Bangladesh University of Engineering and Technology

CSE 314: Operating Systems Sessional

Shell Scripting Assignment
Submission Organizer, Analyzer and Executor

1 Overview

Suppose you have recently joined as a teacher at an engineering university. You are in charge of evaluating an Algorithm assignment that students have submitted via Moodle. After downloading the submission files, you are in awe. No matter how many times you have reminded them to submit their files following a standard, some students paid no attention to that. Now you have to organize the submissions, analyze their code quality, and execute them. Luckily, you are an expert in shell scripting. You will use your expertise to automate these tasks and save your precious time.

Your assignment has three parts:

A. Organize the code files into directories by file types and student ID.

B. Analyze code metrics including line count and comment count.

C. Execute the codes with test cases and match them with the expected results.

You will be provided with:

1. Submission folder (`submissions`)
2. Test case folder (`tests`)
3. Answer folder (`answers`)

2 Task A: Organize

Inside the submissions folder, there will be multiple zipped files – one zipped file for each student (see the helping materials section to unzip). Inside the zipped file are the files submitted by the student. The name of the zipped file will be in the following format:

`<Full Name>_<Serial Number>_submission_<Student ID>.zip`

Both the serial number and student ID will be of 7 digits. Students will submit codes in C (`.c` extension), C++ (`.cpp` extension), Python (`.py` extension), and Java (`.java` extension). It is guaranteed that one student has submitted only one file in either of C, C++, Python, or Java languages. There may be other files ending in different extensions. You have to ignore those extra files. The code file of a student can be located inside any directory/subdirectory of the zipped file of the student.

You have to create a new target directory (`targets`). Inside the target directory, there will be four subdirectories - `C`, `C++`, `Python`, `Java`.

- Suppose, student 2105XXX has written his/her code in C. You have to create a new subdirectory inside `<target directory>/C/2105XXX` and copy the C file inside `<target directory>/C/2105XXX` and rename it to `main.c`.
- Suppose, student 2105XXX has written his/her code in C++. You have to create a new subdirectory inside `<target directory>/C++/2105XXX` and copy the C++ file inside `<target directory>/C++/2105XXX` and rename it to `main.cpp`.
- Suppose, student 2105XXX has written his/her code in Java. You have to create a new subdirectory inside `<target directory>/Java/2105XXX` and copy the .java file inside `<target directory>/Java/2105XXX` and rename it to `Main.java`.
- Suppose, student 2105XXX has written his/her code in Python. You have to create a new subdirectory inside `<target directory>/Python/2105XXX` and copy the .py file inside `<target directory>/Python/2105XXX` and rename it to `main.py`.

Refer to the Match/targets directory that has been provided to better understand the task.

3 Task B: Code Analysis

After organizing the code files, you need to analyze each student's code for the following metrics:

1. **Line Count**: Count the total number of lines in the main file.
2. **Comment Count**: Count the number of single-line comments in the file.
   - For C/C++: Starts with `//`
   - For Java: Starts with `//`
   - For Python: Starts with `#`
3. **Bonus Task - Function Count**: Count the number of functions/methods defined in the file.

These metrics will later be included in the final report.

4 Task C: Execute and Match

Inside the test case folder (`tests`), test cases are stored in the pattern of `test1.txt`, `test2.txt`, ..., `testN.txt` and so on. They are guaranteed to follow such naming conventions.

For each test file, there will also be an accepted answer file inside the answer folder (`answers`), stored in the pattern of `ans1.txt`, `ans2.txt`, ..., `ansN.txt` and so on. They are also guaranteed to follow such naming conventions.

Your task is to compile and run the codes you have organized. Note that Python codes do not require compiling. Refer to the helping materials section to compile and run the code files.

- For a C file, the compiled executable file must be named as `main.out`
- For a C++ file, the compiled executable file must be named as `main.out`
- For a Java file, the compiled class file must be named as `Main.class`

You have to store the output files of each test case inside the student's folder inside the organized target folder. The output files will follow the naming convention of `out1.txt`, `out2.txt`, ..., `outN.txt` and so on.

Then, you will match the output files with the corresponding answer files using the command `diff`. If there are any mismatches between an output file and an answer file, it will be considered as a failure in that test case.

Finally, you have to generate a CSV file (inside `<target directory>`, named `result.csv`) having columns: `student_id`, `student_name`, `language`, `matched`, `not_matched`, `line_count`, `comment_count`, `function_count`. For each student, you have to record his/her student ID, his/her programming language (C/C++/Python/Java), number of test cases matched, number of test cases that did not match, total number of lines in the main file, number of single-line comments, and number of functions/methods defined. It is to be noted that except the first three columns, inclusion of the other columns are dependent on the usage of the optional command line arguments. Refer to optional arguments for better clarity.

Tentative Algorithm

1. For each student
   (a) Find the code file type (C/C++/Python/Java)
   (b) Analyze the code metrics (line count, comment count, optionally function count)
   (c) Compile if needed
   (d) For each test case `test<i>.txt`
      i. Run the compiled file
      ii. Store the output in `out<i>.txt`
      iii. Match `out<i>.txt` with corresponding `ans<i>.txt` file
   (e) Record how many test cases matched, how many did not match
2. Generate a CSV file `<target directory>/result.csv`

5 Example Directory Structure

Take a look at the demonstration video for better understanding.

Download the "Shell-Scripting-Assignment-Files.zip" file from Moodle. After unzipping it, you will find two folders and the following

```
-> Workspace
   + submissions # submission folder
   + tests # test input folder
```

```
     * test1.txt
     * test2.txt
     * ...
     * testN.txt
   + answers # accepted output folder
     * ans1.txt
     * ans2.txt
     * ...
     * ansN.txt
-> Match
   + targets # this will not be provided during evaluation
     * C # student who submitted C code
       - 2105121
         + main.c
         + main.out
         + out1.txt
         + ...
         + outN.txt
       - 2105122
       - ...
     * C++ # student who submitted C++ code
       - 2105125
         + main.cpp
         + main.out
         + out1.txt
         + ...
         + outN.txt
       - ...
     * Python # student who submitted Python code
       - 2105128
         + main.py
         + out1.txt
         + ...
         + outN.txt
       - ...
     * Java # student who submitted Java code
       - 2105130
         + Main.java
         + Main.class
         + out1.txt
         + ...
         + outN.txt
       - ...
     * result.csv # result of each student
```

The `Workspace/submissions` folder consists of zipped files of individual students.

You aim to write a shell script (`organize.sh`) that will be executed inside the Workspace directory.

Mandatory Arguments (In Order):

1. Path of submission folder
2. Path of target folder (create target folder if it does not exist)
3. Path of test folder
4. Path of answer folder

Optional Arguments (In Order):

1. `-v`
    a. If provided, will print useful information while executing scripts.
    b. Refer to the demonstration video to follow exactly what to print and what not to print.
    c. You must only print what has been shown on the video.
2. `-noexecute`
    a. If provided, will not perform Task C.
    b. Note that, with this switch, no output files and executable files will be generated.
    c. The CSV file won't have `matched` and `not_matched` columns.
    d. Code metrics might still be calculated.
3. `-nolc`
    a. If provided, will not calculate line count in code metrics.
    b. The `line_count` column will not be included in the CSV file.
4. `-nocc`
    a. If provided, will not calculate comment count in code metrics.
    b. The `comment_count` column will not be included in the CSV file.
5. `-nofc`
    a. If provided, will not calculate function count in code metrics.
    b. The `function_count` column will not be included in the CSV file.
    c. Note that this is only applicable if you implement the bonus task.

Sample commands:

```
user@machine:~/Offline/Workspace$ ./organize.sh submissions targets tests
answers -v
```

After executing the `organize.sh` file, it will generate a new folder inside the Workspace directory named `targets`. Your script must generate `Workspace/targets` identical to the `Match/targets` directory.

```
user@machine:~/Offline/Workspace$ ./organize.sh submissions targets tests
answers -v -noexecute
user@machine:~/Offline/Workspace$ ./organize.sh submissions targets tests
answers -v -nolc -nocc
```

6 Additional Information

- Each Java file is guaranteed to have `Main` as the main class.
- Each code is guaranteed to compile and run without error/exception.
- You must generate a CSV file following the exact format that is provided. Order of student ID does not matter. The names of the columns must match.
- You must generate executable files, output files following the format. Failure to follow the format will result in a penalty.
- If the number of arguments is less than the required number, you must print a usage message showing how to use the script.

7 Helping Materials

7.1 Unzipping

- Use the command `unzip` to unzip files.
- Use `-d` switch to unzip to a specific folder.
- Refer to the `man unzip` for more.
- Test the command in the terminal before using it inside the script.

7.2 Substring Extraction by Pattern

- Run the following commands in a script and see what happens.

```
string=hello.world
echo ${string%.world}
```

7.3 Substring Extraction by Index

- Run the following commands in a script and see what happens.

```
string=hello.world
echo ${string:2:1}
echo ${string:2:2}
echo ${string:2:20}
echo ${string: -1} # mind the space before - sign
echo ${string: -4}
echo ${string:2: -1}
```

7.4 Difference between the Two Files

- Use the command `diff` to check if the two files are the same.
- Refer to the `man diff` for more.
- Test the command in the terminal before using it inside the script.

7.5 Run a C File

```
gcc file_name.c -o executable_name
./executable_name
```

7.6 Run a C++ File

```
g++ file_name.cpp -o executable_name
./executable_name
```

7.7 Run a Python Script

```
python3 file_name.py
```

7.8 Run a Java File

- `javac file_path.java`
- This will create a `.class` file at the same directory as the `.java` file
- `java -cp <directory_of_the_java_file> <name_of_main_class>`

7.9 Kill a Running Script from Inside the Script

```
kill -INT $$
```

8 Marks Distribution

Tasks	**Sub Task**	**Marks**
Task A	Organize Files by Type & ID	15
	Ignore other files	5
Task B	Code metrics (lines and comments)	15
	Function Count (Bonus)	5
Task C	Compile & Run Files	15
	Match Outputs	10
	Generate CSV with all required columns	10
	Target folder matches the specified format exactly	5
	Mandatory Arguments	10
	Optional Arguments	10
	Usage Message	5
Total		**100 (+5)**

9 Submission

- Create a directory by your 7 digit student name (2105XXX).
- Put your `organize.sh` file inside.
- Zip the folder, rename it to `2105XXX.zip`.
- Submit the zipped folder.

Deadline: 03 May, 2025. 11:45 PM.

10 Plagiarism Policy

- Plagiarism can lead to awarding of –100% marks.
- Work on the problem on your own.