# LAB TASK:5-423132

1. Given a log file with the format:
   [Timestamp] [Log Level] [Module] [Message].

Write an awk command to extract only ERROR messages along with their timestamps.

awk 'condition { action }' file
matches the pattern.
Checks error in any line

```
student@ai-HP-ProDesk-600-G4-MT:~$ awk '/ERROR/ {print $1, $2}' logfile.txt
[2024-02-0112:05:23] ERROR:
[2024-02-0112:15:50] ERROR:
```

2. Given a CSV file with tab-separated values.

| ID | Math | Science | English |
|---|---|---|---|
| 1 | 78 | 85 | 90 |
| 2 | 82 | 80 | 88 |
| 3 | 75 | 92 | 95 |

Write an awk script to compute the average of each subject.

Nano- to create the files in runtime

semicolon is must.

```
student@ai-HP-ProDesk-600-G4-MT:~$ nano marks.csv
```

```
student@ai-HP-ProDesk-600-G4-MT:~$ awk  '{math+=$2; science+=$3; english+=$4; count
++} END {print "Math:", math/count,
"Science:", science/count, "English:", english/count}' marks.csv
Math: 47 Science: 51.4 English: 54.6
```

3. Given a server log with IP addresses:

192.168.1.1 - - [17/Feb/2025:12:00:01] "GET /index.html"
192.168.1.2 - - [17/Feb/2025:12:05:23] "POST /login"
192.168.1.1 - - [17/Feb/2025:12:10:45] "GET /dashboard"

Write an awk script to count occurrences of each IP.

```
student@ai-HP-ProDesk-600-G4-MT:~$ awk '{print $1}' server.log | sort | uniq -c
      1 192.168.1.10
      1 192.168.1.11
      1 192.168.1.12
      1 192.168.1.13
```

`|`: This is the **pipe** operator, which passes the output from the previous command (`awk`) into the next command (`sort`) **`sort`**: This sorts the lines in ascending order. **`uniq -c`**: This command filters out duplicate lines

# LAB TASK:5-423132

4. Given lines of text: apple banana cherry dog cat elephant . Write a sed command to swap the first and last words.

```
student@ai-HP-ProDesk-600-G4-MT:~$ sed -E 's/^([^ ]+) (.*) ([^ ]+)$/\3 \2 \1/' file
.txt
 quick brown fox jumps over the lazy The
 journey of a thousand miles begins with a single A
 world! This is a simple test Hello
 and awk are powerful text-processing Sed
 expressions are very useful in Regular
 file contains multiple lines for testing This
```

**^**: Asserts the start of the line.

- **([^ ]+)**: Matches one or more characters that are **not spaces**. This is captured as **Group 1**.
- **(.*)**: Matches the **middle part** of the line, which can contain anything (including spaces). This is captured as **Group 2**.
- **([^ ]+)**: Matches one or more characters that are **not spaces** again. This is captured as **Group 3**.
- **$**: Asserts the end of the line.

5. Given a file with duplicate words:

hello hello world this is is a test test

Write a sed command to remove consecutive duplicate words.

```
student@ai-HP-ProDesk-600-G4-MT:~$ sed -E 's/\b([a-zA-Z]+) \1\b/\1/g' file.txt
The quick brown fox jumps over the lazy dog.
A journey of a thousand miles begins with a single step.
Hello world! This is a simple test file.
Sed and awk are powerful text-processing tools.
Regular expressions are very useful in scripting.
This file contains multiple lines for testing purposes.
student@ai-HP-ProDesk-600-G4-MT:~$
```

S:This is a **substitution** command that looks for a pattern in each line and replaces it with something else.

**\b**: A **word boundary** **\1**: This is a **backreference**

**g**: The **global flag**

# LAB TASK:5-423132

6. Given a file containing email addresses:

john.doe@example.com
alice123@gmail.com

Write a sed command to mask the usernames (before @).

**`[^@]`**: A negated character

**`@****`**: This is the replacement part of the substitution.

```
student@ai-HP-ProDesk-600-G4-MT:~$ sed -E 's/^[^@]+/@****/' emails.txt
@****@gmail.com
@****@yahoo.com
@****@outlook.com
@****@gmail.com
@****@hotmail.com
```

7. Given a text file, find and print the most frequent word and its count.

```
student@ai-HP-ProDesk-600-G4-MT:~$ tr ' ' '\n' < file.txt | sort | uniq -c | sort -
nr | head -1
      3 a
student@ai-HP-ProDesk-600-G4-MT:~$
```

**`tr`**: This command is used for translating or replacing characters.
**`sort -nr`**: This sorts the output from `uniq -c` in **numeric**
**`head -1`**: This command only outputs the **first line**

8. Given a file with repeated lines in different cases, extract unique lines (case-insensitive and case-sensitive)

u-unique
-f : case insensitive

```
student@ai-HP-ProDesk-600-G4-MT:~$ sort -f -u file.txt
```

```
student@ai-HP-ProDesk-600-G4-MT:~$ sort -u file.txt
A journey of a thousand miles begins with a single step.
Hello world! This is a simple test file.
Regular expressions are very useful in scripting.
Sed and awk are powerful text-processing tools.
The quick brown fox jumps over the lazy dog.
This file contains multiple lines for testing purposes.
```

9. Given a file, reverse the order of words in each line.

```
student@ai-HP-ProDesk-600-G4-MT:~$ awk '{for(i=NF; i>0; i--) printf("%s ", $i); pri
nt ""}' file.txt
 lazy the over jumps fox brown quick The
 single a with begins miles thousand a of journey A
 test simple a is This world! Hello
 text-processing powerful are awk and Sed
 in useful very are expressions Regular
 testing for lines multiple contains file This
```

# LAB TASK:5-423132

10. Given a file, print the longest line and its length.
    **$0**: This refers to the **entire current line**.

```
student@ai-HP-ProDesk-600-G4-MT:~$ awk '{if(length > max) {max=length; line=$0}} EN
D {print line, max}' file.txt
 57ourney of a thousand miles begins with a single step.
student@ai-HP-ProDesk-600-G4-MT:~$
```

11. Given login records, extract usernames and sort by frequency.

```
student@ai-HP-ProDesk-600-G4-MT:~$ awk '{print $1}' login.txt | sort | uniq -c | so
rt -nr
      3 bob
      3 alice
      1 david
      1 charlie
```

12. Convert YYYY-MM-DD format to DD-MM-YYYY.

```
student@ai-HP-ProDesk-600-G4-MT:~$ sed -E 's/([0-9]{4})-([0-9]{2})-([0-9]{2})/\3-\2
-\1/' file.txt
18-02-2025
31-12-2024
15-05-2023
```

13. Normalize spacing by replacing multiple spaces with a single space.

```
student@ai-HP-ProDesk-600-G4-MT:~$ sed -E 's/[[:space:]]+/ /g' file.txt
2025-02-1 8
2024-12-31
2023-05-1 5
```

14. Given a file with phone numbers, mask all but the last 4 digits.

```
student@ai-HP-ProDesk-600-G4-MT:~$ sed -E 's/[0-9]{6}/******/g' phones.txt
******5318
******2581
******6834
******4890
******4333
******2345
******4322
******3211
student@ai-HP-ProDesk-600-G4-MT:~$
```

15. Extract text inside parentheses.

K: keeps `[^)]+'`: This matches one or more characters that are not a closing parenthesis `)`

# LAB TASK:5-423132

```
student@ai-HP-ProDesk-600-G4-MT:~$ grep -oP '\(\K[^)]+' file.txt
first example
second example
student@ai-HP-ProDesk-600-G4-MT:~$
```

16. Reverse the characters in each line of a file.

```
student@ai-HP-ProDesk-600-G4-MT:~$ rev file.txt
.)elpmaxe tsrif( si sihT
.)elpmaxe dnoces( rehtonA
```

17. Extract the 3rd word from each line, if it exists.

```
student@ai-HP-ProDesk-600-G4-MT:~$ awk '{if(NF>=3) print $3}' file.txt
(first
example).
student@ai-HP-ProDesk-600-G4-MT:~$
```

18. Find and print lines containing words that are anagrams.

```
student@ai-HP-ProDesk-600-G4-MT:~$ awk '{split($0, words, " "); for (i in words) {a
sort(words); sorted=sorted words[i]} print sorted,
> $0}' file.txt | sort | uniq -w 10
 s(secondAnotherexample).
```

19. Remove lines that contain only numbers.

```
student@ai-HP-ProDesk-600-G4-MT:~$ sed -iE 's/^[0-9]/ /g' phones.txt
student@ai-HP-ProDesk-600-G4-MT:~$
```

20. Identify the word that appears most frequently and its count.

```
student@ai-HP-ProDesk-600-G4-MT:~$ tr ' ' '\n' < file.txt | sort | uniq -c | sort -
nr | head -1
      2 example).
student@ai-HP-ProDesk-600-G4-MT:~$
```

21. Transform hello_world_example to helloWorldExample.

```
student@ai-HP-ProDesk-600-G4-MT:~$ awk -F_ '{for (i=2; i<=NF; i++) $i=toupper(subst
r($i,1,1)) substr($i,2)} 1' OFS='' file.txt
This is (first example).
Another (second example).

student@ai-HP-ProDesk-600-G4-MT:~$
```

22. Add Line Numbers Only to Non-Empty Lines

# LAB TASK:5-423132

```
student@ai-HP-ProDesk-600-G4-MT:~$ awk -F_ '{for (i=2; i<=NF; i++) $i=toupper(subst
r($i,1,1)) substr($i,2)} 1' OFS='' file.txt
This is (first example).
Another (second example).
```

23. Extract all email addresses from a file.

```
student@ai-HP-ProDesk-600-G4-MT:~$ grep -E -o '[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a
-zA-Z]{2,}' file.txt
student@ai-HP-ProDesk-600-G4-MT:~$
```

24. Replace All Digits with Corresponding Words: Convert numbers to words (1 → one, 2 → two).

sed -e 's/\b1\b/one/g' -e 's/\b2\b/two/g' -e 's/\b3\b/three/g' -e 's/\b4\b/four/g' -e 's/\b5\b/five/g' \

-e 's/\b6\b/six/g' -e 's/\b7\b/seven/g' -e 's/\b8\b/eight/g' -e 's/\b9\b/nine/g' \

-e 's/\b0\b/zero/g' file.txt

25. Strip all HTML tags, leaving only plain text.

sed -E 's/<[^>]+>//g' file.html

26. Replace Repeated Characters with a Single Instance

```
student@ai-HP-ProDesk-600-G4-MT:~$ sed -E 's/(.)\1+/\1/g' file.txt
This is (first example).
Another (second example).

student@ai-HP-ProDesk-600-G4-MT:~$
```

27. Extract Sentences Containing a Specific Word

```
student@ai-HP-ProDesk-600-G4-MT:~$ grep -E '\bword\b' file.txt
student@ai-HP-ProDesk-600-G4-MT:~$
```

28. Identify palindromes (words that read the same forward and backward) and wrap them in brackets

```
student@ai-HP-ProDesk-600-G4-MT:~$ awk '{for(i=1; i<=NF; i++) if ($i == rev($i))
 printf("[%s] ", $i); print ""}' file.txt
```

# LAB TASK:5-423132

29. Detect and replace consecutive repeated words with a single instance.

```
student@ai-HP-ProDesk-600-G4-MT:~$ sed -E 's/\b([a-zA-Z]+) \1\b/\1/g' file.txt
This is (first example).
Another (second example).

student@ai-HP-ProDesk-600-G4-MT:~$
```

30. Extract all unique words from a file, sort them alphabetically (ignoring case), and print them

```
student@ai-HP-ProDesk-600-G4-MT:~$ tr ' ' '\n' < file.txt | sort -u

Another
example).
(first
is
(second
This
student@ai-HP-ProDesk-600-G4-MT:~$
```