LAB1 : Write Groovy scripts for data processing (lists/maps)

Groovy Data Processing with CSV (Lists/Maps)

Objective

- Parse a CSV file in Groovy
- Use list/map operations for filtering, grouping, and aggregation
- Export results as JSON (useful in Jenkins pipelines)
- Extend scripts with advanced challenges

Reference github repo url :

DevOps_NICE_22_24_sept/Day1 at master · cloud-dev-user/DevOps_NICE_22_24_sept

---

Step 1 — Create a CSV file

Save this as `builds.csv` :

```
name,time,status
service-a,120,SUCCESS
service-b,240,FAILURE
service-c,60,SUCCESS
service-d,300,UNSTABLE

```

Step 2 — Base Groovy Script (`csv-builds.groovy`)

```groovy
import groovy.json.JsonOutput

// If filename is passed as an argument, use it; else default
def filename = args ? args[0] : "builds.csv"
def file = new File(filename)

// Parse CSV rows (skip header)
def rows = file.readLines().drop(1).collect { line ->
    def parts = line.split(',')
    [name: parts[0], time: parts[1].toInteger(), status: parts[2]]
}

// Step A: total build time
def totalTime = rows*.time.sum()

// Step B: successful builds
def successful = rows.findAll { it.status == 'SUCCESS' }*.name

// Step C: group by status
def grouped = rows.groupBy { it.status }

// Step D: longest build
def longest = rows.max { it.time }

// Step E: prepare JSON output
def result = [
    totalTime: totalTime,
    success  : successful,
    grouped  : grouped,
    longest  : [name: longest.name, time: longest.time]
```

```
]

println JsonOutput.prettyPrint(JsonOutput.toJson(result))

```

Run with:

```
groovy csv-builds.groovy

```

Or with another file:

```
groovy csv-builds.groovy mydata.csv

```

Step 3 — Challenges

🔹 **Challenge 1: Failure Count per Status**

Add a counter map:

```
def failureCounts = [:].withDefault{0}
rows.each { build ->
    if (build.status != 'SUCCESS') {
        failureCounts[build.status]++
    }
}
result.failures = failureCounts

```

Output snippet:

```
"failures": {
  "FAILURE": 1,
  "UNSTABLE": 1
}

```

🔹 **Challenge 2: Filter Builds by Duration (> 100 sec)**

```
def longBuilds = rows.findAll { it.time > 100 }
result.longBuilds = longBuilds*.name

```

🔹 **Challenge 3: Accept CSV Filename as Parameter**

Handled in script already (`args[0]`). Try running with a different file.

🔹 **Challenge 4: Integrate with Jenkins Pipeline**

Instead of `new File()` , use Jenkins' built-in `readFile` :

Jenkinsfile (scripted pipeline):

```
node {
    stage('Parse CSV') {
        def csvText = readFile 'builds.csv'
        def rows = csvText.readLines().drop(1).collect { line ->
            def parts = line.split(',')
            [name: parts[0], time: parts[1].toInteger(), status:
parts[2]]
        }
```

```
8
9           def grouped = rows.groupBy { it.status }
10          echo "Grouped = ${grouped}"
11      }
12  }
13
```

🔹 **Challenge 5: Dynamic Reports**

Write JSON output to a file (`report.json`) and archive it in Jenkins:

```
1  def json = JsonOutput.prettyPrint(JsonOutput.toJson(result))
2  new File('report.json').text = json
3
```

In Jenkins pipeline:

```
1  writeFile file: 'report.json', text: json
2  archiveArtifacts artifacts: 'report.json'
3
```

🔹 **Challenge 6: Extend CSV with More Fields**

Add `team` to CSV:

```
1  name,time,status,team
2  service-a,120,SUCCESS,team1
3  service-b,240,FAILURE,team2
4  service-c,60,SUCCESS,team1
5  service-d,300,UNSTABLE,team2
6
```

Now group failures per team:

```
1  def failuresByTeam = [:].withDefault{0}
2  rows.each { build ->
3      if (build.status == 'FAILURE') {
4          failuresByTeam[build.team]++
5      }
6  }
7  result.failuresByTeam = failuresByTeam
8
```

Validation

Run the script with different CSVs and confirm JSON results.

In Jenkins, check archived artifacts for `report.json` .