System Programming in C – Homework Exercise 3

Publication date: *Friday, April 13, 2018*
Due date: *Thursday, May 3, 2018 @ 21:00*

General notes:

- A piped sequence of commands is an ordered sequence of Linux commands, each connected to its preceding command using the pipe symbol ('|'). Do not use any other way to combine commands (e.g. *process substitution*).
- Restrict use to commands we saw in class:

  cp cat head tail cut paste wc sort uniq tr grep sed

- Make sure to use relative paths and not absolute paths. Your solution scripts should be executable from any location in the file system. In particular, your scripts should not contain names of specific directories, other than the data source directory `/share/ex_data/ex3/`.

Problem 1:

1. Copy story file for 'War and peace' (called `war-and-peace.txt`) from `/share/ex_data/ex3/` to the current directory.
2. Write a piped sequence of commands that takes the `war-and-peace.txt` file from your current directory and prints to file `war-unique-words.txt` all distinct words in `war-and-peace.txt` in alphabetical order.

   Notes:

   - A word is defined as a contiguous sequence of **letters** flanked on both sides by a white space or a punctuation mark. For example, the five words in the sentence "I am Peter-Pan the 5th…" are: "I", "am", "Peter", "Pan", and "the"; "5th" is not counted as a word because it contains a digit.
   - Please note that a word in this Problem is defined slightly differently than how it was defined in Problem 1 of Homework Exercise 2. For instance, the empty word is not counted as a word here.
   - Two words are distinct if they are identical disregarding case. For example: "Amy" and "aMy" are not distinct.
   - File `war-unique-words.txt` should contain exactly 17,543 words. You can compare it with the file `/share/ex_data/ex3/war-unique-words.txt` (using `diff`).
3. Write a piped sequence of commands that takes the file `/share/ex_data/ex3/war-unique-words.txt` and prints to file `war-4consonants.txt` all words that have a consecutive sequence of at

least 4 consonants. A consonant (עיצור) is any letter other than a,e,i,o,u,y. (yes, we consider 'y' to be a vowel here). The words should be printed in alphabetical order. File `war-4consonants.txt` should contain exactly 259 words. You can compare with file `/share/ex_data/ex3/war-4consonants.txt` (using `diff`).

4. Write a piped sequence of commands that finds the most frequent word in `war-unique-words.txt` among all words that have at least 4 occurrences of 'a' or 'e'. The piped sequence should print to the standard output the following line: `The word repeated appears 145 times in file.`
   This is because the word `repeated` is the most common word that has 4 occurrences of 'a' or 'e'; the word appears precisely `145` times in the file.

   Hint: use the `sed` command for the final printing.

5. Write the commands you used in 2-4 in a single file named `problem1sol.sh`. The file should contain exactly 3 lines, each containing a single command or a piped sequence of commands. (You can use `wc -l` to make sure your file contains only 4 lines of text.)

Problem 2:

In this problem, we assume that there is a bash variable named `dir`, which contains the path to some directory in the file system. See comment at the end of the problem related to testing your solution.

1. Write a piped sequence of commands that lists all **empty files** in directory `$dir`. A file is empty if it takes 0 bytes of memory. The file names should be printed, one per line, in a file named `empty_file.out`.

2. Write a piped sequence of commands that lists all files in directory `$dir` whose name contains either "file" or "File" followed by at least one character. For each listed file, your piped sequence of commands should print the file's name and the time it was last updated.

   Examples of file names that should be listed: file1, myfile.txt, myFile.c, file_1
   Examples of file names that should **not** be listed: file, myfile, FILE1

   The output should be printed into the file `updates.out` in the following format:

   `File <file_name> last updated on <time_of_last_update>`

   Note 1: Make sure to print in this exact format. There are no double spaces in the output (a space character is never followed by another space).

   Note 2: there are two ways to filter for filenames that satisfy the requirements: (1) using file name expansions, (2) using grep. Try solving this problem using each of these approaches (but submit only one).

3. Write a piped sequence of commands that prints to the file `proper_dates.out`. a full listing (`ls -l`) of directory `$dir` where the dates of last update for all files are given in "proper" form. This is demonstrated below.

   This is what we originally get with `ls -l`:

```
==> ls -l
-rw-r--r-- 1 gronau staff      0 Apr 21  2017 file_1
-rw-r--r-- 1 gronau staff     12 Apr  2 12:34 FILE1
-rw-r--r-- 1 gronau staff    272 Apr  3 22:16 myFile
-rw-r--r-- 1 gronau staff  37030 Apr 11 14:16 myfile1
```

   Your piped sequence of commands should output: (the yellow highlight is just to emphasize the added text).

```
-rw-r--r-- 1 gronau staff      0 Apr 21st  2017 file_1
-rw-r--r-- 1 gronau staff     12 Apr  2nd 12:34 FILE1
-rw-r--r-- 1 gronau staff    272 Apr  3rd 22:16 myFile
-rw-r--r-- 1 gronau staff  37030 Apr 11th 14:16 myfile1
```

   Note:

   - For the last modified date, the command `ls -l` provides the year if it is not the current year.
   - The suffix "st" is added to following numbers: 1, 21, 31, 41, …
   - The suffix "nd" is added to following numbers: 2, 22, 32, 42, …
   - The suffix "rd" is added to following numbers: 3, 23, 33, 43, …
   - The suffix "th" is added to all other numbers (e.g. 8, 11, 12, 13, 25).
   - You may assume that all files were last updated in the month of April (though not necessarily this year). You may also assume that there is at least one file updated after April 9 (but still in the month of April). (These assumptions potentially simplify the commands you need to use).

   Hint: this can be solved with five piped `sed` commands (maybe even less?).

4. Write the commands you used in 1-3 in a single file named `problem2sol.sh`. The file should contain exactly 3 lines, each containing a single command or a piped sequence of commands. (You can use `wc -l` to make sure your file contains only 3 lines of text.)

To test your solution, you can do the following:

```
==> dir=/share/ex_data/ex3/sampledir-problem2
==> source problem2sol.sh
```

Then use `diff` to compare your three output files with the three files with the same name that are in `/share/ex_data/ex3/`. The files should be identical.

Problem 3:

This problem involves writing a series of script files. For items (2) and (3) you may wish to wait for after Lecture 4. The next page contains detailed execution examples, which clarify the requirements of each task and should also be used for testing.

Important note: Make sure that your scripts produce output in the exact format specified in the examples on the next page. Any difference, such as extra spaces, misspelling, or wrong case, will cause our tests to fail.

1. Write a piped sequence of commands that takes the variable `dir`, which contains a path for a directory. It then prints the names of all subdirectories directly under that directory with read access for all users. The piped sequence should start with `ls -l $dir |`.

   The piped sequence of commands should be written in file `list_dirs.sh`.

2. Write a bash script named `isdir.sh` that receives as input a path specifying either a file or a directory, and prints a number according to the following specification:

   - 1 – path is a directory
   - 0 – path is a file (not directory)
   - -1 – path does not exist or does not have read access

3. Use the code you wrote in (1-2) above to write a bash script named `dir_depth.sh` that receives as an argument a single path and prints the depth of the directory tree rooted at this path. The depth is defined as follows:

   - If the given path is a valid file, but not a directory, the depth is 0
   - If the path is a valid directory with no subdirectories, the depth is 1.
   - If the path is a directory with subdirectories, then its depth is defined by the depth of its **deepest subdirectory with read access for all users (+1)**.
   - If the path does not exist or does not have read access, then the script should print `"no such path"`, and exit with code 1.
   - If no arguments are given, then print a usage notice ("`Usage: ./dir_depth.sh <path>`") and exit with code 1. (Ignore arguments after first one, if given)
   - Your script should utilize the scripts `isdir.sh` and `list_dirs.sh` and should **call itself recursively**.
   - You may assume that you are executing the script from a directory that contains all the scripts you are calling.

Execution examples for Problem 3:

Problem 3.1

```
==> dir=/share/ex_data/ex3/; source list_dirs.sh
sampledir-problem3

==> dir=/share/ex_data/ex3/sampledir-problem3; source list_dirs.sh
data
files
```

Problem 3.2

```
==> ./isdir.sh /share/ex_data/ex3/sampledir-problem3
1

==> ./isdir.sh /share/ex_data/ex3/sampledir-problem3/README
0

==> ./isdir.sh /share/ex_data/ex3/sampledir-problem3/README.bak
-1

==> ./isdir.sh /share/ex_data/ex3/sampledir-problem3/classified
-1
```

Problem 3.3

```
==> ./dir_depth.sh
Usage: ./dir_depth.sh <path>

==> ./dir_depth.sh /share/ex_data/ex3/sampledir-problem3 other args are
ignored
3

==> ./dir_depth.sh /share/ex_data/ex3/sampledir-problem3/README.bak
no such path /share/ex_data/ex2/sampledir-problem3/README.bak

==> ./dir_depth.sh /share/ex_data/ex3/sampledir-problem3/README
0
```

Submission Instructions:

In the directory `~/exercises/ex2/submit/` you should have **exactly 5** items:

- `problem1sol.sh` – file with your solution to Problem 1
- `problem2sol.sh` – file with your solution to Problem 2
- `list_dirs.sh` – file with your solution to Problem 3.1
- `isdir.sh` – file with your solution to Problem 3.2
- `dir_depth.sh` – file with your solution to Problem 3.3
- `PARTNERS` – partner file

Important notes before submission:

- Use the exact file and directory names specified in the assignment. Linux is case sensitive, and file extensions are part of a file's name.
- Test your solutions by executing them, for example:
  `==> source problem1sol.sh`
  Test your code using the provided example files for Problems 1,2 and examples listed in this document for Problem 3.
- Before you finalize your submission, please run `test_ex 3` to validate your PARTNERS file, make sure you have all required files and no unnecessary files. Points will be deducted for incorrect submission format!

All files should be placed on the server (in `~/exercises/ex3/submit/`) at the time of deadline. Any changes made after that point will count as a late submission. For more information, see the **Homework submission instructions** file on the course website.