Handed out 02/15/2016

CS 3423.01

Assignment 4 : Perl Exploration

Due Midnight Monday, February 29, 2016

Write the following Perl scripts:

mysha256 <rootfile> [sha256file]

mysha256 will compute the SHA256 message digest (fingerprint) of every regular file under any of the directories in the file rootfile whose name is given on the command line and will write on each output line, the pathname of each file followed by the message digest. The pathname and message digest should be separated by a tab. Output should be to the sha256file file if the sha256file argument is given on the command line and to stdout otherwise.

In order to compute the SHA256 message digest of a file, use the FSF program, sha256sum. You may not use any switches. The **UNIX** command

```
sha256sum /usr/local/courses/cs3423/doc/tutorial.perl.ps
```

yields the output (all on a single line)

```
be87f0fdb7dec8b977913b4f9d70494991a7a7d99cdd11a9ebb7d1717b0cefef
                                /usr/local/courses/cs3423/doc/tutorial.perl.ps
```

which should be written out as (all on a single line)

```
/usr/local/courses/cs3423/doc/tutorial.perl.ps
                be87f0fdb7dec8b977913b4f9d70494991a7a7d99cdd11a9ebb7d1717b0cefef
```

You can use the files and directories under /usr/local/courses/cs3423 in case you don't have sufficient structure under your home directory.

Extra Credit - verifile rootfile sha256file

verifile should verify that all files under any of the directories in rootfile have the same sha256 digest as listed in sha256file and will print out those files that have changed, been added or been deleted in pairs separated by a blank line as follows (missing entries should be null - just <old> or <new> as appropriate) (again each entry is on a single line):

```
<old>/usr/local/courses/cs3423/doc/tutorial.perl.ps
                be87f0fdb7dec8b977913b4f9d70494991a7a7d99cdd11a9ebb7d1717b0cefef
<new>/usr/local/courses/cs3423/doc/tutorial.perl.ps
                4c17883f3994a95e0f2bcb7075f0fbce6fea970a6e620579260c99cd5995cf92


<old>
<new>/usr/local/courses/cs3423/doc/qperl.pdf
                0b1b94d97c83f21c60a0a28dfe8f87999e6b4970241dc337bee96960c8858efa
```

The real extra credit here is making each program an efficient process. If anything special is done it should be documented carefully in the body of the script.

plgrep

plgrep [-f] <perl regular expression> <file/directory list>

will grep for all occurences of the regular expression in all regular files in the file/directory list as well as all regular files under the directories in the file/directory list. If a file is not a TEXT file then the file should first be operated on by the unix command strings (no switches) and the resulting lines searched. If the -f switch is given only the file name of files containing the regular expression should be printed, one per line. A file name should occur a maximum of one time in this case. If the -f switch is not given then all matching lines should be printed, **each** preceeded on the same line by the file name and a colon. An example invocation:

```
plgrep 'b+a*d' file1 dir1 dir2 file3 file4 dir3
```

ddiff

`ddiff [-ds12] <directory1> <directory2>`

ddiff will compare two directories for differences in regular files. All regular files with the same names should be tested with the unix function `/usr/bin/diff -q` which will determine whether they are identical. A file in `directory1` which does not have a similarily named file in `directory2` will have its name printed after the string `<<<` while a file in `directory2` without a corresponding `directory1` entry will be prefixed with the string `>>>`. If two files have the same name but are different (as determined by `diff`) then the file name will be surrounded by `< >` while if the files are the same then the file name will be surrounded by `> <`. Note that all file names will have a blank separating the name from the symbols.

```
<<< file1
>>> file2
< file >
> file <
```

If the `-s` switch is specified those filenames occuring in both directories which are the same should be printed while the `-d` switch will cause the printing of the names of those different identically-named files. If the `-1` or `-2` switch is specified then those files only in `directory1` or `directory2` respectively will be printed. No switches is the same as all switches.

There are some tricky points in this program:

1. The path name is necessary in order to test the files for regularity and you are given the paths on the command line.
2. You do not want the path in the filename when you test for equality of the names and to sort (if you wish).
3. You will need to use the complete pathname in order to use the `diff` command, just remember that the directories are known!
4. It is permissible to require that the switches be all together, i.e.

   `ddiff -s1 /usr/local/jdk1.6.0_14/bin /usr/local/jdk1.7.0_25/bin`
5. You don't need to bother testing for equality of the switch, only test to see if it contains the appropriate character. There is a function, `getopt`, which is generally used to set switch booleans but we won't use it.
6. Be certain that you understand that what you are doing is classifying every regular file name in either directory into one of four `disjoint` categories.
7. You will need to make two directories which have all types of files in order to test this script.

Submission

1. Submit your assignment:

   `submit assign4 project4 3423`
2. It is your responsibility to be certain that you turned in the correct assignment and that the assignment was correctly turned in. Please just submit the three (or four) perl script files (no extensions). In order to verify what programs you actually turned in type:

   `verify -c cs3423 -p project4`
3. Remember, the assignment **must** be turned in by the due time. The assignment will automatically be turned off at that time. If your project is late, then turn it in as `project4-late`. This will only extend for an additional week and there will be a penalty associated with a late project.