This assignment assumes you have read the sections of Sobell specified on the course website. You should find the notes, and the two bash script references from the notes handy as well.

You must use your rlogin account or your own installation of Linux when you analyze these questions. In all cases, your answers will be tested on the CentOS environment on rlogin.

You may work in pairs for this assignment. If you choose to work with a partner, make sure only one of you submits a solution, and that names and PIDs for both of you are listed as described below.

For question 1 type your answers in a plain text file; put the names and PIDs of both partners at the beginning of this file. For each of questions 2 and 3, you will write a single bash shell script; be sure to name your scripts exactly according to the instructions in the questions.

This is a purely individual assignment. When you have completed the assignment, you will create an uncompressed tar file containing your text file and completed script files, and nothing else. Submit your file to the Curator system by the posted deadline for this assignment. No late submissions will be accepted.

You will submit your answers to the Curator System (www.cs.vt.edu/curator) under the heading HW03.

1. Explain the output of each of the following commands. You may quote output as part of your answer, but we will grade your explanation, not the output. Experimentation is encouraged. So is reading the relevant man pages. The commands shown in the questions below may be clues to useful things for later questions (or life on Linux).

   a) [10 points] Let somefile be the name of a file in the current directory to which you have read access; explain what the output from the following command tells you:

   ```
   bash > stat -c "%x" somefile
   ```

   b) [10 points] The following command would execute the command above, but put the output from stat into a variable:

   ```
   bash > var1=`stat -c "%x" somefile`
   ```

   Explain what the output from the following, subsequent command tells you:

   ```
   bash > date -d "$var1" +%s
   ```

   c) [10 points] The double-quotes around $var1 in the command above are necessary. That is, the command will not execute correctly if you omit them. Explain why... an example is necessary to make your answer clear.

   d) [10 points] Explain the difference between the effects of the following two commands:

   ```
   bash > basename pwd
   bash > basename `pwd`
   ```

Testing a shell script is not very different from testing any program. You must create appropriate input; that's often a matter of setting up a directory structure containing a particular arrangement of files, with certain properties (e.g., permissions, file types, timestamps). Here are some suggestions:

- Make liberal use of echo statements in testing your script. The contents of a string may not be at what you expect it to be, due to the effect of special characters. You may also find it useful to experiment with the use of single- and double-quotes, both in echo statements and elsewhere.
- Be creative in designing your test cases, but don't worry about handling every kind of error a user might make when writing command-line argument (at least not for this assignment).
- Read the man page for the touch and date commands. It's very handy, especially for question 3 below, to know how to manipulate the timestamps associated with a file.
- Be sure to disable diagnostic output in the release version (i.e., the one you submit) of your script.

A shell script is a program, and therefore subject to many of the same design and documentation expectations as any program you would write for an assignment in any course. In particular:

- A script should check its command-line arguments, if it expects any. That includes not only making sure the right number of arguments are provided, but also checking for logically necessary preconditions (e.g., that a file does exist, or is readable).
- A script should be implemented using functions for actions that are likely to be useful in other scripts. (But, for this assignment, do not put those functions into an external file.)
- A script should not leave a trail of side-effects. For example, it should not alter any preexisting files unless the description of the script's functionality implies it will do so. In some cases, a script may need to create a temporary file or directory to use during its execution; if so, the script should remove those things before it exits (in all cases).
- Whenever the script terminates, after a successful execution or otherwise, the script should print useful diagnostic messages to standard output. Such messages should be concise but informative.
- The script should be documented. Internal documentation serves two purposes. Major sections of the script should be preceded by a brief comment stating what role the section plays in the operation of the script. Within a section, comments serve to explain anything subtle that a casual user of the script might not easily understand. It is not necessary to write a comment for every line in the script; trivial comments (e.g., "increment variable") actually reduce readability while providing no enlightenment.

The scripts you write for the following problems are expected to meet the expectations stated above.

2. [30 points] Write a bash script named `sdel.sh`, which provides a sort of safe deletion feature:

```
#   Synopsis:   sdel.sh FILE [DEST]
#   Description:
#       The specified file is moved to DEST (~/TRASH/ by default) and
#       compressed with bzip2.  FILE must be a regular file, and DEST
#       must already exist and be writable.
#
#   Exit codes:
#       0  successful completion
#       1  invalid number of arguments
#       2  FILE does not exist
#       3  FILE exists but is not regular
#       4  DEST does not exist, is not a directory, or is not writeable
#       5  some other error occurred
```

For example, the script might be invoked as:

```
bash > sdel.sh uselessfile.txt
bash > sdel.sh ~/mystuff/scribblings/donotwantit.pdf
bash > sdel.sh getridofit.txt ~/MyAttic
```

3. [30 points] Write a bash script named `rmstale.sh`, which removes, from a given directory, all regular files with a given extension that have not been accessed for more than a specified number of days, at the time the script is run.

```
#  Synopsis:   rmstale.sh SRCDIR EXT DAYS
#  Description:
#      Any regular file in SRCDIR that has extension EXT is removed,
#      provided it has been more than DAYS days since the file was last
#      accessed, according to the Access time as shown by stat.  SRCDIR
#      must exist and be readable, EXT cannot be empty, DAYS must be a
#      positive integer.
#
#      On successful exit, the script reports the number of regular files
#      in SRCDIR that were considered, and the number that were removed.
#
#  Exit codes:
#      0  successful completion
#      1  invalid number of arguments
#      2  SRCDIR does not exist, is not a directory, or is not readable
```

For example, the script might be invoked as:

```
bash > rmstale.sh /home/me/temp txt 3
bash > rmstale.sh ./ png 20
bash > rmstale.sh ~/TRASH gz 7
```