Instructions:

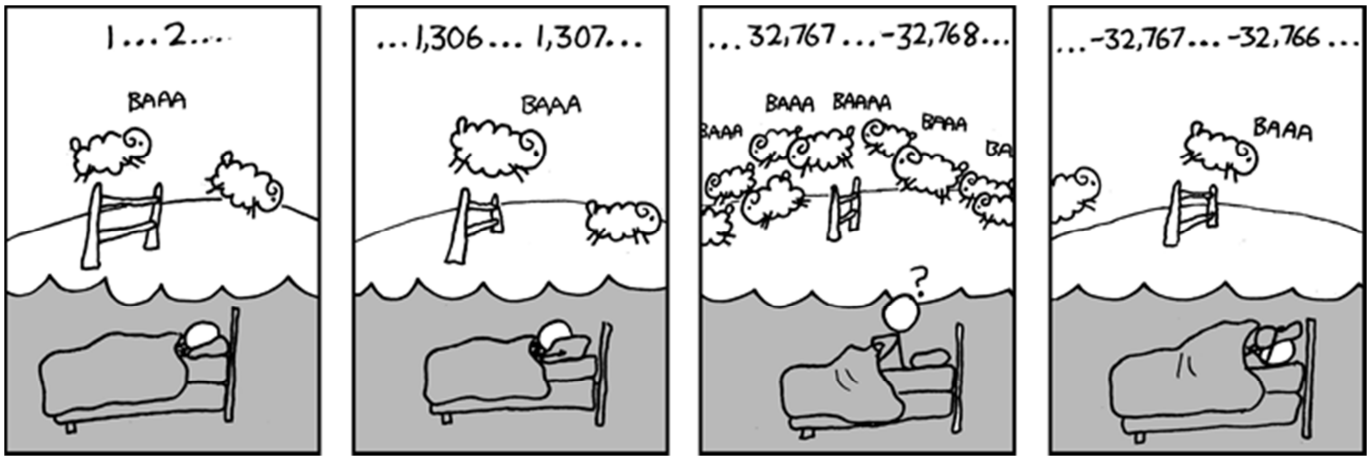
- Print your name in the space provided below.
- This examination is closed book and closed notes, aside from the permitted one-page formula sheet. No calculators or other computing devices may be used. The use of any such device will be interpreted as an indication that you are finished with the test 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 5 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 Solution
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 BAAAA
BAAA
BAAA
?
...-32,767... -32,766...
BAAA

xkcd.com

1. Answer the following questions as if you were executing the commands on the rlogin cluster.

a) **[8 points]** What command will add the current directory to your **path**? Which **.bash*** file should the command be placed in to make the change persistent?

This line will add the current directory to a user's path:

```
PATH=$PATH:.
```

You should put the command in `.bash_profile`

b) **[6 points]** Create an **alias**, `rmr`, that will recursively remove a directory. The alias should contain any required information and needed switches. After creating the alias, to remove a directory you should be able to simply type `rmr dir_to_rm`, without additional parameters or switches.

Something like this will work: `alias rmr="rm -r"`

Note: if you didn't remember the `-r` switch for `cp`, you could also write a script function to do this, and alias it to `cpr`. The function given in question 5 is somewhat similar to this.

2. **[6 points]** Create an **alias**, `sshrlog`, that will let you `ssh` into rlogin from another Linux installation. The alias should contain any login info and other needed switches. Further, your alias **should allow** graphical programs to be started from the command line. After creating the alias you should be able to simply type `sshrlog`, without additional parameters or switches, and then you should be prompted for a password (assuming haven't set up password-less login).

Something like this will work:

```
alias sshrlog="ssh -l hmonti -X rlogin.cs.vt.edu"
alias sshrlog="ssh -X hmonti@rlogin.cs.vt.edu"
```

3. Like before, answer the following questions as if you were executing the commands on the rlogin cluster.

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

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

'cat input.txt' will output the file to stdout, the pipe ('|') will capture that output (and not display it), and then send it to the grep "Captain" command. The grep command will match all and only the lines that contain "Captain", and finally those lines will be written to output.txt using redirection ('>').

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

```
[johokie@rlogin ~] file test.jpg
test.jpg: JPEG image data, JFIF standard 1.01
[johokie@rlogin ~] mv test.jpg test.pdf
[johokie@rlogin ~] file test.pdf
```

Despite renaming the file to test.pdf, the file command actually examines the file's contents and is able to determine that test.pdf is a JPEG document. So it will print out:

test.jpg: JPEG image data, JFIF standard 1.01

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

```
[johokie@rlogin ~] size=`stat -c "%s" test.tar`
[johokie@rlogin ~] dd if=/dev/zero of=test.tar bs=1 count=$size
```

The first line runs the command stat -c "%s" test.tar (note the back ticks), and stores the output of the command in the shell variable size. This invocation of stat command will tell you how big a file is in bytes, so size is the number of bytes in the file.

The next line uses dd to overwrite the file with zeros. Using the parameters bs=1 (so 1 byte) and count=$size (number of bytes in the file) we overwrite the entire file byte by byte. Further, the command uses $size to overwrite the exact number of bytes in the file.

As a result these lines end up acting like the shred command, or safely "deleting" a file's contents.

4. Like before, answer the following questions as if you were executing the commands on the rlogin cluster.

Examine the following terminal session; you may assume these commands are run from the user's home directory.

```
[johokie@rlogin ~] ls  /bin/
bbcp cp ls sh

[johokie@rlogin ~] ls /usr/bin/
bbcp curl cvs diff gnuplot

[johokie@rlogin ~] ls  ~/bin
bbcp  gnuplot  python

[johokie@rlogin ~] ls
bbcp
```

a) **[7 points]** Assume the `$PATH` variable contains the value show below. If the user were to run the command `gnuplot` which executable will run? **Specify the absolute path to the executable (e.g., /foo/bar/gnuplot).**

```
[johokie@rlogin] echo $PATH

/home/johokie/bin:/usr/kerberos/bin:/usr/bin::/usr/local/bin:/bin:.
```

This is the command that will run: `/home/johokie/bin/gnuplot`

b) **[7 points]** Assume the `$PATH` variable contains the value show below. If the user were to run the command `bbcp` which executable will run? **Specify the absolute path to the executable.**

```
[johokie@rlogin] echo $PATH

.:/home/johokie/bin:/usr/kerberos/bin:/usr/local/bin:/bin:/usr/bin
```

Since the working directory is at the beginning of the `PATH` and we are told to assume we are in the users home directory, this command will run:

`/home/johokie/bbcp`

c) **[7 points]** Given your answers above, how does `bash` choose which executable will run when a command is entered?

`bash` searches the colon-separated list of directories from left to right in order, and chooses the first executable it encounters. So order is important. This is why slides recommend that you add the current directory at the end of the path, so system executables are found first.

d) **[5 points]** Suppose you manage to severely mess up your `.bashrc` and/or `.bash_profile` so that your PATH variable is broken and you can't run standard commands on rlogin. How can you recover from this situation?

There are a few way you could fix this:

Manually recreate you path using `PATH=$PATH…` syntax, then use `vi/nano/gedit` to fix the configuration files.

If you've got a working `.bashrc`/`.bash_profile` on your own Linux install, you can transfer them to `rlogin` by using `scp`.

Use absolute paths eg `/bin/vi`, to execute a text editor and fix the files.

Use absolute paths e.g. `/bin/cp` to copy the default system `/etc/profile` or `/etc/bashrc`

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

```
#! /bin/bash
################################################## fn to process directory
process() {
   for fname in $1/*; do

      if [[ -f $fname ]] && [[ -O $fname ]]; then
         echo "$fname"
      elif [[ -d $fname ]] && [[ -r $fname ]]; then
         process $fname
      fi
   done
}
################################################### check arguments
if [[ $# -ne 1 ]]; then
   echo "Please specify a directory"
   exit 1
fi
if [[ ! -d $1 ]]; then
   echo "$1 is not a directory"
   exit 2
fi
if [[ ! -r $1 ]]; then
   echo "I don't have read permission for $1"
   exit 3
fi
################################################### process directory
process $1

exit 0
```

Suppose the current directory contains a subdirectory `Cprogs`, and that the following files occur in `Cprogs` and its subdirectories. The current user, `johokie`, has read access to all the directories except for `Cprogs/Compare`.

```
Cprogs:
-rw-rw-r--. 1 johokie users1  379 Feb 20 20:18 readme

Cprogs/Compare:
total 8
-rw-rw-r--. 1 johokie users1 4869 Nov 20 22:11 compare.c

Cprogs/ESpaced:
total 20
-rwxrwxr-x. 1 root    root    7431 Jan 30 23:04 driver
-rw-rw-r--. 1 johokie users1   384 Jan 30 23:03 driver.c
-rw-rw-r--. 1 johokie users1   584 Jan 30 23:04 EvenlySpaced.c
-rw-rw-r--. 1 johokie users1    95 Jan 30 22:48 EvenlySpaced.h

Cprogs/RemEven:
total 24
-rwxrwxr-x. 1 johokie users1  7076 Feb  2 21:14 driver
-rw-rw-r--. 1 vthoky  users1   412 Feb  2 21:14 driver.c
-rw-rw-r--. 1 vthoky  users1  1808 Feb  2 20:53 driver.o
-rw-rw-r--. 1 johokie users1   612 Feb  2 20:54 RemoveEvenDigits.c
-rw-rw-r--. 1 johokie users1   125 Feb  2 20:42 RemoveEvenDigits.h
```

a) **[4 points]** What will be written if user `johokie` invokes the script as: `./showem.sh` ?

```
"Please specify a directory"
```

The check for the number of parameters being 1 will trigger and this output will be written, after which the script will exit.

b) **[6 points]** What will be written if user `johokie` invokes the script as: `./showem.sh Cprogs/ESpaced` ?

```
Cprogs/ESpaced/driver.c
Cprogs/ESpaced/EvenlySpaced.c
Cprogs/ESpaced/EvenlySpaced.h
```

The script will process the list of files in `Cprogs/Espaced`, and print the names of those for which the user (`johokie`) is owner (`"-O"`).

c) **[8 points]** Suppose `johokie` invokes the script as: `./showem.sh Cprogs`. Then the output from the script will include the following (and possibly more):

```
Cprogs/readme
Cprogs/RemEven/driver
Cprogs/RemEven/RemoveEvenDigits.c
Cprogs/RemEven/RemoveEvenDigits.h
```

So, the script must actually process the subdirectories of `Cprog`. Why does that happen? That is, what is it about the script that makes it process subdirectories? Be very specific.

Within the body of the function process(), we have the following if/elif statement:

```
. . .
   if [[ -f $fname ]] && [[ -O $fname ]]; then
       echo "$fname"
   elif [[ -d $fname ]] && [[ -r $fname ]]; then
       process $fname
   fi
. . .
```

If `fname` names a readable directory, `process()` calls itself, so it processes subdirectories recursively.

6. **[14 points]** Write a bash script, `promote.sh`, that takes a child directory as a parameter, moves as many regular files in that directory into its parent directory as is possible, then deletes the given child directory if it's now empty. For simplicity, you may overwrite files in the parent directory that have the same name as a file in the subdirectory, and you may assume the child directory contains only regular files. For example, the command

```
./promote.sh /home/data/old
```

would move as many files as possible from `/home/data/old` into `/home/data` and remove `/home/data/old`, if that directory was now empty.

The script should verify the parameter is a readable directory, that it has a parent directory, and that the parent directory is writeable. If not, the script should exit with an informative message and exit code.

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 and continue onto page 9.

One solution is shown on the following page. The essential steps the script must perform are:

- verify the script was invoked with one parameter
- decompose that parameter to get the name of the parent and subdirectories
- verify the parent directory exists and is writeable
- verify the subdirectory exists and is readable
- for each file in the subdirectory
  - move it into the parent directory (watch for files that can't be moved)
- if the subdirectory is empty, remove it

```
#! /bin/bash
#  Invoke as:  promote.sh DIR

############################################################## verify dir
chk_directory() {
   if [[ ! -d "$1" ]] || [[ ! -r "$1" ]]; then
      echo "$1 is not a readable directory."
      exit 2
   fi
   if [[ "$1" == "/" ]]; then
      echo "$1 is the root directory."
      exit 3
   fi
   PARENT=$1
   PARENT=${PARENT%/*}
   if [[ ! -w "$PARENT" ]]; then
      echo "The parent of $1, $PARENT, is not writeable."
      exit 4
   else
      echo "parent: $PARENT"
   fi
}

# Verify script was invoked with 1 parameter, that is the name of a
# readable directory.  Exit with an appropriate message if that is
# not true.
if [[ $# -ne 1 ]]; then
   echo "Please specify a directory."
   exit 1
fi

SUBDIR=$1
SUBDIR=${SUBDIR%/}

chk_directory $SUBDIR

# If SUBDIR is not the root directory, copy all the files from SUBDIR into
# its parent directory, and then remove SUBDIR.  You may assume there is no
# file in the parent that has the same name as a file in DIR.
OK=0
for element in $SUBDIR/*; do
   mv -f "$element" "$PARENT"
   if [[ ! $? -eq 0 ]]; then
      echo "Could not move $element from $SUBDIR to parent."
      OK=$((OK + 1))
   fi
done

if [[ OK -eq 0 ]]; then
   rmdir $SUBDIR
fi

exit 0
```