Experiment 8: Shell Programming

Name: Biswanbandya Mohanty Roll No.: 590029274 Date: 2025-11-1

Aim:

- To extend shell programming concepts by using conditional statements, advanced scripting constructs, and command-line arguments.
- To practice writing scripts that perform decision-making and parameter handling.

Requirements

- A Linux system with bash shell.
- Text editor and permission to create/execute shell scripts.

Theory

Conditional execution in shell scripts allows branching logic using `if`, `elif`, `else`, and `case` statements. Scripts can accept command-line arguments using `$1`, `$2`, … and `$@` for all arguments. Control flow constructs combined with user input and arguments allow dynamic and reusable scripts.

Procedure & Observations

Exercise 1: Using `if-else`

Task Statement:

Write a script to check whether a given number is positive, negative, or zero.

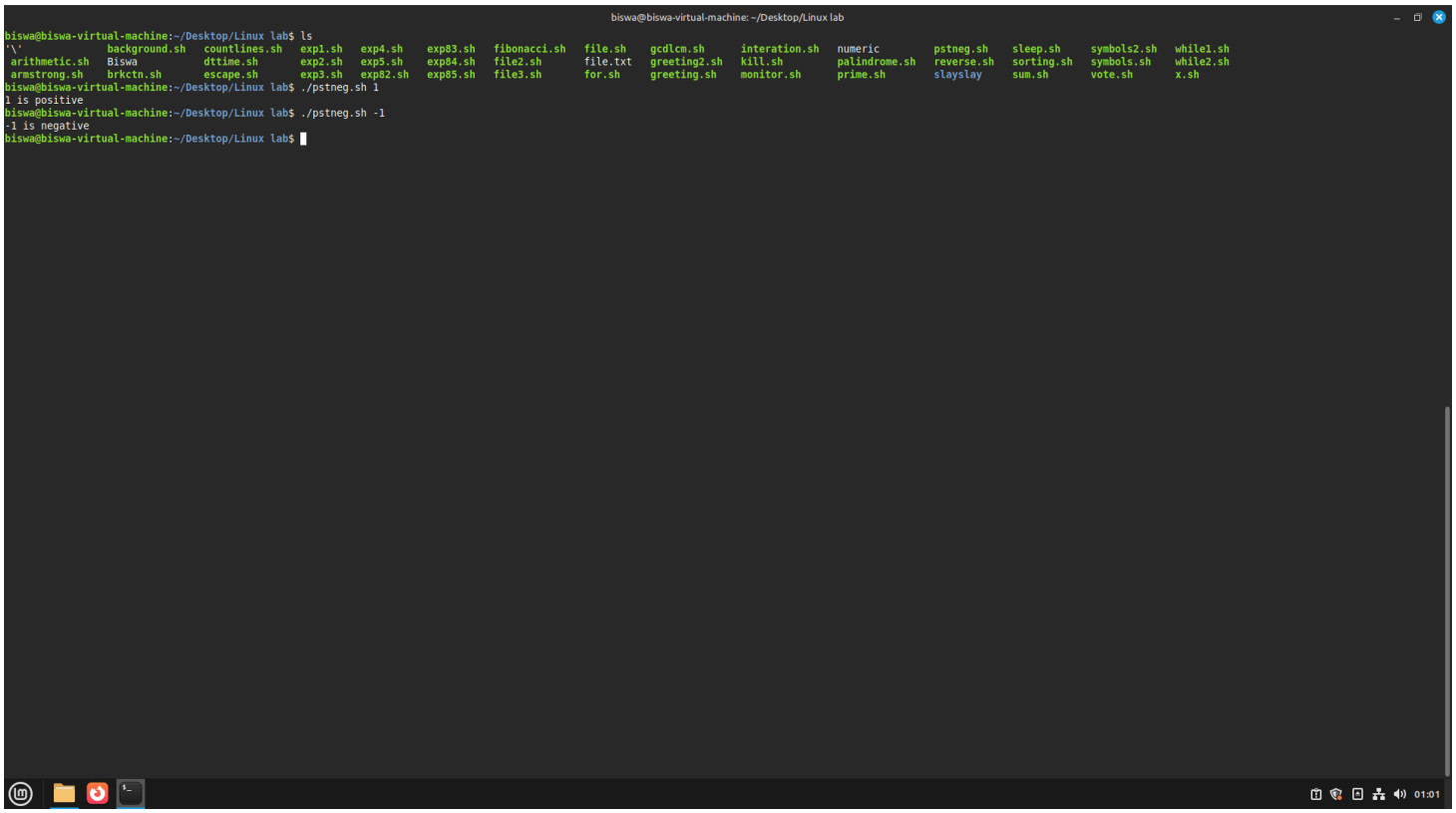
Explanation:

We used an `if-elif-else` construct to compare the number against 0.

Command(s):

```
#!/bin/bash
num=$1
if [ $num -gt 0 ]; then
  echo "$num is positive"
elif [ $num -lt 0 ]; then
  echo "$num is negative"
else
  echo "$num is zero"
fi
```

Output:



Exercise 2: Using `case`

Task Statement:

Write a script that takes a character as input and classifies it as vowel, consonant, digit, or special character.

Explanation:

The `case` statement provides pattern matching for multiple options.

Command(s):

```
#!/bin/bash
ch=$1
case $ch in
  [aeiouAEIOU]) echo "$ch is a vowel" ;;
  [bcdfghjklmnpqrstvwxyzBCDFGHJKLMNPQRSTVWXYZ]) echo "$ch is a consonant" ;;
  [0-9]) echo "$ch is a digit" ;;
  *) echo "$ch is a special character" ;;
esac
```

Output:

Exercise 3: Command-line arguments

Task Statement:

Write a script that accepts filename(s) as arguments and prints the number of lines in each file.

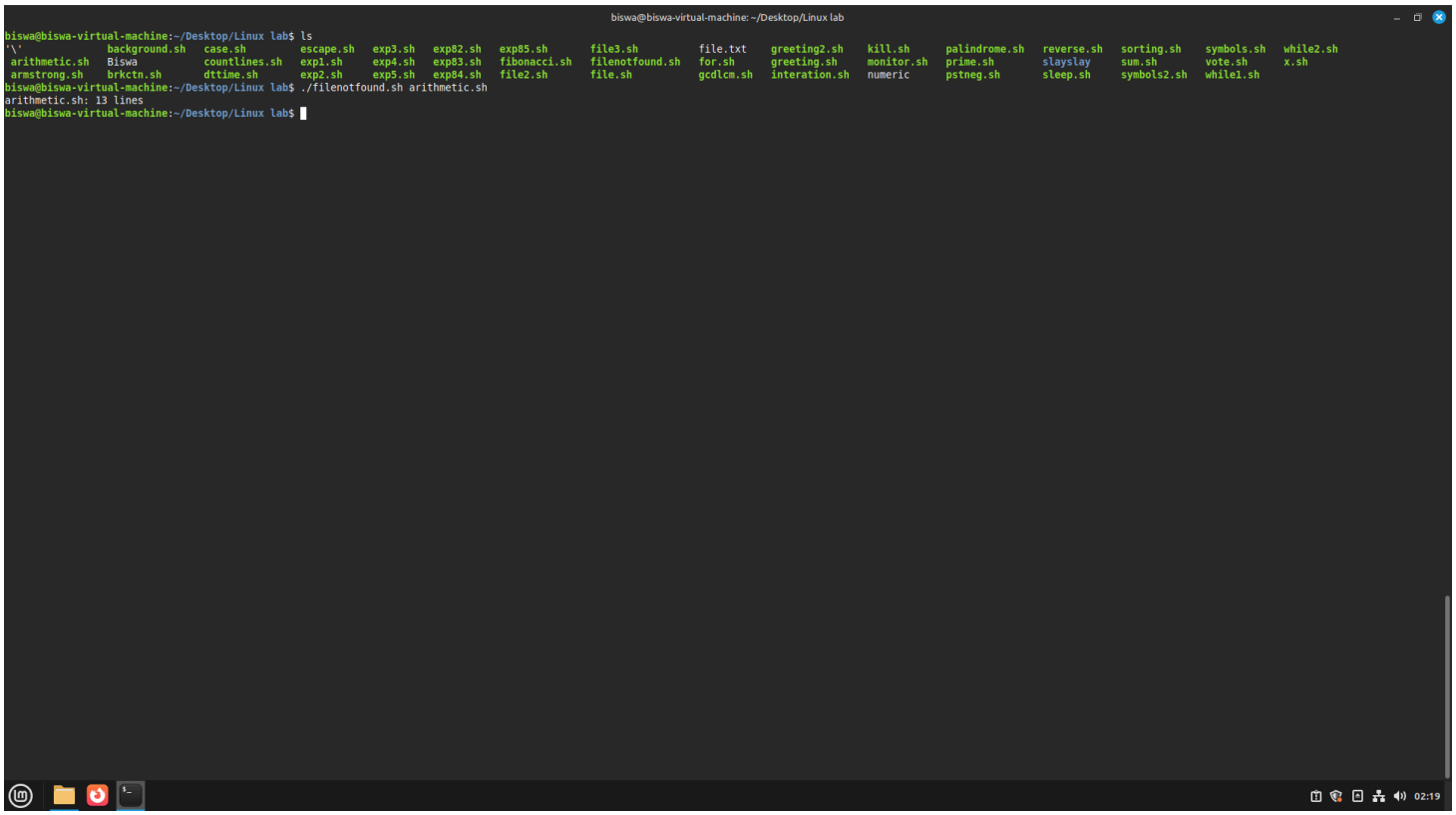
Explanation:

Command-line arguments are accessed using `$@` . Looping through each argument allows file-wise operations.

Command(s):

```
#!/bin/bash
for file in "$@"; do
  if [ -f "$file" ]; then
    echo "$file: $(wc -l < "$file") lines"
  else
    echo "$file not found"
  fi
done
```

Output:



Exercise 4: Nested conditionals

Task Statement:

Write a script to check if a year is a leap year.

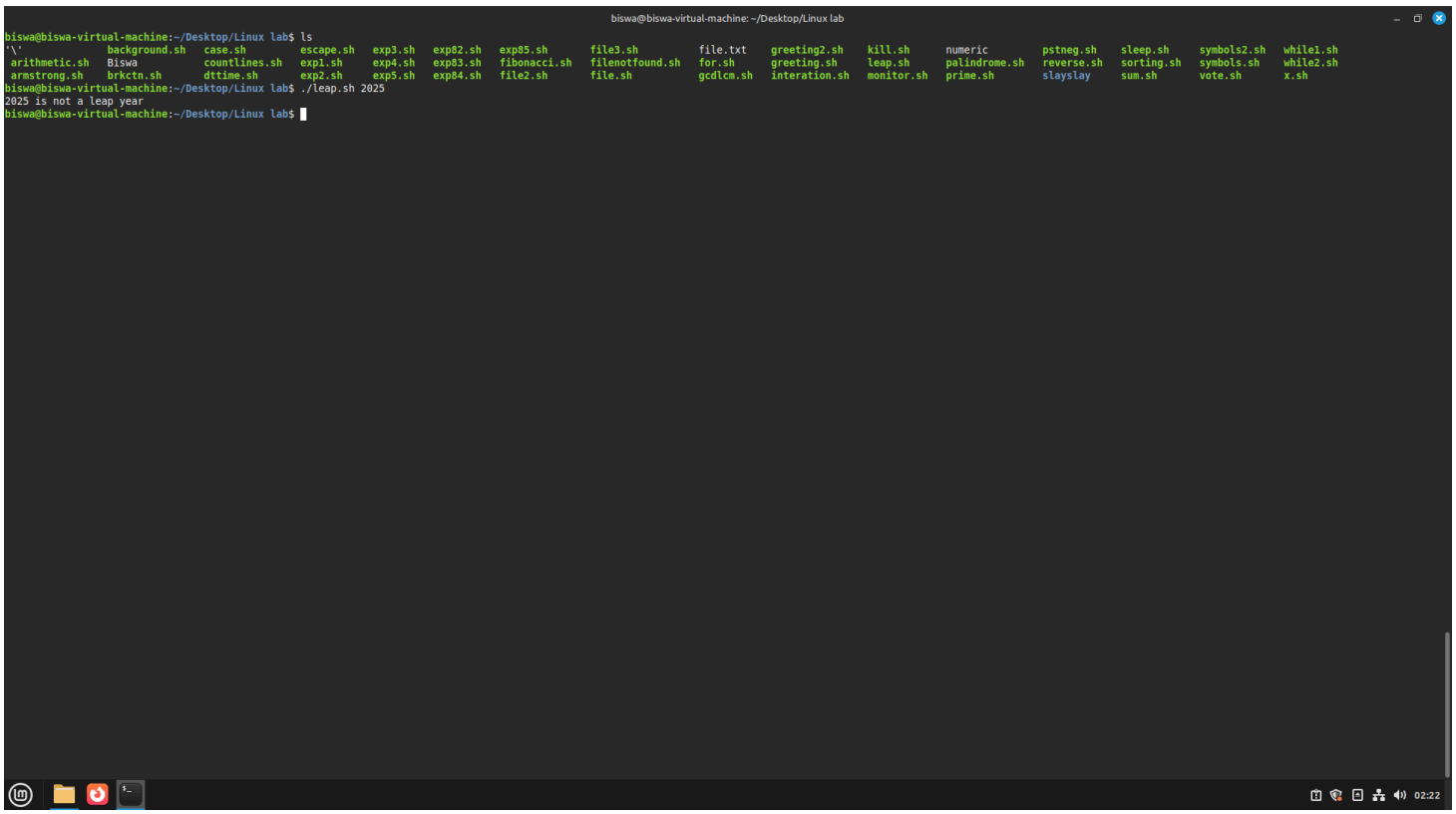
Explanation:

A leap year is divisible by 4, but if divisible by 100 it must also be divisible by 400.

Command(s):

```
#!/bin/bash
year=$1
if (( year % 400 == 0 )); then
  echo "$year is a leap year"
elif (( year % 100 == 0 )); then
  echo "$year is not a leap year"
elif (( year % 4 == 0 )); then
  echo "$year is a leap year"
else
  echo "$year is not a leap year"
fi
```

Output:



Task 1

Task Statement:

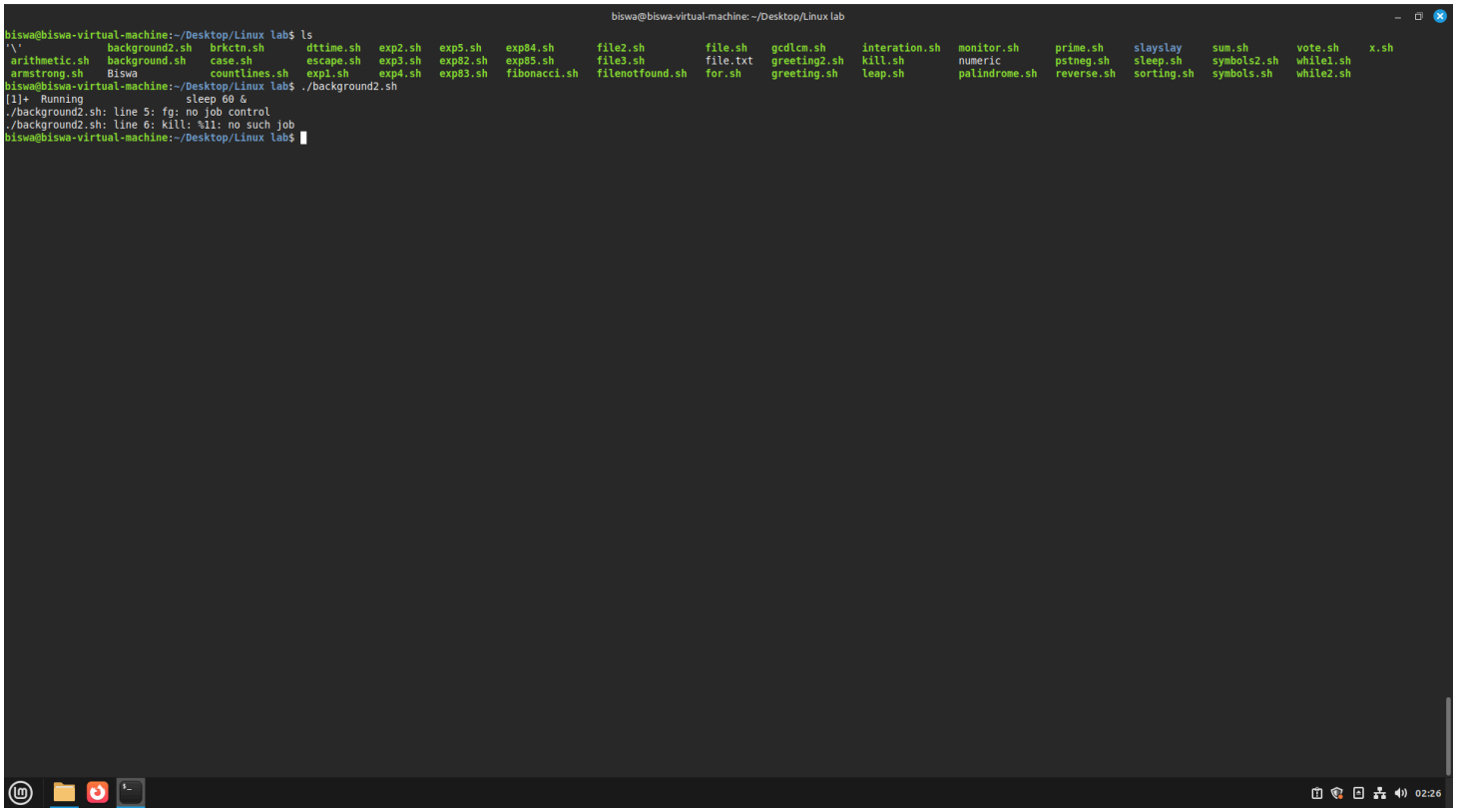
Write a script that starts a background job (e.g., `sleep 60`), lists all jobs, brings the job to the foreground, and then terminates it.

Command(s):

```
#!/bin/bash

sleep 60 &
jobs
fg %1
kill %1
```

Output:



Task 2

Task Statement:

Create a script that compares two files and displays whether their contents are identical or different.

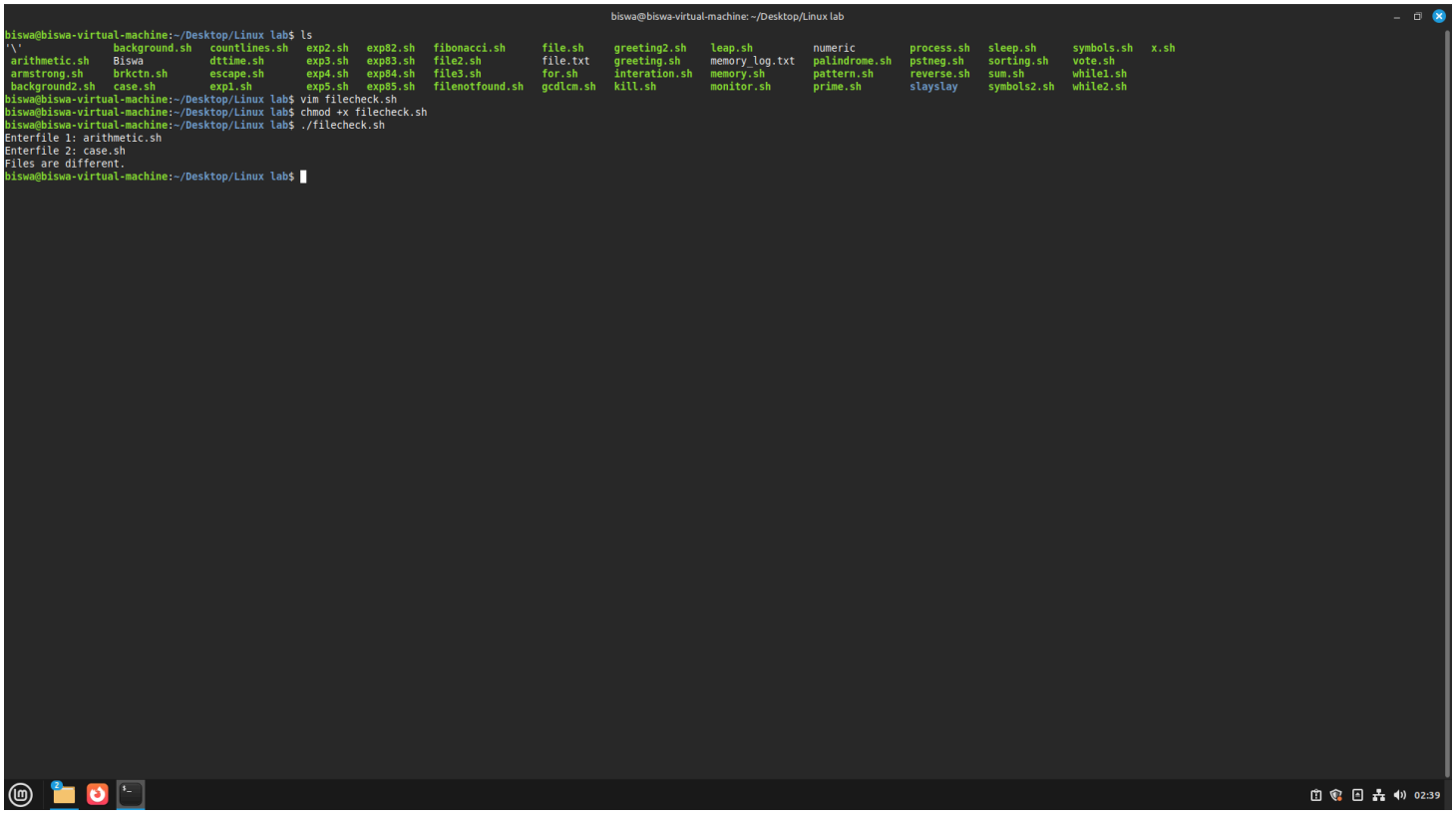
Command(s):

```
#!/bin/bash

read -p "Enterfile 1: " file1
read -p "Enterfile 2: " file2

if cmp -s "$file1" "$file2"; then
    echo "Files are identical."
else
    echo "Files are different."
fi
```

Output:



Task 3

Task Statement:

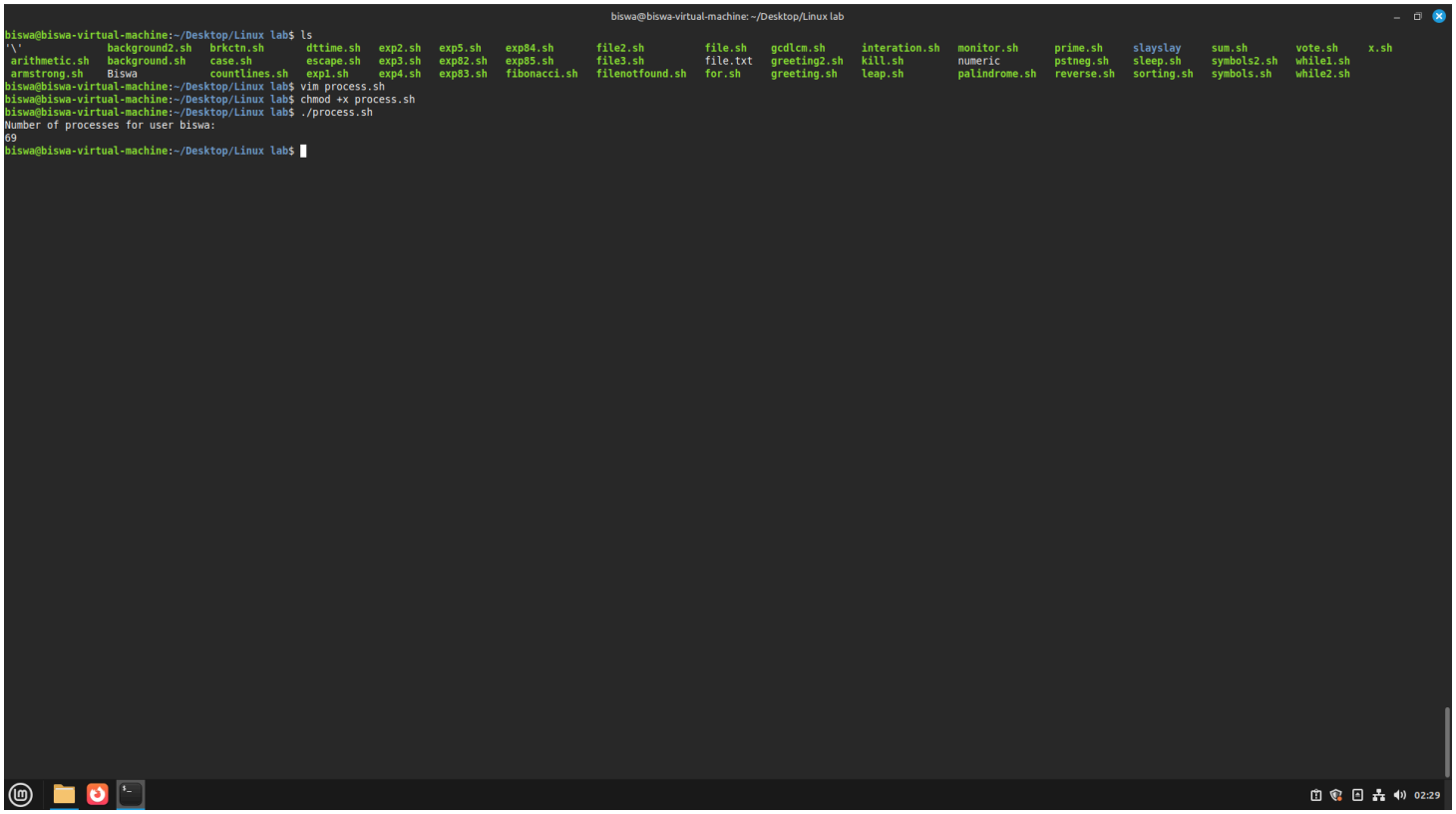
Write a script that counts the number of processes currently being run by your user.

Command(s):

```
#!/bin/bash

echo "Number of processes for user $USER:"
ps -u $USER | wc -l
```

Output:



Task 4

Task Statement:

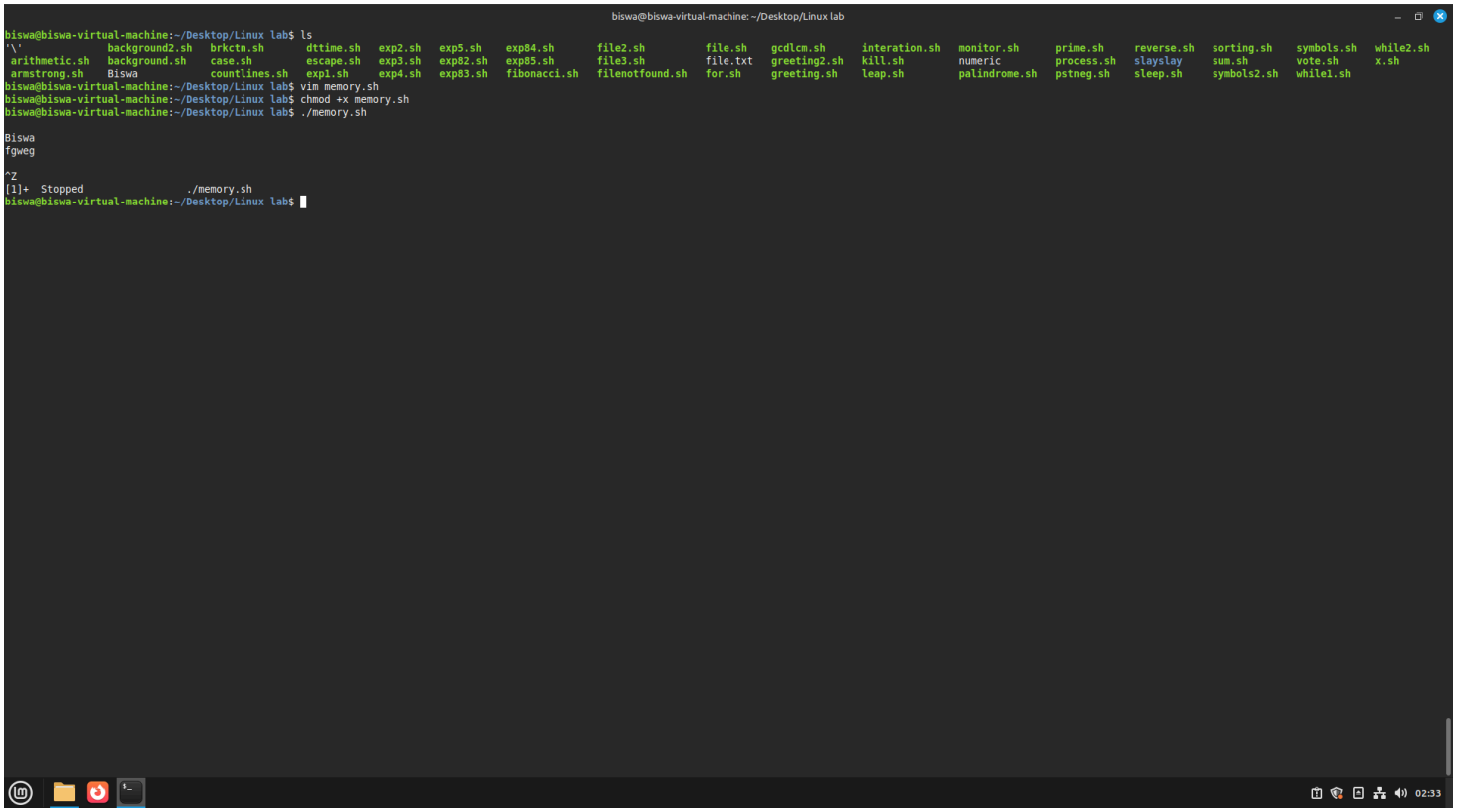
Develop a script that monitors memory usage every 5 seconds and logs it into a file.

Command(s):

```
#!/bin/bash

while true; do
    echo "Mem use  $(date)" >> memory_log.txt
    free -m >> memory_log.txt
    echo "---------------------------" >> memory_log.txt
    sleep 5
done
```

Output:



Task 5

Task Statement:

Write a script that prompts for a filename and a search pattern, then displays the count of matching lines.

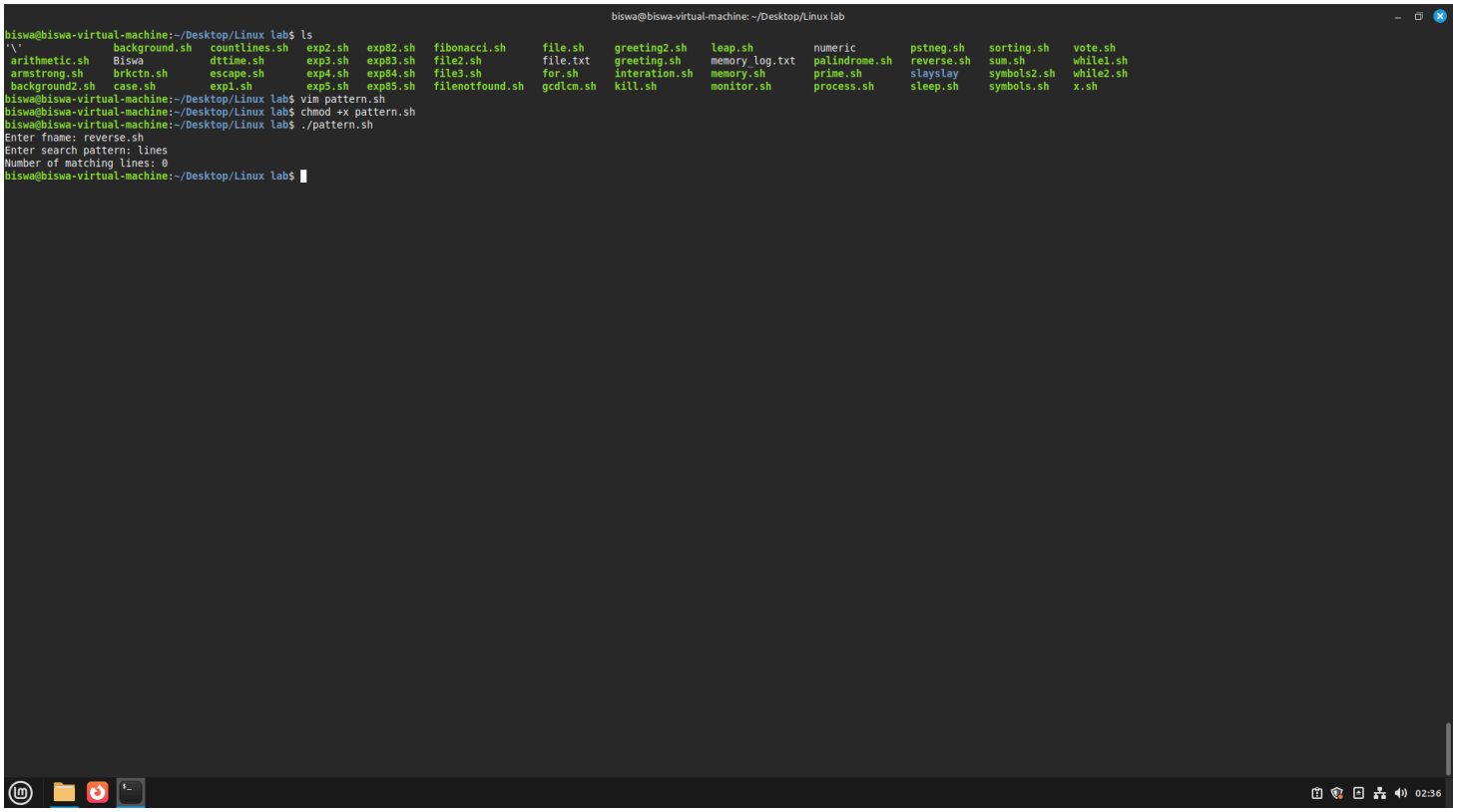
Command(s):

```
#!/bin/bash

read -p "Enter fname: " file
read -p "Enter search pattern: " pattern

count=$(grep -c "$pattern" "$file")
echo "Number of matching lines: $count"
```

Output:



Result

- Implemented conditional statements (`if-else` , `case`) in shell scripts.
- Practiced handling command-line arguments and nested conditions.
- Wrote reusable and flexible shell scripts.

Challenges Faced & Learning Outcomes

- Challenge 1: Forgetting to quote variables in conditions — resolved by using `"$var"` to avoid word splitting.
- Challenge 2: Pattern matching in `case` — practiced with multiple examples.
- challenge 3: Could not run experiment 2 perfectly.

Learning:

- Learned practical use of branching and decision-making in shell scripting.
- Understood command-line argument handling for automation.

Conclusion

This experiment extended shell programming by introducing decision-making and parameter handling. The scripts demonstrate the flexibility of shell programming for different use cases.