Part II Shell Programming - Points: 61 (13 of 25%)

Write code for an executable shell script named **`test2.sh`** that will do the following actions, in the exact order given below. Your **`test2.sh`** script must follow the script writing guidelines and four-part error message guidelines given in Notes file script_style.txt (and summarized in script_checklist.txt).

If you don't know exactly how to implement a step, part marks may be given for substitution of accurate PDL (written as shell comments) instead.

Summary and Purpose:

The first argument to the script should be a Web URL. The second, optional, argument is an integer count. The script will fetch the raw URL and analyse the count of lines in the fetched URL file against the supplied count.

Only Step Comments are Required:

Put *only* a one-line comment containing the **step number** in front of the executable code in each step (except the first step). Do not put a Step Number comment as the first line of an executable script!

Submission of test2.sh :

To submit your finished script, use this command: **`~alleni/bin/copy test2.sh`**

---

1. *[Points: 8]* Start your script with a standard CST8129 script header; but, do *not* include the Purpose or Assignment Label sections. (Remember the one-line description, syntax, path, collating, lang, and file creation mask.) The file creation mask should block group write, block other read and write, and allow everything else.
2. *[Points: 9]* If the count of the number of arguments is incorrect, issue a good four-part error message about proper input and exit the script with status 1.
3. *[Points: 1]* Put the first command line argument (the URL) into a variable named **`url`**.
4. *[Points: 2]* Put a unique temporary file name (located in directory **`/tmp`**) into a variable named **`tmp`**. The unique file name must contain the current process ID number.
5. *[Points: 9]* Fetch from the Web the raw (unformatted) URL contained in the **`url`** variable, into the file named by the **`tmp`** variable. If the fetching fails with a bad return status, issue a good error message, remove the file, and exit the script with status 2.
6. *[Points: 8]* Make sure the fetched URL file is not empty (has a file size larger than zero); otherwise, issue a good error message, remove the file, and exit the script with status 3.
7. *[Points: 3]* Remove group read permissions from the the file containing the fetched URL. Do not change any other permissions. Quick-exit the script (no message) with status 4 if changing permissions fails.
8. *[Points: 5]* Put the count of the number of lines in the fetched URL file into a variable named **`urlcount`**. Quick-exit the script (no message) with status 5 on failure.
9. *[Points: 4]* If the number of command line arguments is two, put the optional second argument (the integer count) into a variable named **`count`**; otherwise, put the value 100 into that variable.
10. *[Points: 7]* Compare the value in **`urlcount`** against the value in **`count`** and select and display one of these two messages depending on the result:

    ```
    Your URL 'XXX' contains more than YYY lines: ZZZ.
    Your URL 'XXX' contains less than YYY lines: ZZZ.
    ```

    where **`XXX`** is replaced by the URL given by the user, **`YYY`** by the value of the **`count`** variable, and **`ZZZ`** by the actual count of the number of lines in the downloaded URL file. Note that some URLs will result in no output at all. The spelling and punctuation must be as shown above.
11. *[Points: 5]* Exit the script with status zero only if the number of lines in the fetched URL file equals the user-specified count; otherwise, remove the fetched URL file and exit with status 6.

Put a one-line comment containing only the step number in front of the executable code in steps two and greater.