**Instructions:**

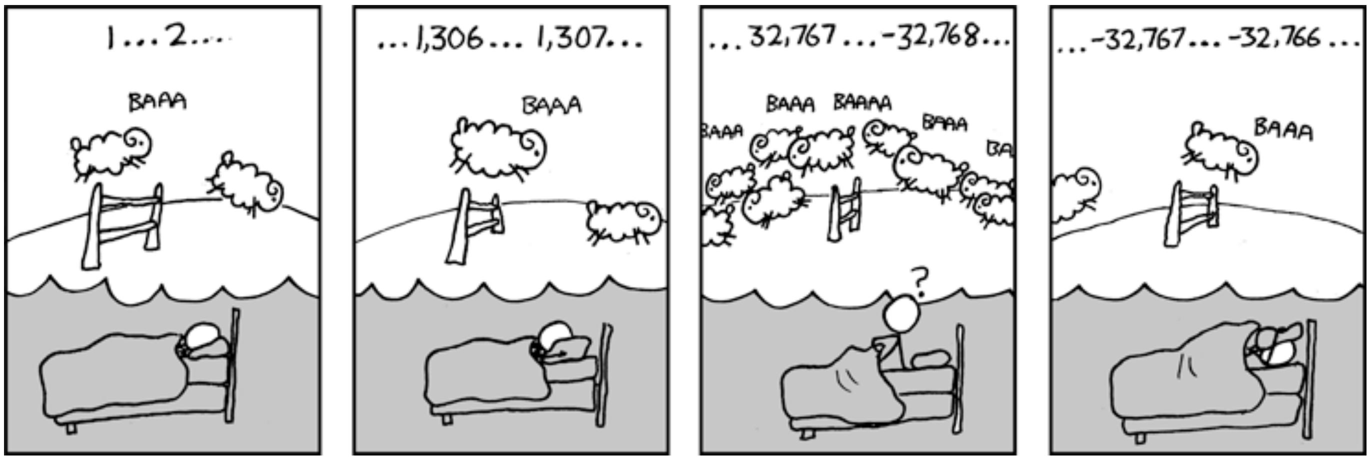
- Print your name in the space provided below.
- This examination is closed book and closed notes, aside from the permitted one-page formula sheet. No calculators or other electronic devices may be used. The use of any such device will be interpreted as an indication that you are finished with the test and your test form will be collected immediately.
- Answer each question in the space provided. If you need to continue an answer onto the back of a page, clearly indicate that and label the continuation with the question number.
- If you want partial credit, justify your answers, even when justification is not explicitly required.
- There are 6 questions, some with multiple parts, priced as marked. The maximum score is 100.
- When you have completed the test, sign the pledge at the bottom of this page and turn in the test.
- If you brought a fact sheet to the test, write your name on it and turn it in with the test.
- Note that either failing to return this test, or discussing its content with a student who has not taken it is a violation of the Honor Code.

**Do not start the test until instructed to do so!**

**Name** ______________________________________________
printed

**Pledge:** On my honor, I have neither given nor received unauthorized aid on this examination.

______________________________________________
*signed*


1... 2...
BAAA
...1,306... 1,307...
BAAA
... 32,767 ... -32,768...
BAAA
BAAA
BAAAA
BAAA
BA
?
...-32,767... -32,766...
BAAA

xkcd.com

**1.** Answer the following questions as if you were executing the commands as a non-root user on CentOS (as you would be on the rlogin cluster).

**a)** **[6 points]** If `input.txt` contains the full text of Roughing It (where the word "thing" frequently occurs), what would be the result of the executing the following commands?

```
[johokie@rlogin ~] cat input.txt | grep "thing" | tail  >> output.txt
```

**`cat` reads the contents of input.txt and pipes them to `grep`. `grep` filters out lines that don't contain "thing", then pipes the result to `tail`. `tail` discards all but the last 10 lines the came from the `grep` command. Finally, the remaining lines are appended to output.txt. At no point does anything print to the terminal.**

**b)** **[6 points]** Given the following terminal session. What would be the result of executing the following command?

```
[johokie@rlogin ~] PS1=`[\u@\W "ps"]`
```

**The intent here was to set the PS1 variable, but using the back ticks in this fashion will result in error. A reasonable attempt to answer this question gave you full credit.**

**c)** **[10 points]** Given the following terminal session. What would be the result of executing the following commands?

```
[johokie@rlogin ~] strvar="slsla"
[johokie@rlogin ~] ${strvar:2:2}
```

**The second lines creates a substring from `$strvar`. The substring is treated like a command entered by user and the shell will the try to run the result. In this case the substring is "sl", so unless you have program named "sl" on your path, an error message would be printed.**

**2.** Consider the bash script, `sum.sh`, given below. The script is syntactically correct but does not execute as its author intended.

```
#! /bin/bash
# Takes an arbitrary number of integer arguments, sums them, then prints the result.

sum=0

for x; do
        sum=$sum+$x
done
echo "The sum is: $sum"
```

**a)** **[8 points]** This bash script does not execute as expected, what does this bash shell script actually print instead of the sum? **Be specific, vague answers will receive little or no credit.**

**Instead of adding numerically, the script will treat any numbers passed at the command line like strings. Each value will be appended to $sum along with a + character.**

**For example, running ./sum.sh 1 2 3 4 would print:**

**"The sum is: 0+1+2+3+4"**

**b)** **[8 points]** What changes can you make to the script so that it works as intended (summing the integers then printing the result)? **Be specific, vague answers will receive little or no credit.**

**Use the arithmetic expansion syntax instead to add the numbers. You would need to change the expression to:**

**$((sum + x))**

**3.** Like before, answer the following questions as if you were executing the commands as a non-root user on CentOS (as you would be on the rlogin cluster).

**a) [12 points]** Create a **bash function,** `lsfoo`, that will list all of the files whose names contain `foo` and that end in a particular extension. For example, after creating the function if you type: `lsfoo txt` only files whose name contains `foo` and that have `txt` extension would be listed. The function should contain any required information or needed switches. The new command should work without additional parameters or switches.

```
lsfoo() {

        ls *foo*.$1

}
```

**b) [4 points]** Which **.bash*** file should the above function be placed in to make the command persistent?

.bashrc

**4. [14 points]** Create a **bash function**, `shgrep`, that will list all of the lines (**and line numbers**) in a bash shell script with the beginning of a `for` statement. Further, your function must test that the **file exists and is a regular file.** If the file doesn't exist or is not regular, your code should handle these cases appropriately with an error message and exit code. After creating the function you should be able to simply type `shgrep file.sh`, without additional parameters or switches, and then you should see the line numbers and lines containing the beginning of a `for` statement.

```
shgrep() {

   if [[ ! -e $1 ]]; then
          echo "File doesn't exist."
          exit 1
   fi

   if [[ ! -f $1 ]]; then
          echo "Not a regular file."
          exit 2

   fi

   grep -n "for" $1
   exit 0

}
```

**5.** Consider the bash script, `runme.sh`, given below. The script is syntactically correct and executes as its author intended.

```
#! /bin/bash
############################################################# processSubdir
                   # $1: fully-qualified name of a directory
                   # $2: fully-qualified name of a directory below $1
processSubdir() {

   for y in $2/*
   do
      if [[ -f $y ]]; then     # check for "regular" file
         fname=$y
         fname=${fname##*/}
         if [[ ! -e $1/$fname ]]; then
            mv $y $1
         fi
      elif [[ -d $y ]]; then   # check for directory
         processSubdir $1 $y   # recursive call
      else
         rm –f $y
      fi
   done
}

############################################################# prelim. stuff
if [[ $# -ne 1 ]]; then
   echo "Invocation:  runme.sh DIR"
   exit 1
fi

############################################################# main body
rootdir="$1"
for x in $rootdir/*
do
   # helpful comment omitted
   if [[ -d $x ]]; then
      # helpful comment omitted
      processSubdir $rootdir $x
      # -rf removes even a nonempty directory
      rm -rf "$x"
   fi
done
exit 0
```

Suppose `ls` command (run recursively) shows the following information:

```
[Centos65 ~]$ ls -lR Q5
Q5:
total 4
-rw-rw-r--. 1 joehokie joehokie 1024 Sep 21 22:06 f0.txt
drwxrwxr-x. 3 joehokie joehokie 4096 Sep 21 22:06 sub1

Q5/sub1:
total 4
-rw-rw-r--. 1 joehokie joehokie 1024 Sep 21 22:06 f1.txt
drwxrwxr-x. 2 joehokie joehokie 4096 Sep 21 22:06 sub2

Q5/sub1/sub2:
total 0
-rw-rw-r--. 1 joehokie joehokie 1024 Sep 21 22:06 f2.txt
```

**a)** **[10 points]** Assuming the script is in the user's path, and the user (`johokie`) has appropriate access to the directories shown above, what would the same `ls` command show after the user executed: `runme.sh ~/Q5`

```
[Centos65 ~]$ ls -lR Q5
Q5:
-rw-rw-r--. 1 joehokie joehokie 1024 Sep 21 22:06 f0.txt
-rw-rw-r--. 1 joehokie joehokie 1024 Sep 21 22:06 f1.txt
-rw-rw-r--. 1 joehokie joehokie 1024 Sep 21 22:06 f2.txt
```

The script traverses the subdirectories below the original directory (~/Q5 in this case), moving any regular files there into the original directory, and removing the subdirectories.

**b)** **[6 points]** In the second section of the script (labeled `prelim. stuff`) the author of the script correctly checks to be sure the script was invoked with the correct number of parameters. In order to guarantee the script will execute without problems, what else should the author of the script check in that section? (No code is necessary, just describe things logically.)

The script should also verify the parameter has the correct properties; that is, it should verify that:

- $1 is the name of a directory
- the user has read permissions for that directory
- the user has write permissions for that directory
- the user has execute permissions for that directory

(Aside from that, the script should check the same thing for each subdirectory detected as the for loop is executed, but that's not relevant to this question.)

6. **[16 points]** Recall that for an earlier assignment you wrote a bash script, `howmany.sh`, that could be used to count the number of files in a given directory, optionally matching a given file extension. For this assignment, you will write a script to do something similar for a collection of tar files.

Write a bash script, `hmintar.sh`, that takes a directory and a file extension as parameters, and reports for each tar file in the given directory the number of regular files in that tar that match the given extension. For example, the command

```
hmintar.sh /home/data txt
```

would report the number of regular files in each tar in `/home/data` that had the extension `txt`. More specifically, suppose the directory `/home/data` contained the following files:

```
OldNotes.tar
2009Archive.tar
ScriptArchive.tar
xkcd_20130513.jpg
```

Then the command above might would generate output like this, echoing the name of each tar file and the number of matching files in that tar, and no output for non-tarfiles:

```
OldNotes.tar        78
2009Archive.tar    1352
ScriptArchive.tar   0
```

Don't worry about alignment issues with your output. The script should verify the command-line parameters are valid, but you don't need to produce useful error messages.

There are several ways to solve this problem with a bash script. You are encouraged, but not required, to have your solution call the script `howmany.sh` in an appropriate manner. You may assume `howmany.sh` is in a directory that's in the user's path.

Be careful of your logic, and be as exact with syntax as possible. Write comments to explain your intent; that will make it easier for us to evaluate your answer if you have syntax errors.

**The following page has been left blank so you can continue your answer to this question.**
**Do not cram an illegible solution into the space below.**
**Start here, with your analysis, and write your complete answer on page 9.**

The basic idea is simple enough, scan the given directory for tar files and examine the contents of each tar file. How to iterate through the directory is obvious enough, from the script in Q5.

To examine the contents of each tar file, we can extract the contents to a directory and then scan that directory (with howmany.sh).

In my solution, I create a temporary directory, unpack a tar file into that directory, scan with howmany.sh and report the results, and them clean up the temporary directory.

I put the temporary directory in /tmp, which is a root-level directory provided so user programs can do exactly this. (You were not expected to know that.)

```
#! /bin/bash
############################################## fn to check for tar file
#                  param1:  name of file to be checked
isTar() {

   mimeType=`file -b --mime-type $1`
   [[ $mimeType == "application/x-tar" ]]
}

################################################################
if [[ $# -ne 1 ]]; then
   echo "Invocation:  q6b.sh DIR EXT"
   exit 1
fi

if [[ ! -d $1 ]]; then
   echo "$1 must be a directory"
   exit 2
fi

if [[ ! -r $1 ]] || [[ ! -x $1 ]]; then
   echo "Must have read/write/execute permissions for $1"
   exit 2
fi

################################################################

rootDir="$1"                   # directory being scanned
extension="$2"                 # file extension to match

tmpdir="/tmp/tmpextractdir"    # extraction directory

for x in $rootDir/*            # iterate through rootDir
do
   isTar $x                    # see if have a tar file
   if [[ $? -eq 0 ]]; then     # if so
      mkdir $tmpdir            #    create extraction dir
      tar xf $x -C $tmpdir     #    extract tar file to it
      fname=${x##*/}           #    strip path info
                               # use howmany.sh to count files
      echo "$fname:  `howmany.sh $tmpdir $extension`"

      rm -Rf $tmpdir           # remove temp directory
   fi
done

exit 0
```