PharmaSUG 2018 - Paper AD-04

A Linux Shell Script to Automatically Compare Results of Statistical Analyses

Chen Wang, Shu Zan and Joshua Conrad, Gilead Sciences, Inc. Foster City, CA

ABSTRACT

Double programming is a standard validation method used in statistical programming in the pharmaceutical industry. With this method, production and validation programmers independently generate and compare data sets or statistical analyses outputs to ensure their correctness. While the SAS® COMPARE procedure is readily used to compare data sets, there are no standard tools available to compare the analytic results in form of tables, figures, and listings (TFL). Generation of a text file in parallel with each TFL PDF file allows us to compare using programming tools such as the **diff** command on Linux systems. However, when the number of TFLs increases, it is time consuming to manually write the commands and difficult to track the results. To automate this process, we developed a Linux shell script that automatically retrieves all output files from production and validation directories using regular expressions, performs **diff** comparisons, and concatenates the results into a single text file. Options exist to allow users to select subsets of output files to compare or to ignore case or certain non-critical text when comparing. The output file consists of a summary section and a details section. The former contains information on missing files, timestamp violations and mismatched files; while the latter contains detailed **diff** results. This script provides an efficient way to compare the TFL outputs from double programming and to visualize the comparison results. It greatly reduces the cycle time for production to correct any programming errors, which is especially important when under tight timelines.

INTRODUCTION

In the pharmaceutical industry, the main job responsibility of statistical programmers is to process clinical data, generate analytical results and present them in the form of TFLs. The TFLs are included in the clinical study report and submitted to regulatory agencies as part of the filing package or used for publications. It is essential that the contents and formats of these documents are validated vigorously to ensure accuracy and completeness. Among various validation methods, double programming is thought as the industry standard and widely used (Shiralkar, 2010). Although this method can reduce programming errors to a minimum extent, it requires significant amount of resources. Therefore automation of any step involved is very beneficial. One such step is comparing the production and validation TFL outputs.

Pairs of text files can be compared by machine using the SAS procedure PROC COMPARE or Linux command **diff**. Some other software exists as well, which is well summarized by Fredette (Fredette, 2008). However, these methods are still time consuming, especially when a task involves a large number of TFLs. Ideally, a comparison program should be able to automatically compare all validation and production outputs for a task and document the results in a file which can be easily searched and reviewed. We decided to develop such a program using Linux shell script for the following reasons: 1. our TFL output files are stored on servers running Linux operating systems; 2. the **diff** command can be directly used to compare files; 3. powerful regular expression and text editing tools come with Linux allowing us to pre-process files in a batch mode as needed; 4. a shell script can be run from any location in the system if saved appropriately.

We named our comparison program **diffdir.** In this paper, we introduce the tools and logic that we used to develop this program.

DIRECTORY STRUCTURE AND NAMING CONVENTIONS

We programmed **diffdir** based on certain conventions that we use to name and store TFL files. For example, we can save all production files in a directory called ‘production’ and all validation in a directory called ‘validation’ at the same level. For each TFL, we use SAS programs to generate a pdf file (.pdf) and a text file (.out or .gsf) with the same contents. The names of production tables, figures and listings start with ‘t-’, ‘g-’ and ‘l-’ respectively. Their corresponding validation output bears the same name except that it starts with ‘v-’. An example is provided in Table 1.

TFL type	Production	Validation
`Tables`	`t-ae.out/t-ae.pdf`	`v-t-ae.out/v-t-ae.pdf`
`Listings`	`l-ae.out/l-ae.pdf`	`v-l-ae.out/v-l-ae.pdf`
`Figures`	`g-lab.gsf/g-lab.pdf`	`v-g-lab.gsf/v-g-lab.pdf`

Table 1. TFLs Naming Convention

LINUX COMMANDS USED IN THE COMPARISON PROGRAM

The program **diffdir** consists of a series of Linux commands. For some of them, we list the general syntax and explain their usage briefly with examples. For detailed use, please refer to (Robbins, 2006) or the **man** pages coming with the Linux system. All sample code follows the syntax of BASH, which is a commonly used shell program on Linux based systems. They can be executed either individually in the shell or as part of a shell script. Just note, Linux commands and variables, unlike SAS, are case sensitive. The square brackets in the syntax mean optional, and the ‘#’ sign in the sample code means start of comments.

COMMAND USED TO COMPARE TEXT FILES

The **diff** command is widely used to compare text files line by line and is the core of our comparison program. It uses the following syntax:

```
diff [options] file1 file2
```

It has multiple options to control the mode of comparison. We use two very simple example files here to demonstrate how these options might affect the output. The contents of the two files are shown in Table 2, each consisting of two lines of text with some difference in cases or spaces.

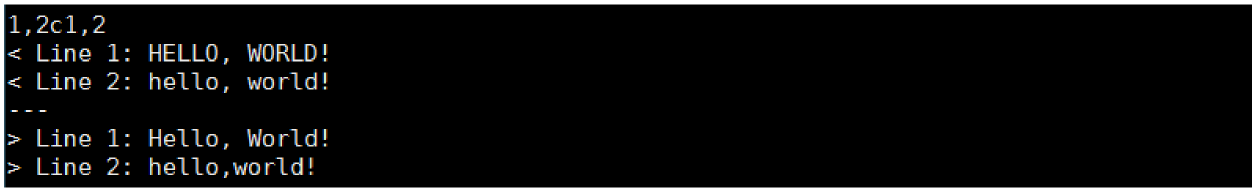
file1	file2

`Line 1: HELLO, WORLD!`
`Line 2: hello, world!`
	`Line 1: Hello, World!`
	`Line 2: hello,world!`

Table 2. Example Text Files

Example #1

```
diff file1 file2
```



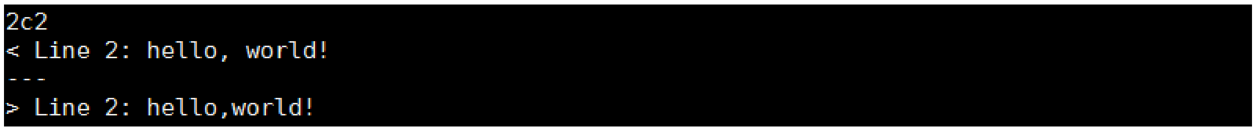
```
1,2c1,2
< Line 1: HELLO, WORLD!
< Line 2: hello, world!
---
> Line 1: Hello, World!
> Line 2: hello,world!
```

Figure 1. Result of diff Command without Options

In example #1, the two files are compared without any option; therefore, all characters are compared. The output is shown in Figure 1. In the output, the first line ‘1,2c1,2’ consists of two line ranges separated by a letter. The line ranges indicate where the difference exists, with the first range from the first file and the second range from the second file. The letter between indicates the operation to be taken to make the two files identical. Here ‘c’ stands for change. Other operations you probably see are ‘a’ (append) and ‘d’ (delete). The lines beginning with ‘<’ or ‘>’ sign display the differing lines.

Example #2

```
diff –i file1 file2
```

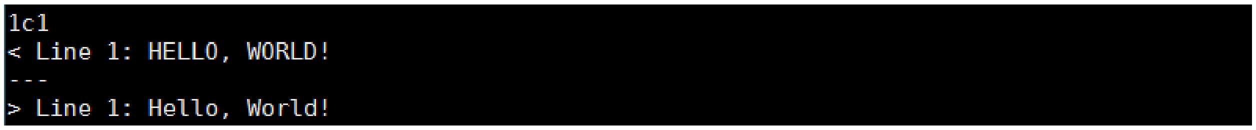


```
2c2
< Line 2: hello, world!
---
> Line 2: hello,world!
```

Figure 2. Result of diff Command Using Option -i

Example #3

```
diff –w file1 file2
```



```
1c1
< Line 1: HELLO, WORLD!
---
> Line 1: Hello, World!
```

Figure 3. Result of diff Command Using Option -w

In example #2, option -i is used to ignore case; therefore, only Line 2 is different, as shown in Figure 2. In example #3, option -w is used to ignore all white space; therefore, only Line 1 is different, as shown in Figure 3. If both options -iw are used, the two files are thought identical and the output is an empty file.

The following example can be used to compare text TFL output files from validation and production:

Example #4

```
diff –w ../production/t-ae.out ../validation/v-t-ae.out
```

For any validation task, we can manually write such a command for each pair of TFL, using appropriate options. Alternatively, we can use the Linux commands introduced thereafter to search file names automatically.

COMMANDS USED TO RETRIEVE FILES AUTOMATICALLY FROM DIRECTORIES

The following example is used to search the validation directory for all TFL files whose names contain certain patterns. It consists of three individual commands separated by a vertical bar. Each component will be explained in more detail.

Example #5

```
ls ../validation | egrep "v-" | egrep  ".*-[tgl]-.*\.(out|gsf)$"
```

Pipeline

The output of Linux commands is printed by default to standard output, usually the display screen. The vertical bar **|** between individual commands is called a pipeline. It functions to redirect the output of the first command to the input of the second command. Therefore, there is no need to create temporary files or variables just to store the mid-step output. Multiple pipelines and commands can be used in each command line. In the above example, files in the validation directory are displayed by **ls** and then filtered by two **egrep** commands to select those with names containing certain patterns.

ls

The command **ls** displays directory contents or lists files with names matching that specified in the command line. If no name is specified, files in the current directory are listed. It adopts the syntax:

```
ls [options] [directory or file names]
```

egrep

The command **egrep** (standing for 'enhanced global regular expression print') searches a file or standard input and prints lines matching a pattern. It uses syntax:

```
egrep [options] pattern [file…]
```

Patterns can be simple text consisting of only literal characters or complex symbolic notations. In example #5, the first **egrep** command is an example for the earlier case, which prints any line containing the string ‘v-’. The pattern in the second **egrep,** ‘.*-[tgl]-.*\.(out|gsf)$’, appears strange. It contains both literal characters and characters with special meanings, which together form so-called regular expressions. For example, the dot ‘.’ matches any character. Such characters are called metacharacters. Table 3 is a list of metacharacters and their matches in the regular expressions supported by **egrep**.

Metacharacter	Matches
`.`	`Any character`
`^`	`Beginning of a line`
`$`	`End of a line`
`*`	`The previous element 0 or more times`
`[]`	`Any character in the brackets`
`|`	`The character(s) before or after`
`\`	`The next character literally`

Table 3. Metacharacters

In example #5, the pattern ‘.*-[tgl]-.*\.(out|gsf)’ matches strings containing these components: any number of any character; a ‘-’; a ‘t’, ‘g’, or ‘l’; a ‘-’; any number of any character; a ‘.’; the string ‘out’ or ‘gsf’ at the end of a line.

Some options are available for **egrep**. A commonly used one is -i, which allows matching by ignoring case.

PROGRAM USED TO EDIT TEXT FILES

Program **sed** is a stream editor that edits file(s) or standard input on a line by line basis. It uses syntax:

```
sed [options] 'script' [input-file]
```

The 'script' can be a file containing commands or that are typed in the command line. The commands are applied to each line of the input text. Below is a simplified example extracted from our **diffdir** program:

Example #6

```
sed 's/_//g' file1
```

The ‘script’ in this example is ‘s/_//g’. The letter ‘s’ stands for substitute, and ‘/_//’ is an expression of ‘/pattern/replacement/’. This ‘script’ searches the pattern ‘_’ and replaces it with an empty string. **sed** supports regular expressions in the pattern. The letter g is a flag to replace all matched instances in each line. Without this flag, only the first matched instance in each line is replaced. The above example removes all underline characters existing in file1 and prints the output to standard output.

OUTPUT AND INPUT REDIRECTION

In addition to the pipeline described earlier, there are also ways to redirect output to and input from file.

The output redirection, denoted by the > sign, is used to redirect the output to a file. When a single > is used, the specified file is created if not existing or overwritten if already exiting. When a double >> is used, the output is appended to the file. Example #7 shows how the output of a **diff** command is redirected to a file to be saved permanently.

Example #7

```
diff -w file1 file2 >> tmpfile
```

The standard input of a command can be typed in or redirected with the < sign from a file or output of other commands. Example #8 below is used in our comparison program, in which two files are first transformed by **sed** (each with two commands separated by a ‘;’ in its ‘script’) and then redirected to **diff**. The output of **diff** in turn is redirected to a file.

Example #8

```
diff -wi <(sed 's/_//g; /Source.*v9.*Output file:/d' file1 ) <(sed 's/_//g;
/Source.*v9.*Output file:/d' file2) >> difftmp
```

FUNCTIONS OF THE COMPARISON PROGRAM

Our resulting program **diffdir** uses the following syntax:

```
diffdir [options] dir1 dir2 [output-file]
```

The two required parameters dir1 and dir2 are the names of the directories to be compared. These directories can be any two accessible in the system and can be provided as relative or full paths. According to our naming convention, if the directory is named 'validation', the program assumes files under have names starting with 'v-'. The name of the output file can be optionally specified as the third argument. The program itself consists of several functions that are called in the main program. We explain these functions as below and provide some sample codes in Appendix I.

PROGRAM OPTIONS

We use command **getopts** to capture the options passed from command line. These options allow users some flexibility to compare only a subset of output, to retrieve filenames from dir1 or from an alternative source (TNF file), or to pre-process the files before comparison. They are summarized in Table 4.

Option	Description	Default
-k	filenames retrieved from TNF file	dir1
-tlga	compare tables, listings, figures or all	tables and listings
-c	compress '_' and 'Source xxx' lines	
-i	ignore case	
-h	usage	

Table 4. Options Available for diffdir

FUNCTION TO RETRIEVE FILENAMES

Function **compd** searches dir1, by default, for filenames identifying them as TFL text output, as shown in example #5. Pattern 'tgl' is replaced by a variable whose value is determined by program options -tlga. If dir1 is named validation, prefix 'v-' in the filename is removed in order to retrieve the corresponding filename from dir2. Option -k requests the function to read filenames from a TNF file. This file contains the key part of names for all TFLs, 't-ae' and 'l-ae', for example. Appropriate suffix (such as '.out' for tables and listings) and prefix (such as 'v-' for validation) are added to make a complete filename. When the filenames for each pair to compare are in place, other functions are called to check the existence of these files, to compare their timestamps and to compare their contents. The code of function **compd** is omitted due to length, but we will provide upon request.

FUNCTION TO COMPARE TWO FILES

Function **compf** accepts two files passed as arguments and compares them using **diff** in ways determined by program options. By default, the comparison ignores white space. Program option -i requests the function to ignore case as well. In our experience, some other information in the TFL files can be ignored too, such as the underline characters and the last line of each page bearing the time stamp and the program source etc. Program option -c requests the function to remove this information before comparison. This last option is very organization specific but can significantly simplify the final output to reduce the reviewing time.

FUNCTION TO FIND AND SUMMARIZE MISSING FILES

Function **nexistf** checks if any TFL is missing in one or both of the directories. If filenames are retrieved from dir1 and the counterparts of any file are not found in dir2, the file is thought as missing from dir2. Or if filenames are retrieved from TNF and any file is not found in either directory, it is thought as missing from that directory. The information of the missing files is summarized and displayed in the output file.

FUNCTION TO COMPARE TIMESTAMPS

Function **compt** uses command **stat** to compare the timestamps of two files. If one file is from validation but the other is not, then we assume the validation file should be generated later than the non-validation file. The information on files violating this rule is also summarized and displayed in the output file.

Just note, although the examples we give here always involve a validation and a production directory, the program can compare any two directories if only the filenames follow our naming conventions. Therefore, this program can also be used to compare different versions of production.

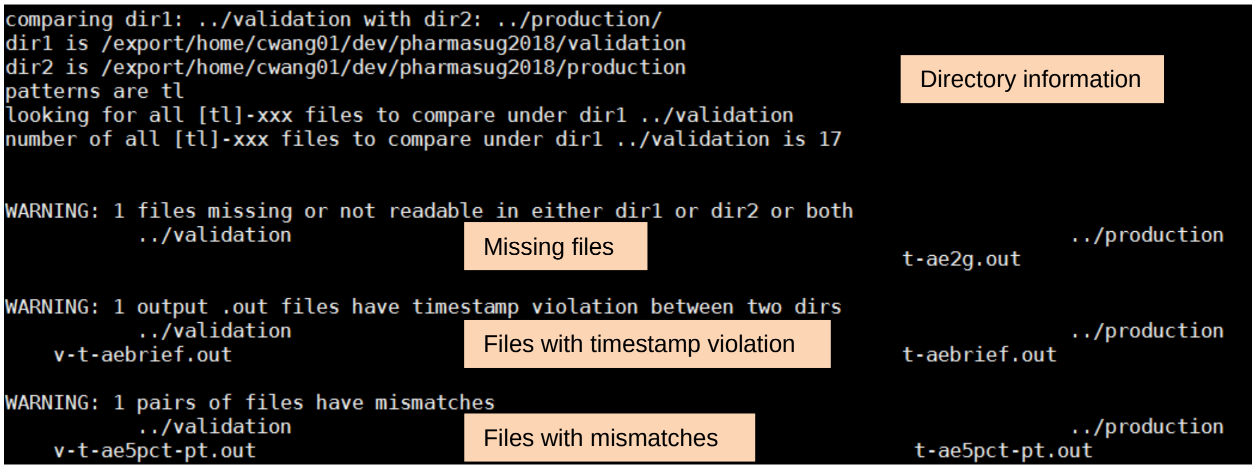
STRUCTURE OF THE OUTPUT

The output of **diffdir** is saved in a file optionally specified as the third argument for the program. If not specified, the program defines a name depending on the options used. For example, if option -c is used, the output file is named 'diffdir-tl-c.lst', indicating tables (t) and listings (l) are compared (by default) and option -c is used. Each option used will be added to the name sequentially.

The output consists of a summary section and a details section. The summary section provides administrative information which might warn the user to take some actions even before looking into the detailed result of comparison. The details section allows users to examine any difference between each pair of TFLs and determine how to correct TFL programs if necessary.

SUMMARY SECTION

The summary section consists of four pieces of information, as shown in Figure 4. The first part lists the full paths of the two directories, the types (patterns) of TFLs compared, the source of filenames (from dir1 or TNF file) and the number of files to compare. The second part is a list of files missing in either directory. If identified, validation or production programmers need to add the missing TFL. The third part lists files that violate the timestamp rule. If identified, validation programmers need to rerun their TFL programs. The last part lists files with mismatches. If identified, programmers need to look at the details section. If there is no missing file, timestamp violation, or mismatches, the corresponding part will not appear.

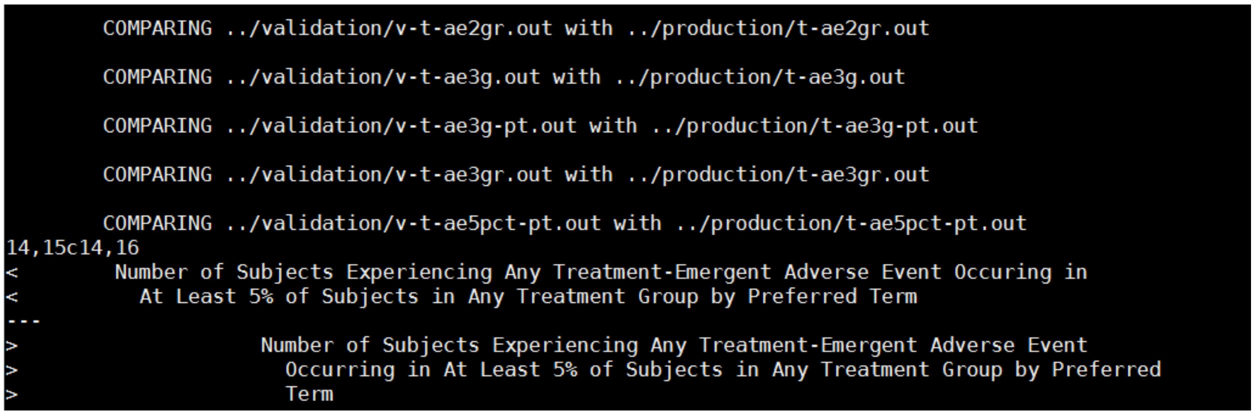


```
comparing dir1: ../validation with dir2: ../production/
dir1 is /export/home/cwang01/dev/pharmasug2018/validation
dir2 is /export/home/cwang01/dev/pharmasug2018/production
patterns are tl
looking for all [tl]-xxx files to compare under dir1 ../validation
number of all [tl]-xxx files to compare under dir1 ../validation is 17


WARNING: 1 files missing or not readable in either dir1 or dir2 or both
           ../validation                                          ../production
                                                   t-ae2g.out

WARNING: 1 output .out files have timestamp violation between two dirs
           ../validation                                          ../production
    v-t-aebrief.out                                t-aebrief.out

WARNING: 1 pairs of files have mismatches
           ../validation                                          ../production
    v-t-ae5pct-pt.out                               t-ae5pct-pt.out
```

Figure 4. The Summary Section of the Output from diffdir

DETAILS SECTION

The detailed section is the **diff** result of each pair of TFL listed one after another, as shown in Figure 5. A subtitle is added to indicate which files are compared, following which is the difference between the two files. If the difference part is missing, it means the two files completely "match" with the options used.



```
        COMPARING ../validation/v-t-ae2gr.out with ../production/t-ae2gr.out

        COMPARING ../validation/v-t-ae3g.out with ../production/t-ae3g.out

        COMPARING ../validation/v-t-ae3g-pt.out with ../production/t-ae3g-pt.out

        COMPARING ../validation/v-t-ae3gr.out with ../production/t-ae3gr.out

        COMPARING ../validation/v-t-ae5pct-pt.out with ../production/t-ae5pct-pt.out
14,15c14,16
<        Number of Subjects Experiencing Any Treatment-Emergent Adverse Event Occuring in
<          At Least 5% of Subjects in Any Treatment Group by Preferred Term
---
>                    Number of Subjects Experiencing Any Treatment-Emergent Adverse Event
>                      Occurring in At Least 5% of Subjects in Any Treatment Group by Preferred
>                      Term
```

Figure 5. The Details Section of the Output from diffdir

CONCLUSION

Although our program **diffdir** is centered on Linux, most of the commands used can find their counterparts in other operating systems, such as Windows. Therefore the same methodology can be adapted to develop a similar program for other systems.

The **diffdir** program has been well accepted and utilized within our programming group. It has saved us a significant amount of manual work and greatly improved our efficiency in identifying programming errors in the validation process. As a result, we can focus our time and energy on modifying the programs for the analyses themselves. This is very important to assure high quality of the statistical analysis results especially when working with limited resources and under tight timelines.

REFERENCES

Fredette, J. (2008). Using Automated File Comparisons to Increase Efficiency and Accuracy in SAS Code Development & Validation. *PharmaSUG Proceedings.* Atlanta, GA.

Robbins, A. (2006). *Unix in a Nutshell, 4th Edition.* O'Reilly Medi, Inc.

Shiralkar, P. (2010). Programming Validation: Perspectives and Strategies. *PharmaSUG Proceedings.* Orlando, FL.

ACKNOWLEDGMENTS

We are very grateful to our wonderful co-workers who have generously shared their ideas and provided their support during the development of the **diffdir** program.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Please contact the authors at:
Chen.Wang@gilead.com, Shu.Zan@gilead.com or Joshua.Conrad@gilead.com

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.

APENDIX I

FUNCTION TO COMPARE TWO FILES

```
compf(){
    fname1=$1
    fname2=$2
    printf "%s\n" "" >> $difftmp
    printf "%s\n" "        COMPARING $_dir1/$(basename $fname1) with
$_dir2/$(basename $fname2)" >> $difftmp
    #if -c remove -- throughout and source line at the end of each page and
append output to $difftmp
    if [ "$_ignorecase" = "1" ]; then
        if [ "$_compress" = "1" ];  then
            diff -wi <(sed 's/_//g; /Source.*v9.*Output file:/d' $fname1 )
<(sed 's/_//g; /Source.*v9.*Output file:/d' $fname2) >> $difftmp
        else
            diff -wi $fname1 $fname2 >> $difftmp
        fi
    else
        if [ "$_compress" = "1" ];  then
            diff -w <(sed 's/_//g; /Source.*v9.*Output file:/d' $fname1 )
<(sed 's/_//g; /Source.*v9.*Output file:/d' $fname2) >> $diffmp
        else
            diff -w $fname1 $fname2 >> $difftmp
        fi
    fi
}
```

FUNCTION TO FIND MISSING FILES

```
nexistf(){
    fname1=$1
    fname2=$2
    _misstfl=$(($_misstfl+1))
    if [ $_misstfl -eq 1 ] ; then
        printf "%-70s     %s\n" "           $_dir1" "              $_dir2" >>
$misstmp
    fi
    if [ ! -r "$fname1" ]; then
        printf "%-70s" "    $(basename $fname1)"  >> $misstmp
        if [ ! -r "$fname2" ]; then
            printf "%s\n" "     $(basename $fname2)"   >> $misstmp
        else
            printf "%s\n" "" >> $misstmp
        fi
    elif [ ! -r "$fname2" ]; then
        printf "%-70s %s\n" "" "    $(basename $fname2)"  >> $misstmp
    fi
}
```

FUNCTION TO COMPARE TIMESTAMPS

```
compt(){
    fname1=$1
    fname2=$2
    #get timestamp for the two files
    f1time=`stat --printf=%y   $fname1 | cut -d. -f1`
    f2time=`stat --printf=%y   $fname2 | cut -d. -f1`
    if [ "$_bdir1" = "validation" -a ! "$_bdir2" = "validation" ]; then
        if [ "$f1time" \< "$f2time" ] ; then
            _dirstvil=$(($_dirstvil+1))
            if [ $_dirstvil -eq 1 ]; then
                #printf "%s\n" "WARNING: following .out files in $_dir1
modified earlier than in $_dir2" >> $timetmp
                printf "%-70s     %s\n" "           $_dir1" "
$_dir2" >> $timetmp
            fi
            printf "    %-40s : %s         %-40s : %s\n" "$(basename
$fname1)" "$f1time" "$(basename $fname2)" "$f2time" >> $timetmp
            printf "%s\n" "" >> $timetmp
        fi
    fi
    if [ ! "$_bdir1" = "validation" -a "$_bdir2" = "validation" ]; then
        if [ "$f1time" \> "$f2time" ] ; then
            _dirstvil=$(($_dirstvil+1))
            if [ $_dirstvil -eq 1 ]; then
                printf "%-70s     %s\n" "           $_dir1" "
$_dir2" >> $timetmp
            fi
            printf "    %-40s : %s         %-40s : %s\n" "$(basename
$fname1)" "$f1time" "$(basename $fname2)" "$f2time" >> $timetmp
        fi
    fi
}
```