PRCMP PL10
Shell scripts 3

November 2024

1. A script is required to restart a set of Docker containers specified by the user.

   The Docker application identifies each container by its name, which is a single word, such as `container_X` . The `docker` command enables operations on containers, typically using the following syntax:

   ```
   docker ACTION CONTAINER
   ```

   For instance, to restart the container `container_X`, the corresponding command would be:

   ```
   docker restart container_X
   ```

   Develop a script that fulfils the following requirements:

   R1: The user must provide the names of the containers to restart via the command line.

   R2: The containers must be restarted in the exact order specified on the command line.

   R3: For each container, the script must display a message on the screen indicating whether the operation succeeded or failed.

2. A developer involved in multiple projects needs to keep their local repositories frequently updated. Each day, he performs a tedious task: for every repository, (1) he navigates to the respective working directory, and (2) executes the command `git pull`.

   This process can be easily automated with a shell script. Such a script can be configured using a text file that lists the working directories of the projects.

   Assume there is a text file where each line contains the path to a repository directory, as shown in the example below:

   ```
   /home/joaosilva/projects/ProjectA/
   /home/joaosilva/projects/ProjectB/
   /home/joaosilva/projects/TopSecretProject/
   ```

   Write a shell script that satisfies the following requirements:

   R1: The user must specify the name of the file containing the repository directories on the command line.

   R2: If the file containing the repository directories cannot be accessed, the script must terminate with an error.

   R3: The command `git pull` must be executed for each valid repository directory.

   R4: If a directory is invalid, an error message must be displayed.

   HINT: A file can be read line by line into a variable as follows:

   ```
   for line in $(cat filename)
   do
       echo $line
   done
   ```

3. You are enrolled in a C programming course where you have been tasked with solving a series of exercises. Each exercise requires developing a program that satisfies the problem's requirements.

   To this end, you created a directory on your disk where you saved all the source code files for the solutions you have developed. Carefully, you named each file in a way that allows you to easily identify which exercise it corresponds to, such as `exercise_23.c`.

   The `gcc` command is used to compile C programs, following the generic syntax:

   ```
   gcc -o EXECUTABLE SOURCE.c
   ```

   For example, to create the executable program `exercise_23` by compiling the source file `exercise_23.c`, you would run the following command:

   ```
   gcc -o exercise_23 exercise_23.c
   ```

   However, compiling the various programs you have already developed can be a repetitive and time-consuming task, which can be streamlined using a shell script. This script should be stored and executed in the same directory where the exercise solutions are located.

   Develop a script that allows compiling a set of C programs specified by the user, meeting the requirements listed below:

   R1: The user must provide a sequence of program names (executables) on the command line. This sequence of names is passed as parameters to the *script*.

   R2: The programs must be compiled in the exact order specified on the command line.

   R3: An error message must be displayed if the source code file for a program is not accessible in the current working directory.

   R4: The script must indicate how many programs were compiled successfully without errors.

4. Sharing data between applications via text files is quite common. However, when a file contains international characters, interpretation issues may arise because the applications involved might use different character encodings.

   The `iconv` command allows text conversion between two character encodings by specifying the original encoding with the `-f` option (for "*from*") and the target encoding with the `-t` option (for "*to*"). The conversion result is sent to standard output. The example below demonstrates how to convert the text in the file `original.txt` from ISO-8859-1 to UTF-8 and save the result to the file `converted.txt`:

   ```
   iconv -f ISO-8859-1 -t UTF-8 original.txt > converted.txt
   ```

   We have an application that processes text files encoded in UTF-8. However, the files come from various sources, and some come from legacy systems and are encoded in ISO-8859-1.

   Develop a script to convert a set of text files from ISO-8859-1 to UTF-8, meeting the following requirements:

   R1: The user must provide (1) the name of the input file containing the original text and (2) the name of the output file for the converted text on the command line. The filenames must be passed in this specific order as parameters to the script.

   R2: The conversion should only proceed if the input file exists.

   R3: The conversion should only proceed if the output file does not already exist.

   R4: The *script* must indicate whether the conversion was successful. Otherwise, it should inform the user of the reason for the failure.

5. You are tasked with automating the process of running unit tests and then compiling a Java project. This process can be implemented as a pipeline where:

   - **Stage 1:** Run unit tests for the Java project.
   - **Stage 2:** If all tests pass, compile the Java project.

- **Stage 3:** Provide a summary of the results including the status of the unit tests and whether the compilation was successful.

The script must fulfil the following requirements.

- **Input.** The script receives from the command line exactly two arguments:
  - A directory containing the Java project files.
  - The unit test framework that should be compatible with JUnit: e.g., **mvn** for Maven projects or **gradle** for Gradle projects).
- **Pipeline Stages:**
  - **Stage 1:** Run the unit tests.
    - If using Maven, use the `mvn test` command.
    - If using Gradle, use the `gradle test` command.
    - The tests should be run first to ensure that the code is functioning correctly before compilation.
  - **Stage 2:** If all tests pass (exit code 0), compile the project using the appropriate build tool:
    - If using Maven: `mvn compile`
    - If using Gradle: `gradle build`
  - **Stage 3:** Provide a summary output:
    - Whether the unit tests passed or failed.
    - Whether the project compiled successfully or not.
- **Output:**
  - The log file **pipeline_log.txt** detailing the results of each stage.
  - If the tests fail, the pipeline stops, and no compilation occurs.
  - If the tests pass, the script attempts to compile the project and provides the result.