CSC 209 Assignment 1, Summer 2019: Shell scripts

Due by the end of Friday May 31, 2019; no late assignments without written explanation.

Please re-read the statement about academic offences at the end of the course information sheet!

Part 1: The *animal* guessing game

```
$ sh animal db
Think of an animal.
Is your animal green?
y
Is your animal a frog?
n
Aw, I lose.  What animal were you thinking of?
tree
Tell me a question which would distinguish a frog from a tree.
Does it have leaves?
What would be the answer for a tree?
y
I'll remember that.
$ sh animal db
Think of an animal.
Is your animal green?
y
Does it have leaves?
n
Is your animal a frog?
y
I got it!
$
```

In this sample interaction, where '$' is the shell prompt, the user typed the bits in bold face: namely, all of the "y" and "n" lines; the line saying "tree"; and the first "Does it have leaves?" but not the second.

For part one of this assignment you will write the above program as an *sh* shell script.

The unix filesystem is used to store all of the data for the program. Each question is represented by a directory. Each directory contains either

• a file named `name` stating the animal at that leaf node (e.g. "frog")

or

• a file named `question` containing a yes/no question to ask (including the question mark if applicable), ***and*** subdirectories `yes` and `no` which are further nodes to be consulted in the case of the respective answers.

As in the example above, your shell scripts will all take the starting directory name as $1 (output a usage message if $# is not 1). As animals are added, it modifies this directory.

Please see two distributed example databases in /u/csc209h/summer/pub/a1: a minimal initial database in db.tar and a slightly larger database in largerdb.tar. Each of these tar files extracts into a subdirectory named with the file's base name, e.g. a subdirectory named largerdb. You can extract all files from a tar file into the current directory (and below) with "`tar xvf file`". Note that you probably have to rm –r the directory before re-extracting if you want to start over.

Also please see the course web page for questions and answers, and other information.

Your task is to write this *animal* script. (It will not use *find*.)
For full marks, your program must obey the sample dialogue exactly, with no extra spaces or newlines.

(continued)

In particular, please try the test

```
sh animal db </u/csc209h/summer/pub/a1/yn.in | diff - /u/csc209h/summer/pub/a1/yn.out
```

which should produce no output (meaning that there are no differences).

(I think that if you are in control of your program you should be able to perform this precise matching; and *diff*s like the above will allow you to check it.)

For the yes/no questions, your program should ideally accept any of "yes", "y", "no", or "n". Our test programs will only use "y" and "n".

Hint: Don't just use "read var" by itself to read input, because you need to detect eof. Put it in an 'if' statement to check its exit status.

Hint 2: To invert a test which doesn't use "test" (and thus you can't simply use test's '!' feature), put your if-body in the else clause. Example:

```
if read var
then
    :
else
    exit 0
fi
```

In cases as simple as the above, you can shorten this to

```
read var || exit 0
```

but please be sure you understand this syntax before using it.

Part 2: Euclid's greatest common divisor algorithm

Your second task is to write an *sh* shell script that uses Euclid's algorithm to calculate the greatest common divisor of any number of non-negative integers, which will be supplied to your script either from files or from standard input.

The algorithm to calculate the GCD of two non-negative integers x and y is:

```
while (y != 0) {
    t = x
    x = y
    y = t % y
}
print x
```

(The "expr" command has a '%' operator.)

You will write a file-processing version of this command. Arguments to your shell script are file names; if there are no arguments, it should process the standard input. All of the contents of all of the concatenated files other than white space are expected to be non-negative integers, and you do not have to check this for this assignment. However, you do need to be able to accept numbers separated by spaces or by newlines, and blank lines, and so on.

Your script outputs a single number which is the greatest common divisor of all of the input numbers. You can calculate this pairwise; that is, gcd(a,b,c) = gcd(gcd(a,b),c), and so on for any number of inputs. The gcd of just one number is that number, and the gcd of no numbers at all is 0, because that is the identity for gcd (i.e. for all positive integers x, gcd(x,0)=x).

(continued)

Note that you can use *cat* to do all of your argument processing by saying

```
cat "$@"
```

But if this is piped into an sh loop construct, please read the note about a tricky issue with pipes and variables in the Q&A web page.

To submit

I suggest you begin by making a new directory to hold your assignment files, plus any other working copies and associated files. You should call your assignment files animal and gcd.

Once you are satisfied with your files, you can submit them for grading with the command

```
submit -c csc209h -a a1 animal gcd
```

You may still change your files and resubmit them any time up to the due time. You can check that your assignment has been submitted with the command

```
submit -l -c csc209h -a a1
```

You can also submit files individually instead of all at once, and you can resubmit a particular file by using the –f option, e.g.

```
submit -c csc209h -a a1 -f animal
```

This is the only submission method; you do not turn in any paper.

Other notes

Your programs must run on the teaching lab linux machines, and they should use standard *sh* features only (avoiding *bash* extensions). Please test them in different shells, not only in *bash*. E.g. also try "dash animal db".

For now, we will run our shell scripts by typing "sh file" or "sh file args ...".

We will not be worrying about the "#!/bin/sh" thing for assignment one.

Do not use python, awk, perl, or any other programming language in your scripts besides *sh*. (These are valuable languages to know, but are not the point of the current assignment, in which you will be graded on your *sh* programming.)

Please see the assignment Q&A web page at

```
https://www.teach.cs.toronto.edu/~ajr/209/a1/qna.html
```

for other reminders, advice, and answers to common questions.

Remember:

This assignment is due at the end of Friday, May 31, by midnight. Late assignments are not ordinarily accepted and *always* require a written explanation. If you are not finished your assignment by the submission deadline, you should just submit what you have (for partial marks).

Despite the above, I'd like to be clear that if there *is* a legitimate reason for lateness, please do submit your assignment late and send me that written explanation. (For medical notes, I need to see the original, in person. E.g. in office hours or after the next class.)

I'd also like to point out that even a zero out of 7% is far better than cheating and suffering an academic penalty. Don't cheat even if you're under pressure. Whatever the penalty eventually applied for cheating, it will be worse than merely a zero on the assignment. Do not look at other students' assignments, and do not show your assignment (complete or partial) to other students. Collaborate with other students only on material which is not being submitted for course credit.