something I SED?

or

How Basic Unix Tools can Save Time and Money During a PeopleSoft 7.5 to 8SP1 Upgrade

Harold Hooper

Introduction

In preparation for supporting their new PeopleSoft Internet Architecture (PIA), PeopleSoft needed to rename table fields within their products to remove pound signs (#). While this new requirement had little impact on PeopleSoft's delivered report programs written in SQRIBE Technology's proprietary programming language, SQR - what impact would it have on the tens, if not hundreds, customized SQRs that had been written in the preceding years? If your shop is using completely vanilla PeopleSoft then you need not read any further, but for the hundreds of companies whose business needs far outstripped PeopleSoft's delivered reporting capabilities, read on….

Background

Much to PeopleSoft's credit, they provided a file that contained the old and new versions of the field names (convert-HR75.inf) which reduced the need for manual examination and comparison to determine the field changes. The question then became how could we efficiently search and replace potentially hundreds of instances of field names within SQRs without having to resort to using extremely labor intensive and expensive developer resources to manually examine each program and make the changes? When evaluating the tools available on a Unix platform, the obvious answer was SED, Unix's powerful script editing language. For those unfamiliar with the capabilities of SED, it allows global search and replace functionality within files being processed.

Solution

Since SED provides search and replace capabilities for exact phrases, a sampling of our customized SQRs and SQCs (SQR copybooks) showed there were six common usages that needed to be converted:

record.#fieldname converted to record.fieldname
#fieldname converted to fieldname
,#fieldname converted to ,fieldname
#fieldname, converted to fieldname,
(#fieldname converted to (fieldname
&#fieldname converted to &fieldname

Those familiar with SQR programming conventions will recognize instances in which each of these structures would come into play. Compounding the complexity was the fact that the SRQ programming language is case insensitive so we had to check for instances of uppercase, lowercase and mixed case.

While SED is an extremely powerful tool, checking every combination and permutation required that we use AWK, another powerful Unix tool, to automate the creation of SED scripts that could be iteratively executed in Unix script command procedures to implement the changes in mass. AWK is a subset of the C programming language and follows the same general syntactical rules.

Below is the first in a series of AWK programs which read an input file and produced a file that could be processed by a SED script.

sed_it01.awk

```
1         {
2          found = "N"
3          compare = sprintf("%c",61)
4          slash = sprintf("%c",92)
5          size = length($0)
6          printf("s/")
7          printf("%c",slash)
8          printf(".")
9          for ( i=1; i<=size; i++)
10            {
11            if (found == "N" && substr($0,i,1) != compare)
12               {
13                printf("%c",substr($0,i,1))
14              }
15             else
16              {
17               found = "Y"
18             }
19            if (substr($0,i,1) == compare)
20              {
21               printf("/")
22               printf("%c",slash)
23               printf(".")
24             }
25             if (found == "Y" && substr($0,i,1) != compare)
26            {
27            printf("%c",substr($0,i,1))
28            }
29       }
30   printf ("/g\n")
31    }
```

In a nutshell, this program reads each line from an input file and first prints the characters “s/\.”, prints out each character until it finds an equal sign (=), prints the characters “/\.”, prints the remaining characters in the input string and finally prints the characters “/g”. An example of the final output formatted in SED syntax for each of the six AWK programs using uppercase, lowercase and mixed case:

UPPERCASE

```
s/\.ACCOUNT#_UK/\.ACCOUNT_NUM_UK/g
s/ ACCOUNT#_UK/ ACCOUNT_NUM_UK/g
s/,ACCOUNT#_UK/,ACCOUNT_NUM_UK/g
s/ACCOUNT#_UK,/ACCOUNT_NUM_UK,/g
s/(ACCOUNT#_UK/(ACCOUNT_NUM_UK/g
s/&ACCOUNT#_UK/&ACCOUNT_NUM_UK/g
```

lowercase

```
s/\.account#_uk/\.account_num_uk/g
s/ account#_uk/ account_num_uk/g
s/,account#_uk/,account_num_uk/g
```

```
s/account#_uk,/account_num_uk,/g
s/(account#_uk/(account_num_uk/g
s/&account#_uk/&account_num_uk/g

Mixed Case
s/\.Account#_Uk /\.Account_Num_Uk//g
s/ Account#_Uk/ Account_Num_Uk/g
s/,Account#_Uk/,Account_Num_Uk/g
s/Account#_Uk,/Account_Num_Uk,/g
s/(Account#_Uk/(Account_Num_Uk/g
s/&Account#_Uk/&Account_Num_Uk/g
```

For each individual field name that needed to be changed there were 18 SED scripts to cover potential spelling and syntactical combinations.

Once these multiple scripts had been produced we used a Unix script command procedure to iteratively execute the scripts and produce output that we could test for expected results. Below is an excerpt of the Unix script that performed the conversion:

Lines 1 and 2 set the script variables

```
1       export SERVER_DIR= /your-directory-path /sqrs/server
2       export WORK_DIR= /your-work-directory-path
```

Line 3 sets a date variable

```
3       PROC_DATE=`date +"%y%m%d_%H%M%S"`
```

Line 4 writes data to sysout (the screen)

```
4       echo "Starting AWK Parse at" `date +"%r"`
```

Line 5 creates a file on the old field names

```
5       awk -f $WORK_DIR/format_em.awk $WORK_DIR/convert-HR75.inf >
$WORK_DIR/parsed_HR75.dat
```

Line 6 writes data to sysout (the screen)

```
6       echo "Starting SERVER fgrep at" `date +"%r"`
```

Line 7 uses the Unix utility fgrep to create a file containing names of SQRs that have fields that need to be changed.

```
7       fgrep -f $WORK_DIR/parsed_HR75.dat $SERVER_DIR/*.* >
$WORK_DIR/find_sqr_changes.rpt
```

Line 8 writes data to sysout (the screen)

```
8       echo "Creating mass SED file" `date +"%r"`
```

Lines 9 through 14 uses AWK to create the SED scripts

```
9       awk -f $WORK_DIR/sed_it01.awk $WORK_DIR/convert-HR75.inf >
$WORK_DIR/sed_awked01.dat
10      awk -f $WORK_DIR/sed_it02.awk $WORK_DIR/convert-HR75.inf >
$WORK_DIR/sed_awked02.dat
11      awk -f $WORK_DIR/sed_it03.awk $WORK_DIR/convert-HR75.inf >
$WORK_DIR/sed_awked03.dat
12      awk -f $WORK_DIR/sed_it04.awk $WORK_DIR/convert-HR75.inf >
$WORK_DIR/sed_awked04.dat
```

```
13      awk -f $WORK_DIR/sed_it05.awk $WORK_DIR/convert-HR75.inf >
$WORK_DIR/sed_awked05.dat
14      awk -f $WORK_DIR/sed_it06.awk $WORK_DIR/convert-HR75.inf >
$WORK_DIR/sed_awked06.dat
```

Lines 15 through 20 concatenates the SED scripts into a single file

```
15      cat $WORK_DIR/sed_awked01.dat  > $WORK_DIR/sed_awked.dat
16      cat $WORK_DIR/sed_awked02.dat >> $WORK_DIR/sed_awked.dat
17      cat $WORK_DIR/sed_awked03.dat >> $WORK_DIR/sed_awked.dat
18      cat $WORK_DIR/sed_awked04.dat >> $WORK_DIR/sed_awked.dat
19      cat $WORK_DIR/sed_awked05.dat >> $WORK_DIR/sed_awked.dat
20      cat $WORK_DIR/sed_awked06.dat >> $WORK_DIR/sed_awked.dat
```

Lines 21 through 26 deletes the temporary working files

```
21      rm -f $WORK_DIR/sed_awked01.dat
22      rm -f $WORK_DIR/sed_awked02.dat
23      rm -f $WORK_DIR/sed_awked03.dat
24      rm -f $WORK_DIR/sed_awked04.dat
25      rm -f $WORK_DIR/sed_awked05.dat
26      rm -f $WORK_DIR/sed_awked06.dat
```

Line 27 writes data to sysout (the screen)

```
27      echo "Creating file containing the directory listings" `date +"%r"`
```

Line 28 creates file that contains a listing of SQR names to be examined for changes

```
28      ls $SERVER_DIR > $WORK_DIR/server.dir
```

Line 29 writes data to sysout (the screen)

```
29      echo "Starting SERVER file conversion at" `date +"%r"`
```

Lines 30 through 36 iteratively steps through the file of SQR names and processes each line of the SED script against each SQR.

```
30      while read FILENAME
31      do

        Lines 32 and 33 export new variable names to the Unix kernel
32        export IN=$SERVER_DIR/$FILENAME
33        export OUT=$SERVER_DIR/converted/$FILENAME
```

Line 34 writes data to sysout (the screen)

```
34        echo "CONVERTING "$IN
```

Line 35 invokes Unix SED to process the AWK produced scripts and outputs the newly converted SQR

```
35        sed -f $WORK_DIR/sed_awked.dat $IN > $OUT
36      done < $WORK_DIR/server.dir
```

Line 37 writes data to sysout (the screen)

```
37      echo "Starting Converted SERVER fgrep at" `date +"%r"`
```

Line 38 writes data to sysout (the screen)

```
38      echo "Script finished at" `date +"%r"`
```

Knowing the audience for this document is technical I will admit that this solution was neither the most elegant nor the most efficient, but these simple scripts were very effective. With over 400 customized SQRs and SQCs we estimate it was over 99% accurate and took only 24 staff hours to create and test. The project manager estimated these Unix processes saved over 5 staff weeks of developer time and helped bring the project in on-time and on-budget.

In summary, an in-depth knowledge of basic Unix tools, applied creatively, can greatly reduce time and expense associated with some upgrades. An additional benefit was a reduced margin of typographical errors introduced by developers manually updating each SCR.

One final thought, remember VI is your friend.