Pharmasug 2012 - Paper CC32

Using File Modification Date Comparisons to Alert Study Teams of Potential SAS Program Revisions between Shared Folders

Stephen Hunt, ICON Clinical Research, Redwood City, CA
Brian Fairfield-Carter, ICON Clinical Research, Redwood City, CA

ABSTRACT

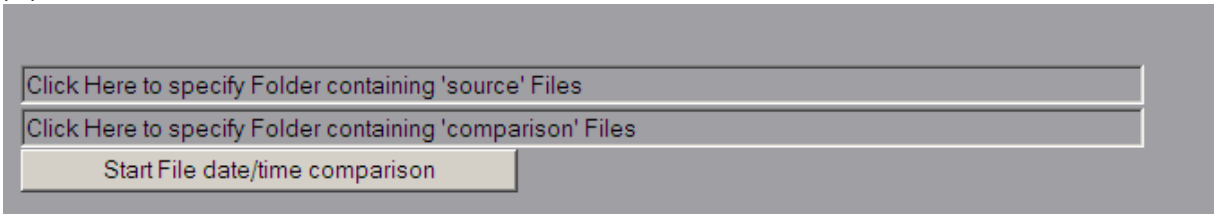
In the absence of version control, communicating changes made to programs between development and production areas (e.g., modifications to a select few blinded-area programs that must be copied over to an unblinded area for re-delivery) can easily result in 'stale' code and outdated output without absolutely perfect coordination across the entire study team. To compound things, there indubitably exists an inverse relationship between size of programming teams and the timely and accurate communication within them. In such circumstances, an unblinded programmer or statistician must often rely on their own manual verification of the state of files to be run in production in situations of concurrent development (i.e., since members of a blinded team would be unable to verify or directly compare against contents of an unblinded folder themselves). Therefore, comparing file modification dates programmatically may be the best, first step towards evaluating consistency across folders intended to share the same production code (such as between blinded and unblinded study teams).

INTRODUCTION

This paper demonstrates 3 methods for programmatically comparing SAS® program modification date/time stamps between 2 study folders in order assure that all production environment output is up-to-date with any changes that have occurred in development. What follows are a series of demonstrations on how to obtain both Windows and UNIX modification dates dynamically for all SAS files in 2 comparator folders. Each method is intended to be simple to apply for unblinded/recipient programmers who wish to compare large numbers of files across multiple study folders in either the operating system/platform.

APPLICATION #1 - WINDOWS (OR UNIX VIA SAMBA):

The first instance to be demonstrated involves an HTA (HTML Application) written with JavaScript and VBScript embedded within an HTML document. The application GUI itself is extremely rudimentary and consists solely of 2 browseforfolder input boxes and a clickable button to initiate the comparison script once the folder objects are populated:



Display 1. View of Simple Application GUI

The BrowseForFolder method contains an element for pre-specifying the initial folder for browsing following clicking upon the input text box (bolded section below indicates the specific server-path to begin browsing; users will need to specify their own):

```
set Folder=shell.browseforfolder(0,"Specify Folder containing 'comparison' files and click
'ok'.",&h0010,"\\pa-legacysas-01\studies")
```

Once folders to compare are specified, the key elements performed by the application include (a) writing out a text file to be populated with any/all differences found via VBScripts filesystemobject createtextfile method and (b) obtaining the 'datelastmodified' file property via the following code:

```
Set file1=fso.GetFile("filename.sas")
moddate1=file.DateLastModified
```

The appearance of the differences text file opened within the ubiquitous notepad Windows application (whether empty, thus signifying no difference found, or populated with differences) marks the completion of the application's processing. Differences are highlighted by program name by including modification dates within both folders and presenting the file comparison in the direction of the difference (e.g., "`File test.sas has been modified more recently(2/25/2011 11:28:33 AM) in the destination vs. source folder (9/7/2011 2:52:45 PM).`"). The entirety of the code used for this simple application is copied below:

```
<HEAD>

<script language="VBScript" event=onclick for=oldfoldtext>

Dim inlog, filehandle, outlog, fso, WshShell, Filename_, shell

Set fso=CreateObject("Scripting.FileSystemObject")
Set WshShell=CreateObject("Wscript.Shell")
set shell=createobject("shell.application")
set Folder=shell.browseforfolder(0,"Specify ‘source' folder & click 'ok'.",&h0010,"C:\")
on error resume next

txtPath = Folder.ParentFolder.ParseName(Folder.Title).Path
txtPath=replace(txtpath,"(","{(}")
txtPath=replace(txtpath,")","{)}")

if txtPath="" then
   wscript.quit
end if

document.getElementById("oldfoldtext").innerText=txtPath
</script>

<script language="VBScript" event=onclick for=newfoldtext>

Dim inlog, filehandle, outlog, fso, WshShell, Filename_, shell

Set fso=CreateObject("Scripting.FileSystemObject")
Set WshShell=CreateObject("Wscript.Shell")
set shell=createobject("shell.application")
set Folder=shell.browseforfolder(0,"Specify ‘comparison' folder & click 'ok'.",&h0010,"C:\")
on error resume next

txtPath = Folder.ParentFolder.ParseName(Folder.Title).Path
txtPath=replace(txtpath,"(","{(}")
txtPath=replace(txtpath,")","{)}")

if txtPath="" then
   wscript.quit
end if

document.getElementById("newfoldtext").value=txtPath
</script>

<script language="VBScript" event=onclick for=startit>

dim fso
Set fso=CreateObject("Scripting.FileSystemObject")
newfold=document.getElementById("newfoldtext").value
oldfold=document.getElementById("oldfoldtext").value
destfold=newfold

retname=newfold & "\differences_found.txt"

if fso.fileexists(retname)=true then
   fso.deletefile(retname)
end if

set o_f= fso.getfolder(oldfold)
set fc= o_f.files
set retfile=fso.createtextfile(retname,true)

For Each f1 in fc

  file=f1.name

  if fso.getextensionname(ucase(f1.name))="SAS" then

     compfile1=oldfold & "\" & file
     compfile2=newfold & "\" & file

     if fso.fileexists(compfile1)=true and fso.fileexists(compfile2)=true then
        Set oldfile=fso.GetFile(compfile1)
        moddate1=oldfile.DateLastModified

        Set newfile=fso.GetFile(compfile2)
        moddate2=newfile.DateLastModified

        if moddate1>moddate2 then
```

```
            retfile.writeline "File " & file & " has been modified more recently(" & moddate1
                              & ") in the source vs destination folder (" & moddate2 & ")."
         end if
         if moddate2>moddate1 then
            retfile.writeline "File " & file & " has been modified more recently(" & moddate1
                              & ") in the destination vs source folder (" & moddate2 & ")."
         end if

      end if
   end if
next

retfile.close

Set wshell=CreateObject("Wscript.Shell")
wshell.run retname

window.close()

</script>

</head>

<body>

<br>
<input type=text value="Click Here to specify Folder containing 'source' Files" id=oldfoldtext
size=100></input><br>
<input type=text value="Click Here to specify Folder containing 'comparison' Files" id=newfoldtext
size=100></input><br>
<input type=button value='Start File date/time comparison' id=startit></input>

</body>
```

While the advantages of an HTML application include ease of use and familiarity via a GUI, the primary disadvantage of the above seems to be speed. Particularly when working in the context of a local instantiation of the application while referencing network folders, the embedded comparison script works via accessing file properties over the WAN (Wide Area Network), which, depending on connection speed and number of files to be evaluated, can take a considerable amount of time to process. Although by default the application only compares all files with a ".sas" extension between folders, it could easily be adapted to check other file types that contribute to primary production output (e.g., input source data sets or spreadsheets).

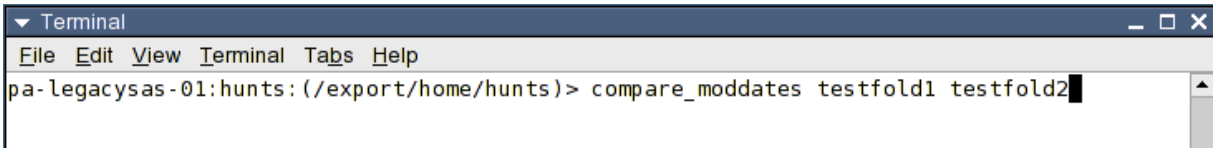
APPLICATION #2 - A PERL SCRIPT EXECUTED WITHIN UNIX:

In contrast to the first example, which demonstrated a HTML Application to be run within the Windows platform, the next applies a few lines of Perl code submitted as a script via the UNIX command line to compare modification dates between UNIX folders. While the inner workings and the instantiation method of the script (i.e., via command line) are a bit different, the general concept and output method are the same. Again, the code in its entirety is included below:

```
#!/usr/local/bin/perl

#################################
### compare_moddates.pl
###
### This script compares the file modification dates for matching files in 2 subdirectories
### specified as arguments.
###
### example call: perl compare_moddates.pl /pub/studies/folder1 /pub/studies/folder2
#################################;

use File::stat;
use Time::localtime;

my $folder1= "$ARGV[0]";
my $folder2= "$ARGV[1]";

if (-e '$folder2/diffs.txt')
{
  system("rm $folder2/diffs.txt");
}

opendir (DIR, $folder2) or die $!;

open(OUTFILE,">$folder2/diffs.txt");
```

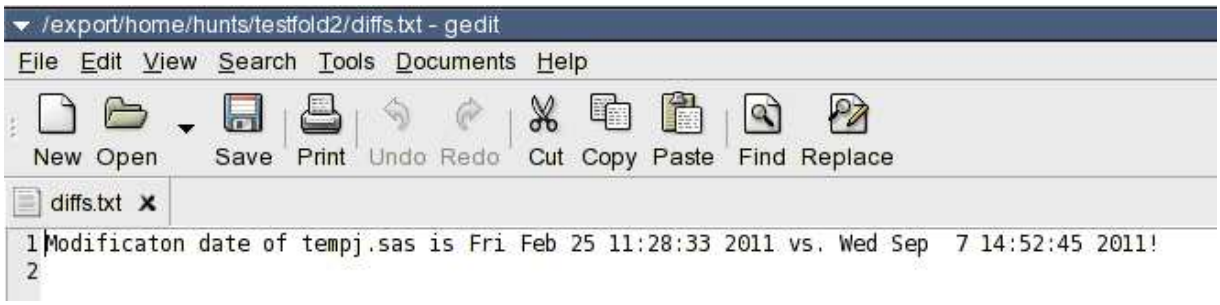
```
while (my $file = readdir(DIR))
{
   next unless ($file=~m/\.sas$/);

   my $file1="$folder1/$file";
   my $file2="$folder2/$file";

   next unless (-e $file1);
   my $date1=ctime(stat($file1)->mtime);

   next unless (-e $file2);
   my $date2=ctime(stat($file2)->mtime);

   next unless ($date1 ne $date2);

   print OUTFILE "Modificaton date of $file is $date1 vs. $date2!\n";
}

closedir(DIR);

close(OUTFILE);

system("gedit $folder2/diffs.txt");
```

As with the first application, the comparison folder is used as a baseline to generate a file list for comparing modification dates between programs that exist in both folders, and any/all differences found are output to a text file, which in this case is opened within the UNIX terminal window by gedit, a standard UNIX text editor application.



Display 2. Terminal Screenshot of Command-Line Call of Script

In the command call above, the 'compare_moddates' actually is an alias of 'perl compare_moddates.pl', which is the actual script name and method of calling Perl at the command line, while testfold1 and testfold2 are subdirectories immediately below the parent directory shown at the command line. Besides individual folder names, relative pathnames (e.g., ../../testfold3) or full path names (/export/home/hunts/testfold1) are permissible.

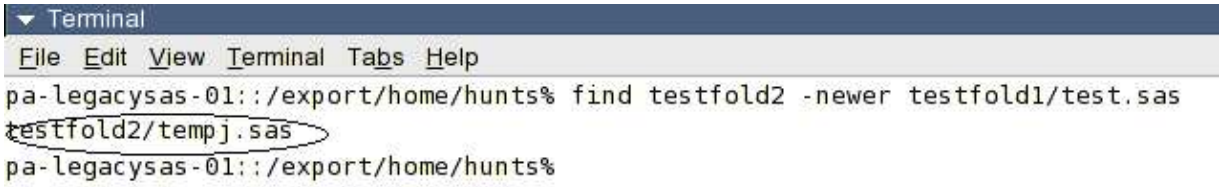


Display 3. Screenshot of Output Generated

The difference in processing speed is distinct advantage of the UNIX-side script over the HTA: Entire directories with hundreds of programs can be compared in just a few seconds. This of course will vary depending upon the amount of memory allocated on your particular UNIX server.

APPLICATION #3 - A SAS PROGRAM WITHIN UNIX:

SAS within UNIX does not include a function for determining modification dates of files (probably reflecting the fact that UNIX itself is rather limited in this fashion). However, UNIX does provide a similar comparison functionality within the confines of the 'find' command (type 'man find' at the UNIX command line for details). In short, 'find' has the ability to compare all files in a folder against one specific comparator file using the '-newer' option:



Display 4. Screenshot Example of Using then '-newer' UNIX Option

While this is useful for discovering more general changes to a folder relative to a static file, it's not quite as useful as a file-by-file comparison (i.e., since not all possible comparison files have an identical date/time stamp in all likelihood, this might miss a change in a source file resulting from a rollback to a previous version, for example). However, by using SAS to pipe out a comparison list of files followed by a macro loop applied to the comparison list, the x-command (or call system) can be using within SAS to direct the UNIX 'find' to generate a file-by-file comparison and accompanying output list of differences. All code used is presented below:

```
%let folder1=%str(/export/home/hunts/testfold1);
%let folder2=%str(/export/home/hunts/testfold2);

DATA _null_;

*** GET A LIST OF THE COMPARITOR SAS FILES AND OUTPUT TO A TEMPORARY FILE.;
call system("ls &folder2 | grep .sas > list2.txt");
run;

DATA folder2;
length pgmname $200;
infile "list2.txt";
input pgmname;
run;

DATA folder2;
set folder2;

c+1;

*** GET THE TOTAL NUMBER OF PROGRAMS.;
call symput('tot',compress(c));
run;

*** CREATE AN EMPTY 'SHELL' FILE FOR CONTAINING ANY DIFFERENCES FOUND.;
x "ls xyz123abc789.sas > &folder2/diffs_sas.txt";

%macro compdate;

%do i=1 %to &tot;

DATA _null_;
set folder2;

if c=&i;
call symput('filename1',"&folder1/"||trim(left(pgmname)));
call symput('filename2',"&folder2/"||trim(left(pgmname)));
run;

x "find &filename1 -newer &filename2 >> &folder2/diffs_sas.txt";

%end;

%mend compdate;

%compdate;

x "gedit &folder2/diffs_sas.txt &";
```

CONCLUSION

In the absence of version control, study teams sharing files between production and development areas have to be both cautious and diligent of development-side modifications impacting concurrent output production. This paper has presented 3 distinct methods for comparing input sources (typically, but not limited to, SAS code) across folders to ensure that potential modifications are programmatically identified for further investigation (e.g., targeted 'diff' utility usage or manual review of individual files) and reconciliation completed satisfactorily prior to output delivery.

REFERENCES

SAS Knowledge Base. "Usage Note 40934: Retrieve file size, create time, and last modified date of an external file" **HTTP://SUPPORT.SAS.COM/KB/40/934.HTML**

CONTACT INFORMATION

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

Author name(s): Stephen Hunt and Brian Fairfield-Carter
Company: ICON Clinical Research
Address: Suite 500, 303 Twin Dolphin Rd.,
City state ZIP: Redwood City, CA 94065
E-mail: Stephen.Hunt@iconplc.com
B.Fairfield-Carter@iconplc.com