System Programming in C – Homework Exercise 3

Publication date: *Wednesday, April 17, 2024*
Due date: *Sunday, May 12, 2024* @ 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*).
- You should only **use commands we saw in class**. For command **`grep`**, you may use the following options (if needed): **`-E -v -c`**. For command **`sed`**, use the **`-r`** option. **Do not use any other option for these two commands!**
- As always, make sure to use relative paths and not absolute paths. Your solution scripts should be executable from any location in the file system. The only exception to this rule is to use `/share/ex_data/ex3/` which is an absolute path.

Problem 1:

This problem involves additional parsing tasks for files `planets.txt` and `story.txt`, which you parsed in the previous HW assignment. Recall that `planets.txt` holds information about the nine planets in our solar system. Every line in file `planets.txt` has the following format:

<planet><comma><space>**<year>[**<space>**<status>]**

- **<planet>** consists of an uppercase letter followed by any number of lowercase letters (e.g., `Mars`).
- **<year>** is either a four-digit number representing the year of discovery or -1 to indicate year of discovery unknown (or not applicable).
- **<status>** is an optional field and is not specified for every planet. When it is specified, it contains text describing the planet. This text may contain white spaces, but you may assume it does not contain any punctuation marks.
- <space> and <comma> represent the characters `' '` and `','`.

Further, recall that `story.txt` contains the text of the novel *Animal Farm* by George Orwell and follows no particular format.

Using these files, write commands for each of the tasks specified in 1-6 below. Specify each piped sequence in a separate line in a solution file named `ex3.1.sh`. The file should contain exactly six lines, each containing a single command or a single piped sequence of commands.

1. Copy the files `planets.txt` and `story.txt` from directory `/share/ex_data/ex3/` to the current directory.

2. Write a single command that takes the file `planets.txt` and prints each line reformatted as follows into file `planet-discovery.txt`:

   `In year <year>, planet <planet> was discovered`

   Fields `<year>` and `<planet>` should be printed as they are in `planets.txt`, and the optional status should be omitted.

   Validation: File `planet-discovery.txt` should contain the same number of lines as `planets.txt`, and its first two lines should be:
   ```
   In year -1, planet Earth was discovered
   In year 1610, planet Mars was discovered
   ```

3. Write a single command that takes the file `planets.txt` and prints to file `planet-status.txt` all lines in `planets.txt` that correspond to planets whose name contains exactly five letters and whose status is non-empty.

   Validation: the file `planet-status.txt` should contain the lines corresponding to planets Earth and Pluto, exactly as they are in `planets.txt`.

4. Write a piped sequence of commands that takes the file `planets.txt` and prints to file `planets-before-1900.txt` the names of all planets which were discovered between the years 1000 and 1899. The output file should only contain planet names, and the names should be sorted alphabetically. Planets with unknown discovery dates (-1) should not be listed. Note that the optional status may contain numbers, and in particular years.

   Validation: the file `planets-before-1900.txt` should contain exactly six lines with the following planet names (in that order, one planet per line): Jupiter, Mars, Neptune, Saturn, Uranus, and Venus.

5. Write a piped sequence of commands that assigns a `bash` variable named `num_lines` with the number of lines of `story.txt` that satisfy the following criteria:

   - The line contains a word with a vowel (`a`, `e`, `i`, `o`, or `u`) followed by either `cc` or by `ss` (all in lower-case) and right after the word there is a punctuation character.
   - The line contains at most 200 letters (a-z or A-Z).

   Validation: the value of variable `num_lines` should be 9.

6. Snowball is one of the main characters in *Animal farm*. As a result, the story mentions the word “Snowball's” many times with different words following it. For example, “Snowball's plans”, and “Snowball's tail”. Write a piped sequence of commands that prints to file `snowballs_words.txt` all distinct words mentioned after the word “Snowball's”.
   - A word is said to follow “Snowball's” if it is a string of lower-case letters that follows the word "Snowball's" and a non-empty sequence of white spaces. Note that one of these white spaces can be a newline. This word should be followed by a white space or a punctuation mark.
   - A line in `story.txt` may contain several instances of the word "Snowball's".
   - The words in `snowballs_words.txt` should be sorted alphabetically and without repetitions.

   Note: If you use `sed` to search for the word “Snowball's”, then you should wrap the `sed` command with double quotes (`" "`) and not with single quotes (`' '`), as we typically do, so that the quote character in “Snowball's” is not mistaken to be the closing quote for the command.

Validation: the file `snowballs_words.txt` should contain 19 lines. You can find a copy of the expected file in `/share/ex_data/ex3/snowballs_words_expected.txt`. Compare your output with this file for validation.

Final validation and testing for Problem 1:

1. Verify that file `ex3.1.sh` has exactly six lines (use `wc`) and follow the specific validation steps specified for each item in 2-6.
2. Execute the commands in file `ex3.1.sh` as scripts (using `source`) and confirm that the following six files are created in the current directory (use `ls`):

   ```
   planet-discovery.txt  planets-before-1900.txt  planet-status.txt
   planets.txt           snowballs_words.txt      story.txt
   ```

3. The script should **not print anything to the standard output** and no error messages should be printed.
4. Check the value of bash variable `num_lines` for Problem 1.5.
5. Run the script from **various locations** (not just `~/exercises/ex3/`) to ensure that it does not contain unwanted absolute paths.

6. Use the script `/share/ex_data/ex3/test_ex3.1` to test your solution. Execute the following command from directory `~/exercises/ex3/`:
   ```
   ==> /share/ex_data/ex3/test_ex3.1
   ```
   The script produces a detailed error report to help you debug your code. It tests your code on the original input files as well as slightly modified versions to make sure that your code follows various requirements. If your solution fails on the modified input file, try to see where you made assumptions based on the specific input files. If you cannot find the cause, just submit as is. The graders will let you know what the problem was.

Problem 2:

In this problem, we ask you to write two bash scripts: in Problem 2.1, you will write one for computing the storage size of a single item (file or directory) and in Problem 2.2 you will write one for computing the storage size of a given directory subtree. The second script uses the first one.

1. Write a bash script called `get-storage.sh` which accesses a bash variable named `my_item` that holds a path to a file or directory and prints the number of bytes that are used by this item for storage (as specified by `ls -l`). Follow these guidelines:
   - The script `get-storage.sh` should consist of exactly one line containing a piped sequence of commands that starts with `ls -ld $my_item`
   - Assume that the bash environment in which the script `get-storage.sh` is executed contains a variable named `my_item`, which holds a valid path to an item (file or directory) in the file system.

Execution examples:

```
==> my_item=/share/ex_data/ex3/myDir/myFile
==> source get-storage.sh
113

==> my_item=/share/ex_data/ex3/myDir/1984.txt
==> source get-storage.sh
118902

==> my_item=/share/ex_data/ex3/myDir
==> source get-storage.sh
4096
```

2. Write a bash script named `subtree-storage.sh` that receives a path as an argument in the command line. The script should print the total number of bytes that are used for storage by the (directory) tree rooted in this path. Follow these guidelines (and see execution examples on the next page):

   - If the script does not receive exactly one input argument (either no arguments or more than one), then it should print the following error message and exit with status 1:

     `Usage: <SCRIPT_PATH> PATH`

     Where `<SCRIPT_PATH>` is the name (and path) of the script, as called by the user.

   - If the script receives exactly one input argument, but this argument is not a valid path (to a file or directory), then it should print the following error message and exit with status 2:

     `<ARG> is not a valid path`

     Where `<ARG>` is the argument given to `subtree-storage.sh`.

   - If the input argument of the script is a valid file path, the script should print the size of that file in bytes (as script `get-storage.sh`) and exit with status 0.

   - If the input argument of the script is a valid directory path (not file), then the script should print its size plus the sizes of all items (files and directories) contained within that directory's subtree in the filesystem and exit with status 0.

   Implementation notes:

   - The script should be implemented as a stand-alone executable script. Make sure to have an accurate "`#!`" comment at the top of the file and grant the script executable permissions to enable this.
   - Use the appropriate `if` statements to check that the first input argument contains a valid file path or a valid directory path.
   - Use the appropriate call to script `get-storage.sh` from Problem 2.1 to get the size of items. You may assume that this script exists in the current directory.
   - Use an appropriate recursive call to script `subtree-storage.sh`. The halting condition of the recursion should be when the input argument corresponds to a file path.
   - Use simple arithmetic operations for bash variables to sum sizes.

Execution examples:

```
==> ./subtree-storage.sh /share/ex_data/ex3/myDir/myFile
113
==> echo $?          (← recall that this prints the exit status)
0

==> ./subtree-storage.sh /share/ex_data/ex3/myDir
929566
==> echo $?
0

==> ./subtree-storage.sh \
      /share/ex_data/ex3/myDir/dir1/dir5/a-tale-of-two-cities.txt
781102

==> ./subtree-storage.sh /share/ex_data/ex3/myDir/dir1/dir5
785485

==> ./subtree-storage.sh /share/ex_data/ex3/myDir/dir1
789581

==> ./subtree-storage.sh /share/ex_data/ex3/myDir/dir2/dir4
4096

==> ./subtree-storage.sh /share/ex_data/ex3/myDir/dir2
16874


==> ./subtree-storage.sh /share/ex_data/ex3/someDir
/share/ex_data/ex3/someDir is not a valid path
==> echo $?
2

==> ./subtree-storage.sh /share/ex_data/ex3/someDir SOMETHING
Usage: ./subtree-storage.sh PATH
==> echo $?
1

==> ~/exercises/ex3/subtree-storage.sh \
      /share/ex_data/ex3/myDir /share/ex_data/ex3
Usage: ~/exercises/ex3/subtree-storage.sh PATH
==> echo $?
1
```

Validation and testing for Problem 2:

1. Make sure to follow the specific guidelines for each question.
2. Execute your script on the execution examples provided for each question.
3. You should try additional inputs to make sure your scripts provide the required output.
4. Use the testing script `/share/ex_data/ex3/test_ex3.2` to test your solution by executing the following command from directory `~/exercises/ex3/`:

   ```
   ==> /share/ex_data/ex3/test_ex3.2
   ```

   The script produces a detailed error report to help you debug your code. It tests your scripts on various executions, including the execution examples specified above.

Submission Instructions:

1. After you validated and tested your solution, make sure that your `~/exercises/ex3/` directory contains the following scripts:
   - `ex3.1.sh`
   - `get-storage.sh`
   - `subtree-storage.sh`
2. Your `~/exercises/ex3/` directory should also contain a `PARTNER` file with the user id of the non-submitting partner. The non-submitting partner should also add a `PARTNER` file containing the user id of the submitting partner.
3. Check your solution by running **`check_ex ex3`**. The script should be executed from the account of the submitting partner and may be run from any directory. Clean execution of this script guarantees you 80% of the assignment's grade.
4. Once you are satisfied with your solution, you may submit it by running **`submit_ex ex3`**. The script should be executed from the account of the submitting partner, and may be run from any directory. You may modify your submission any time before the deadline (**12/5 @ 21:00**) by running **`submit_ex ex3 -o`** from any location.
5. For more information on the submission process, see the **Homework submission instructions** file on the course website.