CS6.201 Introduction to Software Systems

Lab Activity 2

January 22, 2025

Bash and Shell

Instructions

- Only use shell commands to solve the questions.

Tasks

Part 1: `grep` Tasks

1. **Exact Word Search:** Find all lines in `sample.txt` that contain the exact word `pattern`.
2. **Limit Matching Lines:** Display the first **3 lines** in `sample.txt` that contain the word `log`.
3. **Pattern File Matching:** Search for lines in `sample.txt` that match any of the patterns listed in a file named `patterns.txt`.
4. **Count Non-Matching Lines:** Count the number of lines in `sample.txt` that do **not** contain the word `debug`.
5. **Search for Dates:** Find all lines in `sample.txt` that contain dates in the format `YYYY-MM-DD`.

Part 2: `awk` Tasks

1. **Filter Lines Based on Condition:** Print all lines from `data.csv` where the second field is greater than `50`.
2. **Add to Field Values:** Add `10` to the first field of all lines in `data.csv` and print the modified lines.
3. **Find Duplicate Lines:** Identify and print all duplicate lines in `data.csv`.
4. **Maximum of Average Values:** Compute the average of the first three columns in the `data.csv` file individually, and then determine and print the maximum of these averages

Part 3: `sed` Tasks

1. **Delete Specific Lines:** Remove lines 3 to 5 from `sample.txt` and display the result.
2. **Replace Word in File:** Replace all occurrences of the word `important` with `lite` in `sample.txt` and display the modified content.
3. **Transform Characters:** Convert all lowercase letters to uppercase in `sample.txt`.
4. **Multiple Substitutions:** Replace all occurrences of `code` with `coding` and `summary` with `iss` in `sample.txt`.
5. **print Specific Lines:** print lines 10 through 20 from `sample.txt`
6. **Reverse Line Order:** Reverse the order of all lines in `sample.txt`.

CTF Based Questions

Q1 - Background Story

Neel and Kevin were renowned adventurers in the land of **Scriptoria**, a mystical realm filled with ancient libraries, forgotten manuscripts, and puzzling ciphers. One fateful day, deep within the cavernous chambers of the Grand Archive, they stumbled upon a peculiar text file engraved on a glowing tablet. It was titled: "The Cipher of Aeons."

"Neel, this looks promising!" Kevin exclaimed, his voice echoing in the still air. "It says here that only those of keen intellect can break the Cipher of Aeons. If we solve it, we may unlock the location of the lost treasure of Scriptoria!"

Neel adjusted his glasses, already intrigued. "Hmm, let's see what we've got. The instructions are clear. To decode this cipher, we need to follow these steps."

1. Filter words that start with the letter `s` but are **not followed by** `a`.
2. Calculate the frequency of each character in these filtered words.
3. Shift each frequency count by adding it to the ASCII value of `'a'` to map it to a letter in the alphabet.
4. Output the final flag as a sequence of these mapped letters for each character in the alphabet (`a-z`).

The final flag will be in the format: `CTF{mapped_letter_sequence}`

Challenge Details

Step 1: Filter Words

Use `grep` to extract all words from the file `script.txt` that start with `s` but are **not followed by** `a`. For example:

- Valid words: `script`, `shell`, `system`
- Invalid words: `sail`, `sand`, `sauce`

NOTE: WORDS SHOULD BE UNIQUE

Step 2: Calculate Character Frequencies

Use `awk` or other tools to calculate the frequency of each character in the filtered words. For example, if the filtered words are `script` and `shell`, the character frequencies would be:

- `s: 2, c: 1, r: 1, i: 1, p: 1, t: 1, h: 1, e: 1, l: 2`

Step 3: Shift Frequencies by Adding 'a'

For each character's frequency, add the count to the ASCII value of 'a' (97) to map it to a letter in the alphabet. For example:

- If the frequency of s is 2, then 2 + 'a' maps to 'c' (since 97 + 2 = 99, which is 'c').
- If the frequency of c is 1, then 1 + 'a' maps to 'b'.

Step 4: Output the Final Flag

Generate the final flag as a sequence of these mapped letters for each character in the alphabet (a-z). If a letter does not appear in the filtered words, its mapped letter will be 'a' (since the frequency is 0, and 0 + 'a' maps to 'a').

The final flag will be in the format: CTF{mapped_letter_sequence}

Sample Test Case

Input File (script.txt)

Step 1: Filter Words

Filtered words: script, shell, system

Step 2: Calculate Character Frequencies

Character frequencies:

- s: 4, c: 1, r: 1, i: 1, p: 1, t: 1, h: 1, e: 2, l: 2, y: 1, m: 1

Step 3: Shift Frequencies by Adding 'a'

Mapped letters:

- s: 4 + 97 = 101 → 'e'
- c: 1 + 97 = 98 → 'b'
- r: 1 + 97 = 98 → 'b'
- i: 1 + 97 = 98 → 'b'
- p: 1 + 97 = 98 → 'b'
- t: 1 + 97 = 98 → 'b'

- `h: 1 + 97 = 98 → 'b'`
- `e: 2 + 97 = 99 → 'c'`
- `l: 2 + 97 = 99 → 'c'`
- `y: 1 + 97 = 98 → 'b'`
- `m: 1 + 97 = 98 → 'b'`

Step 4: Generate the Final Flag

Final flag: `CTF{aabacaabbaacbaababebaaaaba}`

Q2 - Background Story

In the ancient city of Palindromia, Ansh discovered a mysterious text file in the library. It was filled with thousands of words and a note that read: "Find the palindromes to uncover the truth." Ansh tirelessly searched for words like civic and radar, piecing together the puzzle. When he solved it, a hidden compartment revealed a treasure map. Excited, Ansh prepared for the greatest adventure of his life. To solve the puzzle, Ansh must follow these steps:

- **Find all palindromic words** in the file.
- **Filter words with odd lengths** and extract their middle letters.
- **Check if the middle letters** can form another palindromic word.
- **Generate the final flag** based on the result: - If the middle letters form a palindrome, concatenate all even-length palindromic words. - Otherwise, concatenate only the first half of each even-length palindromic word.
- The final flag will be in the format: `CTF{final_flag_output}`

Challenge Details

Step 1: Find Palindromic Words

A palindromic word reads the same forwards and backwards (e.g., `radar`, `level`). Use tools like `grep` or `awk` to extract all palindromic words from the file `script.txt`.

NOTE: WORDS SHOULD BE UNIQUE

Step 2: Filter Odd-Length Words and Extract Middle Letters

From the list of palindromic words, filter those with odd lengths (e.g., `radar` has length 5). Extract the middle letter of each odd-length word. For example: - `radar` → middle letter is `d`. - `level` → middle letter is `v`.

Step 3: Check if Middle Letters Form a Palindrome

Combine the middle letters into a single word and check if it is a palindrome. For example: - If the middle letters are `d`, `v`, and `e`, the combined word is `dve`, which is not a palindrome. - If the middle letters are `a`, `b`, and `a`, the combined word is `aba`, which is a palindrome.

Step 4: Generate the Final Flag

Based on the result from Step 3: - **If the middle letters form a palindrome**, concatenate all even-length palindromic words to form the final flag. - **If the middle letters do not form a palindrome**, concatenate only the first half of each even-length palindromic word to form the final flag.

The final flag will be in the format: `CTF{final_flag_output}`

Sample Test Case

Input File (`script.txt`)

Step 1: Find Palindromic Words

Palindromic words: `radar`, `level`, `civic`, `deed`, `noon`, `stats`, `rotor`, `madam`, `racecar`, `refer`

Step 2: Filter Odd-Length Words and Extract Middle Letters

Odd-length palindromic words and their middle letters:

- `radar` → `d`
- `level` → `v`
- `civic` → `v`
- `stats` → `a`
- `rotor` → `t`
- `madam` → `d`
- `racecar` → `e`

Middle letters: `d`, `v`, `v`, `a`, `t`, `d`, `e`

Step 3: Check if Middle Letters Form a Palindrome

Combined word: `dvvadtde` This is **not a palindrome**.

Step 4: Generate the Final Flag

Since the middle letters do not form a palindrome, concatenate the first half of each even-length palindromic word: - Even-length palindromic words: `deed`, `noon`, `refer` - First half of each word: - `deed` → `de` - `noon` → `no` - `refer` → `re`

Final flag: `CTF{denore}`

Another Sample Test Case

Input File (`script.txt`)

Step 1: Find Palindromic Words

Palindromic words: `madam`, `racecar`, `civic`, `level`, `radar`, `stats`, `deed`, `noon`, `refer`, `rotor`

Step 2: Filter Odd-Length Words and Extract Middle Letters

Odd-length palindromic words and their middle letters:

- `madam` → `d`
- `racecar` → `e`
- `civic` → `v`
- `level` → `v`
- `radar` → `d`
- `stats` → `a`
- `rotor` → `t`

Middle letters: `d`, `e`, `v`, `v`, `d`, `a`, `t`

Step 3: Check if Middle Letters Form a Palindrome

Combined word: `devvdat` This is **not a palindrome**.

Step 4: Generate the Final Flag

Since the middle letters do not form a palindrome, concatenate the first half of each even-length palindromic word: - Even-length palindromic words: `deed`, `noon`, `refer` - First half of each word: - `deed` → `de` - `noon` → `no` - `refer` → `re`

Final flag: `CTF{denore}`

BONUS - Background Story

In the ancient library of Scriptoria, Kevin and Neel uncovered a text file containing a cryptic puzzle. To solve it, they filtered words starting with vowels and calculated unique patterns using character counts and binary conversions. Each result updated a mysterious frequency array, unlocking hidden clues. After completing the array, they summed its values to reveal the final flag. Together, Neel and Kevin uncovered a forgotten secret of Scriptoria.

1. **Filter words** that start with a vowel and do not end with a consonant.
2. For each filtered word:
   - Find the number of unique characters, say x.
   - Traverse the first half of the word and count the number of letters less than ''m'', say y.
   - Traverse the second half of the word and count the number of letters greater than ''m'', say z.
   - Concatenate the binary form of y and z to form a binary string.
   - Convert the binary string to a decimal number.
   - Update the x-th position of a 26-sized frequency array with the maximum of the existing value and the decimal number.
3. **Report the sum** of the frequency array as the final flag.

The final flag will be in the format: `CTF{final_flag_output}`

—

Challenge Details

Step 1: Filter Words

Filter words from the file `words.txt` that start with a vowel (`a, e, i, o, u`) and do not end with a consonant. For example: - Valid words: `apple`, `echo`, `idea` - Invalid words: `banana`, `cat`, `dog`

Step 2: Process Each Filtered Word

For each filtered word:

1. **Find the number of unique characters (x)**: For example, the word `apple` has 4 unique characters (`a, p, l, e`).
2. **Traverse the first half and count letters less than ''m'' (y)**: For `apple`, the first half is `ap`. Letters less than ''m'' are `a` and `p`, so $y = 2$.
3. **Traverse the second half and count letters greater than ''m'' (z)**: For `apple`, the second half is `ple`. Letters greater than ''m'' are `p` and `l`, so $z = 2$.
4. **Concatenate binary form of y and z**: For $y = 2$ and $z = 2$, the binary strings are `10` and `10`, respectively. The concatenated binary string is `1010`.
5. **Convert the binary string to a decimal number**: The binary string `1010` is `10` in decimal.
6. **Update the frequency array**:
   - If the x-th position of the frequency array is currently `0`, update it to `10`.
   - If the x-th position already has a value, update it to the maximum of the existing value and `10`.

Step 3: Report the Final Flag

After processing all filtered words, sum the values in the frequency array and report it as the final flag.

The final flag will be in the format: `CTF{final_flag_output}`

Sample Test Case

Input File (`script.txt`)

```
apple
echo
idea
banana
cat
dog
```

Step 1: Filter Words

Filtered words: `apple`, `echo`, `idea`

Step 2: Process Each Filtered Word

1. **Word:** `apple`
   - Unique characters: `a, p, l, e` $\rightarrow x = 4$
   - First half: `ap` $\rightarrow$ Letters less than ''m'': `a, p` $\rightarrow y = 2$
   - Second half: `ple` $\rightarrow$ Letters greater than ''m'': `p, l` $\rightarrow z = 2$
   - Binary string: `10` (for $y = 2$) + `10` (for $z = 2$) = `1010`
   - Decimal: `1010` $\rightarrow$ `10`
   - Update frequency array: array[4] $= \max(0, 10) = 10$
2. **Word:** `echo`
   - Unique characters: `e, c, h, o` $\rightarrow x = 4$
   - First half: `ec` $\rightarrow$ Letters less than ''m'': `e, c` $\rightarrow y = 2$
   - Second half: `ho` $\rightarrow$ Letters greater than ''m'': `h` $\rightarrow z = 1$
   - Binary string: `10` (for $y = 2$) + `1` (for $z = 1$) = `101`
   - Decimal: `101` $\rightarrow$ `5`
   - Update frequency array: array[4] $= \max(10, 5) = 10$
3. **Word:** `idea`
   - Unique characters: `i, d, e, a` $\rightarrow x = 4$
   - First half: `id` $\rightarrow$ Letters less than ''m'': `i, d` $\rightarrow y = 2$
   - Second half: `ea` $\rightarrow$ Letters greater than ''m'': `e` $\rightarrow z = 1$
   - Binary string: `10` (for $y = 2$) + `1` (for $z = 1$) = `101`
   - Decimal: `101` $\rightarrow$ `5`
   - Update frequency array: array[4] $= \max(10, 5) = 10$

Step 3: Report the Final Flag

Frequency array:

$$[0, 0, 0, 0, 10, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$$

Sum of the array:

$$0 + 0 + 0 + 0 + 10 + 0 + \cdots + 0 = 10$$

Final flag: `CTF{10}`

Hit Submit

Part 1

```
lab2-part1
|-- sample.txt
|-- data.csv
|-- grep-command-1.sh
|-- grep-output-1.txt
.
.
|-- grep-command-5.sh
|-- grep-output-5.txt
|-- awk-command-1.sh
|-- awk-output-1.txt
.
.
|-- awk-command-4.sh
|-- awk-output-4.txt
|-- sed-command-1.sh
|-- sed-output-1.txt
.
.
|-- sed-command-6.sh
|-- sed-output-6.txt
```

Part 2 (CTF Based)

```
lab2-part2
|-- script.txt
|-- script-1.sh
|-- flag-1.txt
|-- script-2.sh
|-- flag-2.txt
|-- script-3.sh
|-- flag-3.txt
```

Put both folders in rollno and zip the rollno directory

```
rollno
|- lab2-part1
|- lab2-part2
```
