Assignment 1: Shell Scripting

CS 3424 - Systems Programming — Sam Silvestro - UTSA

For this assignment, you will use **bash** create a simple course catalog for administrators to update the details of each course offered by their department. The system will store basic information about each course, allowing the user to create, read, update, and delete them.

This assignment requires only the utilities used so far in the lecture notes. **Do not** use sed, awk, or any programming languages, utilities, or bash features not yet covered in the lecture notes.

Note that you **need not** validate the formatting of user input. For example, verifying whether a provided date field is consistent with `MM/DD/YYYY` formatting. Instead, you may presume each field, when present, will be entered according to the specification below.

Important: The script should accept all inputs *prior* to checking whether the given record exists. This is necessary to ensure that redirected input files will work correctly with your script. In other words, if you returned the user to the main menu without consuming all input fields for an operation (e.g., "create"), those fields will instead be read by the main menu, resulting in numerous "invalid menu choice" errors.

Tip: Filenames in the native Linux filesystem are case-sensitive. Thus, without proper consideration, a course whose associated filename was created as `cs3424.crs` (based on lowercase input given for the course's department code the time of its creation) will not also be associated with the file `CS3424.crs`. This is not desirable behavior, however, as user's may unexpectedly enter either uppercase or lowercase department codes when using your script.

Therefore, you may utilize a **bash** feature (known as "case modification" parameter expansion) that enables you to easily convert the contents of a variable to its uppercase equivalent. For example, this feature can be utilized to produce the capitalized contents of the variable containing a department code by simply writing `${dept_code^^}` instead of `$dept_code` when building the filename you will use for a particular course.

Tip: Note, that when prompted to enter a department code and course number, you should enter these values as *two separate tokens separated by a space*. Otherwise, it would be necessary to perform complex token-splitting to separate these values, which is outside the scope of this assignment. For example, when prompting the user:

```
Enter a department code and course number:
```

You should enter "`cs 3424`", **not** "`cs3424`"!

Storing Course Information

Course information will be stored in text files.

1. Files will be stored inside a directory called `data` within the same directory as your script.
2. Each file will be named based on the combination of a department code and a course number, which consists of **two or three** letters followed by an integer with exactly **four** digits, followed by the extension `.crs`.
3. A course file consists of *exactly* these lines:
   - dept_code (two or three letter abbreviation) dept_name (string with **probable whitespace**)
   - course_name (string with **probable whitespace**)
   - course_sched (string consisting precisely of either "MWF" or "TH") course_start (string with **no whitespace**) course_end (string with **no whitespace**) course_hours (credit hours, unsigned integer) course_size (enrolled students, unsigned integer)

   * Department names may contain whitespace. You should account for names with multiple tokens (e.g., "ESL English as a Second Language" $\implies$ dept_code = "ESL", and dept_name = "English as a Second Language")
4. Example file named `esl3053.crs`

```
ESL English as a Second Language
Literacy in a Second Language
MWF 8/20/24 12/13/24 3 52
```

Script Execution

When the script is run, the following should occur. **All script output should appear exactly as it appears below.**

1. Upon running your script, the user should be presented with the following menu:

```
Enter one of the following actions or press CTRL-D to exit.
C - create a new course record
R - read an existing course record
U - update an existing course record
D - delete an existing course record
E - update enrolled student count of existing course
T - show total course count
```

2. The user then enters a one-character action (upper or lowercase), leading to one of the following.
   - `C`: a course is created
     (a) From the terminal, read the following one at a time:
        i. Department code (two-to-three character string)
        ii. Department name (string possibly containing whitespace)

iii. Course number (integer)

iv. Course name (string possibly containing whitespace)

v. Course schedule (string $\in \{\text{MWF}, \text{TH}\}$)

vi. Course start date (string with slashes)

vii. Course end date (string with slashes)

viii. Course credit hours (unsigned integer)

ix. Initial course enrollment (unsigned integer)

(b) Using the values entered by the user, create a new file in the **`data`** folder based on the instructions above.

(c) Update **`data/queries.log`** by adding the following line:
`[date] CREATED: dept_code course_num course_name`
where *date* is the output from the **`date`** command and *dept_code*, *course_num*, and *course_name* are the corresponding values.

(d) If the course already exists, print the following error and continue with the script. The script should accept all seven inputs *before* checking if the record exists.
`ERROR: course already exists`

- `R`: read an existing course's information

(a) Prompt the user for a course department and course number: (e.g., "`cs 3424`")
`Enter a department code and course number:`

(b) Search for the specified course using the provided department and number (e.g., "`cs 3424`").

(c) Print the course information in the following format:
`Course department: dept_code dept_name`
`Course number: course_num`
`Course name: course_name`
`Scheduled days: course_sched`
`Course start: course_start`
`Course end: course_end`
`Credit hours: course_hours`
`Enrolled Students: course_size`

(d) If the course is not found, print the following error instead and continue with the script.
`ERROR: course not found`

- `U`: update an existing course record

(a) Prompt the user for the following one at a time:

i. Department code (two-to-three character string)

ii. Department name (string possibly containing whitespace)

iii. Course number (integer)

iv. Course name (string possibly containing whitespace)

v. Course meeting days (string $\in \{\text{MWF}, \text{TH}\}$)

vi. Course start date (string with slashes)

vii. Course end date (string with slashes)

viii. Course credit hours (unsigned integer)

ix. Course enrollment (unsigned integer)

(b) Search for the specified course using the course department and course number (e.g., "`cs 3424`").

(c) Update each of the corresponding fields based on the user input. **If the user input is blank for a particular field, keep the original value already present in the file.**

(d) Update **`data/queries.log`** by adding the following line:
`[date] UPDATED: dept_code course_num course_name`
where *date* is the output from the **`date`** command and *dept_code*, *course_num*, and *course_name* are the corresponding values.

(e) If the course is not found, print the following error and continue with the script. The script should accept all nine inputs *before* checking if the record exists.
`ERROR: course not found`

- `D`: delete an existing course

  (a) Prompt the user for a string representing the course department and number (e.g., "`cs 3424`"):
  `Enter a course department code and number:`

  (b) Delete the specified course's file.

  (c) Update **`data/queries.log`** by adding the following line:
  `[date] DELETED: dept_code course_num course_name`
  where *date* is the output from the **`date`** command and *dept_code*, *course_num*, and *course_name* are the corresponding values.

  (d) Print the following message to stdout with the course's number:
  `course number` `was successfully deleted.`

  (e) If the course is not found, print the following error instead and continue with the script.
  `ERROR: course not found`

- `E`: update the number of enrolled students for an existing course

(a) Prompt the user for a string representing the course department and number (e.g., "cs 3424"):
`Enter a course department code and number:`

(b) Prompt the user for an enrollment change amount (e.g., entering a value of 3 would represent enrolling three new students in the class, while -2 would reflect two students having dropped the course):
`Enter an enrollment change amount:`

(c) Search for the specified course using the course number.

(d) Update the course record by adding the new enrollment count to the course's current enrollment count. **Negative values are allowed.** (i.e., students could drop the class).

(e) Update **`data/queries.log`** by adding the following line:
`[date] ENROLLMENT: dept_code course_num course_name changed by change_amt`
where *date* is the output from the **`date`** command and *dept_code*, *course_num*, *course_name*, and *change_amt* are the corresponding values.

(f) If the course record is not found, print the following error and continue with the script. The script should accept the enrollment change amount *before* checking if the record exists.
`ERROR: course not found`

- `T`: print the total number of course records. Again, do **not** utilize external tools to perform this count (e.g., `ls`, `wc -l`, `find`, etc.).

  (a) Print the total number of **`.crs`** files within the **`data`** directory:
  `Total course records: total`
  where *total* is the total **`.crs`** files.

- If an invalid character is entered, print the following error and continue with the script.
  `ERROR: invalid option`

3. After an action is completed, display the menu again. This should go on indefinitely until `CTRL-D` or the end of a file is reached.

Assignment Data

An initial data set can be found in **`/usr/local/courses/ssilvestro/cs3424/Summer24/assign1`**. Copy this to your own assignment's directory.

Script Files

Your program should consist of seven bash files with the following names (case sensitive):

- **assign1.bash** - the main file which is initially invoked
- **create.bash** - logic for the create option
- **read.bash** - logic for the read option
- **update.bash** - logic for the update option
- **delete.bash** - logic for the delete option
- **enroll.bash** - logic for the enrollment option
- **total.bash** - logic for the total option

Verifying Your Program

Your program must work with the input provided in **a1Input.txt**. To test it:

1. Verify that your assignment folder has a **data** directory with the initial data set.
2. Execute your script and **redirect** **a1Input.txt** into it. You should not be copying or typing the contents of **a1Input.txt** into your terminal. Redirection must work, as this is how grading will be performed on your submission (i.e., your script will be fed an input file, and the resultant `data` directory will be examined for changes).
3. Verify that the output and files are as expected. Not everything in the input file is meant to be valid; it is worthwhile to check your work against erroneous input as well.

Submission

Turn your assignment in via Blackboard. Your zip file, named **a1-*abc123*.zip** where *abc123* is your myUTSA ID. The archive should contain only your seven bash files. This is very important in order to streamline the processing of submissions. Submissions that do not comply precisely with this naming scheme, or which contain extraneous files or folders, will incur a point penalty.