
Lab 13 - Shell Scripting

Problem statement

Write a Bash script named `analyze_logs.sh`. The script must inspect every file with extension `.log` in the current directory, clean each file, generate severity-specific extracts, classify each file by severity, detect duplicates, produce per-file summaries, and generate one consolidated global report. The script must rely only on shell scripting and standard Linux command-line tools.

A log line may contain one of the following uppercase severity tags: `INFO`, `WARNING`, `ERROR`, and `CRITICAL`. Only uppercase matches count. For example, `error` and `Warning` must not be counted.

Mandatory processing requirements

- First check whether any `.log` file exists in the current directory. If no such file is present, print `No log files found.` on the terminal and terminate.
- If log files exist, create the directory structure `reports/`, `reports/cleaned/`, `reports/errors/`, and `reports/summary/`.
- Before analysis, create a cleaned copy of each log file. Cleaning must:
 - replace consecutive spaces or tabs with a single space,
 - remove leading spaces,
 - remove trailing spaces,
 - delete blank lines.
- Analyze every `.log` file one by one.
- For each cleaned log file, compute:
 - total lines,
 - `INFO` count,
 - `WARNING` count,
 - `ERROR` count,
 - `CRITICAL` count.
- Create three severity-specific files in `reports/errors/`:
 - `<base>_warning.txt`
 - `<base>_error.txt`
 - `<base>_critical.txt`

Empty files must still be created when no matching lines exist.

- Classify each file with the priority:

SEVERE > DANGEROUS > ATTENTION > SAFE

A file is:

- SEVERE if it contains at least one CRITICAL line,
 - DANGEROUS if it contains no CRITICAL but at least one ERROR,
 - ATTENTION if it contains no CRITICAL and no ERROR but at least one WARNING,
 - otherwise SAFE.
- Create a duplicate report in `reports/summary/<base>_duplicates.txt`. A duplicate means the exact same cleaned line occurs more than once. The required format is:

count line

Example:

2 ERROR Disk failure

- Create a per-file summary report in `reports/summary/<base>_summary.txt` containing:
 - the file name,
 - all counts,
 - the classification,
 - the most frequent severity.
- The most frequent severity is the category with the highest count among INFO, WARNING, ERROR, and CRITICAL. If there is a tie, break the tie using the priority:

CRITICAL > ERROR > WARNING > INFO

If all four counts are zero, write NONE.

- Generate one global report `reports/final_report.txt` that contains:
 - the number of processed files,
 - a short section for each file,
 - overall totals across all files,
 - the file with maximum ERROR count,
 - the file with maximum CRITICAL count.

You can assume that test cases will not contain any scenario where there is a tie for these maxima.

-
- Display progress of file analysis on the terminal in the style:

```
Processing app.log ...
Classification: DANGEROUS
```

```
Processing case.log ...
Classification: SEVERE
```

... and finally:

```
Analysis complete.
Reports stored in reports/.
```

Example of cleaning and duplicate detection

Suppose the input file contains the following lines before cleaning:

```
INFO
Start

WARNING
Low memory
ERROR
Disk failure
ERROR          Disk failure
```

After cleaning, the script must treat the file as:

```
INFO Start
WARNING Low memory
ERROR Disk failure
ERROR Disk failure
```

The duplicate report for that file must therefore contain:

```
2 ERROR Disk failure
```

Required output structure

```
reports/
|-- cleaned/
|   '-- <base>_cleaned.log
|-- errors/
|   |-- <base>_warning.txt
```

```
| |-- <base>_error.txt
| '-- <base>_critical.txt
|-- summary/
| |-- <base>_duplicates.txt
| '-- <base>_summary.txt
'-- final_report.txt
```

Input and output specification

Input: There is no interactive input from the keyboard. The input is the set of .log files already present in the current working directory.

Output: The script generates the `reports/` directory tree shown above. It must also print progress information on the terminal. If there are no .log files, it prints only No log files found. and terminates.

Representative test cases

(For Reference. Will be removed from the problem statement given to the student)

Test case	Purpose	Key expectations
1	No .log files	Print only No log files found. and stop.
2	Single ATTENTION file	Count INFO and WARNING correctly; create empty ERROR and CRITICAL files.
3	Single DANGEROUS file with duplicates	Detect duplicate ERROR line; most frequent severity resolves correctly on ties.
4	Single SEVERE file with messy spaces and blank lines	Cleaning by <code>sed</code> is necessary before analysis; duplicate CRITICAL line appears in duplicate report.
5	Multiple mixed files	Correct global totals and tie-breaking for file with max ERROR / max CRITICAL.
6	Case sensitivity and empty file	Ignore lowercase forms; empty file produces SAFE classification and most frequent severity = NONE.

A sample detailed testcase is shown below.

Input file: `server.log`

INFO Boot

ERROR Disk failure
ERROR Disk failure
WARNING Retry requested
INFO Shutdown

Expected classification: DANGEROUS.

The duplicate report must contain:

2 ERROR Disk failure

The summary file must report:

- total lines = 5
- INFO count = 2
- WARNING count = 1
- ERROR count = 2
- CRITICAL count = 0
- most frequent severity = ERROR

Submission expectation

Submit a single executable Bash script named `analyze_logs.sh`. The script must run correctly when copied into a directory containing the testcase `.log` files.