CS1023 Software Development Fundamentals

Assignment: 7, Maximum Marks: 30

1. **[2 Marks]** Write a shell script named `datep` that produces the date in this more printable form:

```
> date
Thu Jan 24 09:10:28 CST 2008
> datep
Jan 24, 2008
```

2. **[2 Marks]** Write a shell script that renames files based on the file extension. It should ask the user what prefix to prepend to the file name(s). By default, the prefix should be the current date in YYYY-MM-DD format. If the user simply presses enter, the current date will be used. Otherwise, whatever the user entered will be used as the prefix. Next, it should display the original file name and new name of the file. Finally, it should rename the file.

3. **[3 Marks]** Write a shell script named `findwords` that finds words matching a regular-expression pattern given as its argument. `findwords` always searches a file, provided as a second argument (if not provided then it should look for a default file in the current directory). This tool should:
   - write the words it finds to standard output, in alphabetical order
   - write up to four words per line, lined up in columns
   - ignore case both in finding words and sorting them
   - redirect any error messages, such as file does not exist, to `/dev/null`

   For example:

```
> ./findwords 'twas' macbeth-list.txt
'twas              outwash            twas
miltwaste          outwaste           twasome

> ./findwords '^a.*rum.*$'
Agarum             anoestrum          Aplectrum          Asarum
alabastrum         antirumor          archicerebrum      Ascyrum
alarum             antiserum          arcocentrum        aurum
ambulacrum         antistrumatic      Arrhenatherum      autoserum
Anatherum          antistrumous       Arum
androphorum        antrum             arumin

> ./findwords '^a.*rum.*$' macbeth-list.txt
Agarum             Anatherum          antistrumous       Arum
alabastrum         androphorum        antrum             arumin
alarum             anoestrum          Aplectrum          Asarum
alarum'd           antirumor          archicerebrum      Ascyrum
alarums            antiserum          arcocentrum        aurum
ambulacrum         antistrumatic      Arrhenatherum      autoserum
```

Note: Feel free to use this file or if you prefer, you can use your own. Just make sure to include it in your submission.

4. **[2 Marks]** Using Bash functions, write a script `mkrandom.sh` that generates a user-specified number of files of user-specified size filled with random content.

5. **[5 Marks]** Write a shell script `phonebook` which has the following behavior:
   - `./phonebook new <name> <number>` adds an entry to the phonebook. Don't worry about duplicates (always add a new entry, even if the name is the same).
   - `./phonebook list` displays every entry in the phonebook (in no particular order). If the phonebook has no entries, display `phonebook is empty`
   - `./phonebook remove <name>` deletes all entries associated with that name. Do nothing if that name is not in the phonebook.
   - `./phonebook clear` deletes the entire phonebook.
   - `./phonebook lookup <name>` displays all phone number(s) associated with that name. You can assume all phone numbers are in the form `ddd-ddd-dddd` where `d` is a digit from 0-9.

NOTE: You can print the name as well as the number for each line. For an additional challenge, try printing all phone numbers *without* their names. (See the example below for more details)

For example,

```
$ ./phonebook new Linus Torvalds 101-110-0111
$ ./phonebook list
Linus Torvalds 101-110-1010
$ ./phonebook new Tux Penguin 555-666-7777
$ ./phonebook new Linus Torvalds 222-222-2222
$ ./phonebook list
Linus Torvalds 101-110-1010
Tux Penguin 555-666-7777
Linus Torvalds 222-222-2222
# OPTIONAL BEHAVIOR
$ ./phonebook lookup Linus Torvalds
101-110-1010
222-222-2222
# ALTERNATIVE BEHAVIOR
$ ./phonebook lookup Linus Torvalds
Linus Torvalds 101-110-1010
Linus Torvalds 222-222-2222
$ ./phonebook remove Linus Torvalds
$ ./phonebook list
Tux Penguin 555-666-7777
$ ./phonebook clear
$ ./phonebook list
phonebook is empty
```

If you run into an edge case that isn't described here, you can handle it however you wish (or don't handle it at all). You can assume all inputs are in the correct format.

6. **[5 Marks]** Write a shell script that takes a CSV file (a sample is given below, but consider making your own small CSV file) as a command-line argument and repeatedly prompts the user for operations on the CSV file until they choose to exit.
   - Supports operations like concatenation, and summation by reading the operation and the columns for the operation as input from the user.
   - Create a new column with an appropriate name based on the operation (eg. concat_c1_c2 if concatenating column 1, column 2)
   - Save the updated data to modified.csv
   - Open-ended: Try including any 2 other string or arithmetic operations of your choice

   Input.csv

   ```
   ID,FirstName,LastName,BaseSalary,Bonus
   1,Alice,Johnson,50000,5000
   2,Bob,Smith,60000,7000
   3,Charlie,Brown,55000,6000
   ```

   Sample Interaction:

   ```
   Choose an operation:
   1. Concatenate
   2. Sum
   3. Exit
   Enter your choice: 1

   Enter column numbers to concat (comma-separated): 2,3

   Choose an operation:
   1. Concatenate
   2. Sum
   3. Exit
   Enter your choice: 2

   Enter column numbers to sum (comma-separated): 4,5

   Choose an operation:
   ```

```
1. Concatenate
2. Sum
3. Exit
Enter your choice: 3

Final output saved to `output.csv`
```

Output.csv

```
ID,FirstName,LastName,BaseSalary,Bonus,Concat_c2_c3,Sum_c4_c5
1,Alice,Johnson,50000,5000,Alice Johnson,55000
2,Bob,Smith,60000,7000,Bob Smith,67000
3,Charlie,Brown,55000,6000,Charlie Brown,61000
```

7. **[3 Marks]** Take a CSV file (create one by yourself), and a column number as a command line argument. Check the first 3 values in the specified column for an Arithmetic Progression (AP) pattern in the numbers. If the numbers are in AP, automatically fill the further rows in the column by getting the required number of rows from the user. If the numbers are not in AP, exit with exit code -1.

   While automatically generating new rows, you may consider the values in other columns as "" (empty string).

8. **[5 Marks]** Create a script `spellcheck` that takes one or more arguments, `f1, f2, ..., fn` and should work as follows:
   - If the script is not supplied with an input argument, print a usage note to `stderr` and exit with a code of 1

     ```
     [localhost]$ spellcheck

     Usage: spellcheck filename
     ```

   - If an argument is supplied that is not a valid file (`[ ! -f $1 ]`), spellcheck prints an appropriate error message to `stderr` and skips that argument.
   - Your script should read each word in a supplied file and compare (you may use `grep` or any other utility) that word to the dictionary (which is assumed to exist at `/usr/share/dict/words`). If the word is *not* found in the dictionary it should be added to the end of a file called `<FILE>.spelling`. You will want to add words in the order you encounter them.
   - If `<FILE>.spelling` does not exist, print a message to `stdout` stating that the script is **creating** it.

- If `<FILE>.spelling` already exists, print a message to stdout stating that you are deleting the old file and **replacing** it.
- The search should be case-insensitive.
- At the end of processing each file, print a message to stdout asserting completion of the file, with the number of words found, and the number of unique words found.
- Your script should exit with a return code of 0 after completing the final print statement.

Note: You can use this file if you cannot locate a file called `words` in your system.

9. **[3 Marks]** Create a bash script `combine` that takes 2 or more arguments, call them `f1, f2, ..., fn`. Script `combine` should work as follows:
    - All arguments are treated as filenames.
    - If fewer than two arguments are given, print `Usage: combine outputfilename [inputfilename ...]` error message on `stderr` and exit with a return code of 1.
    - If a file or directory `f1` already exists, print `Error: Output file should not exist` on `stderr` and exit with a return code of 1.
    - Otherwise, concatenate the contents of `f2, ..., fn` and put them in `f1`. You will want to handle errors with input files (for example, if a file does not exist), but do not print these errors. Instead, these error messages should be redirected to `f1`. Exit with a return code of 0 after copying all the input files.

    A simple example of running combine is:

```
[localhost]$ echo "making file 1" > file1
[localhost]$ echo "and another one for file 2" > file2
[localhost]$ touch file3
[localhost]$ ./combine output file1 file2 nonfile file3
[localhost]$ cat output
making file 1
and another one for file 2
nonfile: No such file or directory
```

Submission Guidelines

Submit a single zipped file containing all your shell scripts and any additional files used. Name the file in the format: **Assign7_<roll no>.zip** (for example, **Assign7_cs24btechxxxxx.zip**).

Organize your scripts and related files into separate folders for each problem, naming the folders as **Problem_X**, where **X** represents the problem number (for example, **Problem_1/** for Problem 1). If the problem statement does not specify a script name, use the format **Problem_X.sh** for naming your script.