CS69201: Computing Lab-1
Shell scripting and Shell commands

Deadline: October 22, 2024, 5 PM

Full Marks: 100

Important Instructions

1. Using Linux

1. This assignment is based on OS programming and will be beneficial for the students to do this assignment in a Linux OS.
2. If you don't have Linux, use Virtualbox + Linux image, both free. We suggest Ubuntu but feel free to use any Linux version. If you decide to use Cygwin or WSL, it's up to you; just remember that most of your batchmates (and the instructors) will use Linux (and Ubuntu), so, naturally, we might not be able to provide support for other systems.
3. In case of any difficulties in understanding any functionalities, you may use the same functionality in a Linux Terminal to get a better understanding of how it works.

2. Programming Language

Write programs in shell script under the Linux environment that would run in bash terminal.

3. Error Handling & Input-Output

1. Proper error handling (e.g., invalid no of arguments) is expected in this assignment.
2. The inputs should be taken from the Terminal and results displayed in the Terminal, if explicitly not mentioned.

4. Deliverables

Submit a zip file named `<your-roll-no_Assignment12.zip>` which should include:

1. All .sh files [10 files for the 10 parts] (names for each bash program is mentioned along with the question)
2. An additional readme file (call it `README`) containing details about how to compile the files and any additional information required for the instructor during checking that you feel is necessary.

Objective

In this assignment we will start with a simple exercise – familiarization with shell scripting. Shell or "bash" (a variant of shell software) is your interface to access a multitude of software (often called "commands"), some of which are directly provided by the OS. Shell OR command line OR terminal can be simply accessed in your Linux distribution by opening the terminal app.

Shell commands to be familiar with

There are a few shell commands that you should be familiar with to efficiently use shell (aside from the shell scripting syntax). Search google (also write `man commandname` in terminal) to know more about them (some or all of them might also be useful for you to solve this assignment). Each command can take multiple arguments to perform various tasks.

- cat
- sort
- awk
- head
- tail
- cd
- mkdir
- ls
- touch
- file
- tr
- grep
- alias
- dd
- nohup
- wc
- split
- cut
- sed
- curl

Questions

Write programs in shell script under the Linux environment that would run in bash terminal and perform the following Tasks.

Question 1: Basic calculator [calculator.sh] [5 marks]

Design a shell script for implementation of a basic calculator that takes 3 arguments: `number1`, `operator`, and `number2`, and performs basic operations (+, -, *, /). The result is displayed up to 2 decimal places.

Implementation Requirements

1. Must accept exactly 3 command-line arguments: First and third argument will be a number (can be integer or decimal). Second argument: Mathematical operator (+, -, *, /).
2. Output must be: Displayed with exactly 2 decimal places Clearly formatted on the terminal Rounded appropriately
3. Proper error handling needs to be implemented, such as handling division by zero.

Example Usage

```
./calculator.sh 10 + 5
Output: 15.00
./calculator.sh 20 / 3
Output: 6.67
./calculator.sh 7 * 8
Output: 56.00
```

Question 2: Finding out lcm of numbers [lcm.sh] [5 marks]

Design a shell script that takes numbers as input from a text file `numbers_2.txt` and calculates their Least Common Multiple (LCM). The script reads these numbers from the file, calculates the LCM and outputs the result. .

Implementation Requirements

1. Must accept a single command-line argument.
2. LCM of the numbers should be displayed clearly on the terminal
3. numbers_2.txt

Example Usage

```
./lcm.sh numbers_2.txt
Output: 40
```

Question 3: Random Password Generator [password.sh] [5 marks]

Create a shell script that generates a secure random password of length n adhering to specific security criteria. The script should ensure the password contains a diverse set of characters and meets modern security standards as described below

The generated password must include:

- At least one uppercase letter (A-Z)
- At least one lowercase letter (a-z)
- At least one digit (0-9)
- At least one special character from the set: `!@#$%^&*()_+`
- Minimum total length of 4 characters to accommodate all required character types
- Characters must be arranged in random order in the final password

Implementation Requirements

1. Must accept a single command-line argument : n (the length of the password)
2. Generated password should be displayed clearly on the terminal

Example Usage

```
./password.sh 5
Generated password: Kj#97
```

Question 4: Text File Analysis Script [wordcount.sh] [5 marks]

Create a shell script that performs comprehensive text analysis on an input file. The script should process the file and perform the following :

- Calculate the total number of lines in the input file
- Count the total number of words present
- Determine the total character count
- Find the frequency of a specific word (provided as an argument)

Implementation Requirements

1. Must accept two command-line arguments:
   - First argument: Input text file path
   - Second argument: Search word for frequency analysis [case insensitive]
2. All counts should be displayed clearly on the terminal
3. Proper error handling for missing or invalid arguments
4. file_4.txt

Example Usage

```
./wordcount.sh file_4.txt fifa
Total Lines: 3
Total Words: 77
Total Characters: 430
Frequency of word 'fifa': 3
```

Question 5: API Fetch Script [api-fetch.sh] [15 marks]

You are provided with a list of GitHub API endpoints in `api-links_5.txt`, each returning detailed information about issues in a repository. Your task is to write a shell script that that fetches data from the endpoints, store them in a JSON file, and allows searching through the titles of the issues.

Implementation Requirements

1. Fetches the data from each URL.
2. Extracts specific fields from each response, such as:
   - `id`: The issue ID.
   - `number`: The issue number.
   - `title`: The issue title.
   - `user`: The username of the person who created the issue.
   - `labels`: A list of labels assigned to the issue.
3. Combines the extracted data from all URLs into a single JSON file, where each issue is a JSON object in an array.
4. Saves the combined data in a JSON file named `combined_issues.json`.
5. Implements a search feature that allows the user to search for a label (can be single or multiple words) in the combined JSON file. The script should display all matching results (only `id` and `title`).
6. Must accept a single command line argument : the word(s) to be searched for. [case insensitive searching]
7. api-links_5.txt

combined_issues.json

```
[
  {
    "id": 2448206827,
    "number": 2817,
    "title": "Incorrect␣fallback␣behavior␣in␣`@shopify/react-i18n`␣causing␣timezone␣to␣
       reset␣to␣default",
    "user": "asuponev",
    "labels": ["Type:␣Bug"]
  },
  {
    "id": 2448206817,
    "number": 2816,
    "title": "Another␣issue␣title",
    "user": "anotherUser",
    "labels": ["Type:␣Enhancement"]
  }
]
```

Example Usage

```
./api-fetch.sh Enhancement
Matching Results:
- ID: 2448206817 Title: Another issue title
```

Question 6: To-Do List Manager Script [todo.sh] [15 marks]

Design and implement a shell script for managing a simple to-do list. The script should support the following functionalities: `add` , `show` and `complete` a task.

Task Operations:

- **Add:** Add a task with the specified priority and mark it as incomplete by default.
- **Show:** Display tasks sorted by priority (high → medium → low). Incomplete tasks should be displayed first, followed by completed tasks.
- **Complete:** Mark a specific task (by task ID) as completed.
- You may store the tasks in a file (`todo_list.txt`), so the tasks are available even when the script is restarted.

Implementation Requirements

1. Must accept 3 command-line arguments for `add`:
   - First argument: Operation type (add)
   - Second argument: Task description (for add operation)
   - Third argument: Priority (for add operation: high, medium, low)
2. Must accept 2 command-line arguments for `complete`:
   - First argument: Operation type (complete)
   - Second argument: Task Id (for add operation)
3. Must accept only 1 command-line arguments for `show`:
4. Must implement proper error handling for Incorrect number of arguments, Invalid operation type, Task ID not found for completion and Invalid priority level .
5. The tasks should be stored in a text file (`todo_list.txt`) to ensure persistence between runs of the script.
6. For the `add` operation, the new task should be marked as incomplete by default (can be altered to ”complete”) and stored with the format:

   ```
   <task_id> | <task_description> | <priority> | incomplete
   ```

7. For the `show` operation, the tasks should be displayed sorted by priority (high, medium, low) with incomplete tasks listed first, followed by completed tasks.
8. For the `complete` operation, mark the specified task (by task ID) as complete and update the status in the text file.
9. You can treat task ID as a serial number in which tasks are being added to the to do list.

Example Usage:

```
./todo.sh add "Walk␣the␣dog" medium
./todo.sh add "Call␣mom" low
./todo.sh add "Finish␣homework" high
./todo.sh add "Submit␣project␣report" high
./todo.sh add "Buy␣groceries" high

./todo.sh complete 5
./todo.sh show
3 | Finish homework | high | incomplete
4 | Submit project report | high | incomplete
5 | Buy groceries | high | complete
1 | Walk the dog | medium | incomplete
2 | Call mom | low | incomplete

./todo.sh complete 3
4 | Submit project report | high | incomplete
3 | Finish homework | high | complete
5 | Buy groceries | high | complete
1 | Walk the dog | medium | incomplete
2 | Call mom | low | incomplete
```

Question 7 Search Books Script [search_books.sh] [15 marks]

Write a shell script to search for a phrase in book summaries and display matching books with context.

Input:

- A text file `books_7.txt` has lines in the format:

```
    <Book_Title> | <Author> | <Year> | <Summary>
```

Output:

For each match:

1. Print the book title and author.
2. Show the context around the search phrase from the summary based on the context type:
   - `-before`: Show exactly `n` words before the phrase.
   - `-after`: Show exactly `n` words after the phrase.
   - `-both`: Show exactly `n` words before and after the phrase.

Implementation Requirements

1. Must accept exactly 4 command-line arguments:
   - First argument: Input file (text file)
   - Second argument: Search phrase (case-insensitive, may contain multiple words)
   - Third argument: Context type (-before, -after, or -both)
   - Fourth argument: Number of context words (n)
2. Must implement proper error handling for: Incorrect number of arguments, File not found
3. The search for the input phrase should be case-insensitive.
4. For lines containing the search phrase: Show the context around the phrase from the summary based on the context type [exactly n words on either side]. If n words are not present on any side, print whatever words present.
5. For lines not containing the search phrase: Keep the original line unchanged.
6. books_7.txt

Example Usage

```
./search_books.sh books_7.txt "dark␣night" -both 3
Book: Dracula | Author: Bram Stoker
Context: "...tall␣against␣the␣dark␣night,␣as␣the␣wind..."

Book: Wuthering Heights | Author: Emily Bronte
Context: "The␣dark␣night␣brought␣with␣it..."
```

Question 8: Log File Analyzer [analyselog.sh] [15 marks]

Design a script that takes a log file and time window (in minutes) as input. You have to find the time window with the maximum number of log entries and display:

1. Start time of the busiest window
2. End time of the busiest window
3. Number of entries in that window
4. Top 3 most common IP addresses in that window (along with their occurrences)

Implementation Requirements

1. Must accept exactly 2 command-line arguments:
   - First argument: Input log file path
   - Second argument: Time window in minutes
2. Must implement proper error handling for: Incorrect number of arguments, file not found and others as necessary.
3. Each line of the log file starts with a timestamp in the format `[YYYY-MM-DD HH:MM:SS] <ip address> <request type>`.
4. The script should be able to parse timestamps and perform a sliding window calculation to determine the busiest window.
5. For IP counting, extract the IP address from each line of the log and calculate the frequency of the top 3 most common IP addresses within the busiest window.
6. access_8.log

Example Usage

```
./analyselog.sh access_8.log 15
Start Time: 2024-01-01 10:00:01
End Time: 2024-01-01 10:15:01
Number of Entries: 5
Top 3 IPs in the  window:
    192.168.1.2 - 2
    192.168.1.1 - 2
    192.168.1.3 - 1
```

Question 9: Alternating Case Text Processor [alternatecase.sh] [10 marks]

You are given a text file containing numerous lines. Each line includes words, numbers, and special characters and ends with a newline character. You are also given an input word. The script should process each line of the text file such that if a line contains the input word, then that line should be output in alternating case; otherwise, the line should remain unchanged.

Implementation Requirements

1. Must accept exactly 2 command-line arguments:First argument: Input text file path and the second argument: Search word
2. Must implement proper error handling for: Incorrect number of arguments, file not found and others.
3. The search for the input word should be case-insensitive
4. For lines containing the search word: Convert to alternating case (uppercase/lowercase). Preserve numbers and special characters and maintain original spacing
5. For lines not containing the search word: Keep the original line unchanged
6. References for this question : Alternating case
7. file_9.txt

Example Usage

```
./alternatecase.sh file_9.txt test
ThIs Is A tEsT fIlE
Welcome to IIT Kharagpur
ThIs Is A tEsT lInE
```

Question 10: Lexiographic Text Processor [lexiographic.sh] [10 marks]

You are given an input directory path (as a command-line argument) which contains several `.txt` files, where every file contains multiple names (one per line). You are also given an output directory path (it may or may not exist — if it doesn't, create it). You have to gather all names (across all files) beginning with the same letter in a single file. Thus, there will be 26 files (created by you inside the output directory) — one for each alphabet. Files can be empty if there is no name beginning with the respective character. You also have to ensure that the names inside each of these 26 output files are sorted in lexicographic order (ascending).

Implementation Requirements

1. Must accept exactly 2 command-line arguments:
   - First argument: Input directory path
   - Second argument: Output directory path
2. Must implement proper error handling for Incorrect number of arguments and Input directory not found and others as necessary
3. Must create output directory if it doesn't exist
4. Must create 26 separate files (A.txt to Z.txt) in the output directory
5. Must ensure names are sorted lexicographically within each output file
6. Must handle empty input files and invalid input gracefully
7. References for this question : Lexicographic ordering
8. input_dir_10

Example Usage

```
./lexiographic.sh <input_dir_10> <output_dir>

cat output_dir/A.txt
Adam
Alice
Amy

cat output_dir/B.txt
Bob
brief

cat output_dir/C.txt
Charlie

cat output_dir/D.txt
David

cat output_dir/E.txt

cat output_dir/F.txt
```