Assignment Solution 2: Shell Programming
COMPSCI 215

Due: 3pm, Wednesday 23 August, 2006

UNIX [100 marks]

- You can use any Unix system you wish (e.g. Knoppix, Linux). However, your answers must be correct when run on the department Unix box (chaos). (This should only be an issue if you attempt to use fancy features of your own Unix environment.)
- Submit your answers to the questions electronically.
- This assignment is worth 5% of your final grade.

Q1. [10 marks]

A shell script is a text file containing shell commands. It must start with a line like `#!/bin/bash`, which names the shell interpreter to use.

Write a shell script, `guess.sh` that implements a simple number guessing game. When run, the shell script prompts the user to guess a random number between 0 and 100. If the user guesses incorrectly, the script prompts the user "Guess higher" or "Guess lower" as appropriate and allows the user to guess again. If the user guesses correctly, the script terminates printing out how many guesses the user needed.

For example:

```
chaos$ ./guess.sh
Guess my secret number (0-100): 101
Thats not between 0 and 100
Guess my secret number (0-100): 80
Guess higher
Guess my secret number (0-100): 85
Guess lower
Guess my secret number (0-100): 82
Right!  You guessed my secret in 3 guesses.
chaos$
```

Remember, before you run the script, you must make it executable by typing:

```
chaos$ chmod +x guess.sh
```

You may also wish to read about the shell variable `$RANDOM` in `man bash`.

Answer: We are looking for good comments and variable names. A few points of note:

1. **Picking a random number** The assignment suggests using $RANDOM. This is probably the simplest way to get a random number but a program may use other means. For example, getting values from `/dev/random` in some way (clumsy), using $$ (not so random) or using poorly predictable eg `` `date +%N` ``.
2. **Normalizing the random value.** We need to a random number between 0 and 100. The easy way to do this is by finding the remainder after we divide by 100. It is also possible to get values between 0 and 99 inclusive as opposed to 0 and 100, using say `echo $RANDOM | rev | cut -c1-2` to get two characters. But this is much more clumsy and error prone (eg. because of potentially leading zeros) and probably should get less marks.
3. Marks for:
   - Checking the correct range of input
   - Counting guesses correctly
   - Comments and style
   - Generating random value correctly
   - Prompting the user for greater, less or right answers

```
#!/bin/bash
#  Author           : Jasvir Nagra <jas@cs.auckland.ac.nz>
#  Created On       : Sat Aug 26 14:01:16 2006
#  Description      : Plays guessing game with numbers between 0 and 100

secret=$((RANDOM%100))
user_guesses=0

while true; do
  echo -n "Guess my secret number (0-100): "
  read guess
  ((user_guesses++))

  # Sanitizing input
  if [ -z "$guess" ]; then
      echo "Thats not even a guess!"
      continue
  elif echo "$guess" | grep "[^0-9]" >&/dev/null; then
      echo "Thats not a number!"
      continue
  elif [ $guess -lt 0 -o $guess -gt 100 ]; then
      echo "Thats not between 0 and 100"
      continue
  fi

  # Playing the game
  if [ $guess -lt $secret ]; then
      echo "Guess higher"
  elif [ $guess -gt $secret ]; then
      echo "Guess lower"
  else
      echo "Right! You guessed my secret in $user_guesses guesses."
      exit 0
  fi
done
```

Q2. [20 marks]
Write a bash script, `lower.sh` to "lowercase" a directory of files. If a lowercase file of the same name already exists in that directory, the program should warn the user and NOT overwrite the existing file.

E.g.

```
chaos$ ls dir/
One-File    TWO-File    Three-FILE    one-file
chaos$ ./lower.sh dir/
Warning: Not overwriting one-file
chaos$ ls dir
One-File    one-file    three-file  two-file
```

Your script may benefit from using the `for` shell command. The directory name passed to the command is available in the variable `$1`. Finally, the magic backquote (`) wants to come into it somewhere. You can assume there are no subdirectories in the given directory.

Answer: Compared to Q3, this is considerably straight forward and ought to be marked less leniently than Q3. The solution need *not* handle filenames with spaces or files that being with dashes to get full marks.

Marks for:

- Correcting determining the new name.
- Looping over all files in the correct directory
- Doing the right thing if a file is already lowercased or if there is a name collision.
- Comments and style

```
#!/bin/bash
#  lower.sh --
#  Author          : Jasvir Nagra <jas@cs.auckland.ac.nz>
#  Created On      : Sat Aug 26 14:11:34 2006
#  Description     : Converts filenames in a directory to lowercase

for oldname in $1/*; do
    newname=`echo $oldname | tr [:upper:] [:lower:]`
    if [ ! "$oldname" = "$newname" ]; then
        if [ ! -e "$newname" ]; then
            mv -- "$oldname" "$newname"
        else
            echo Warning: Not overwriting $newname
        fi
    fi
done
```

Q3. [30 marks]
Extend the previous shell script to create `superlower.sh` so that instead of merely issuing a warning, it appends a numbered extension to make the filename unique.

E.g.

```
chaos$ ls dir
One-File    TWO-File    Two-File    Three-FILE    ONE-FILE    ONE-File
chaos$ ./superlower.sh dir/
chaos$ ls dir
one-file    one-file.1    one-file.2    three-file    two-file    two-file.1
```

Answer: In addition to meeting the requirements for Q2, marks for:

- remembering the lower-case file name in a shell variable, rather than calling `tr` repeatedly.
- testing for the existence of the file before calling `mv`.
- do nothing on lowercase files/directories . In particular, *not* adding new numbered extensions.

```
#!/bin/bash
#  superlower.sh --
#  Author          : Jasvir Nagra <jas@cs.auckland.ac.nz>
#  Created On      : Sat Aug 26 14:33:14 2006
#  Description     : Converts filenames in a directory to lowercase,
#                    appending a numbered extension if necessary to
#                    make the new filename unique.

curr=1;
currfile="NULL"

for oldname in $1/*; do
    newname=`echo $oldname | tr [:upper:] [:lower:]`
    if [ ! "$oldname" = "$newname" ]; then
        if [ ! -e "$newname" ]; then
            mv -- "$oldname" "$newname"
        else
            if [ "$currfile" != "$newname" ]; then
                currfile=$newname
                curr=1
            fi

            # Recompute a new file name rather than storing it
            while [ -e "$newname.$curr" ]; do
                ((curr++))
            done
            mv -- "$oldname" "$newname.$curr"
            ((curr++))
        fi
    fi
done
```

Note: I recompute filename extensions. With a lot of files with the same name (but different case), my solution is a little inefficient. Its possible to do a quicker (but complicated) solution using arrays (not covered in lectures) or by emulating arrays using UNIX commands we have covered eg. `cut`. This is a very interesting exercise if you're keen!

Q4. [40 marks]
Write a shell script called `viztree.sh`, which recursively displays a visual version of a directory like this:

```
$ ./viztree.sh mydir
mydir
|-- alarm
|-- myfile.txt
|-- fileslist.txt
|-- jas
    |-- a.txt
    |-- docs
        |-- memo
        |   |-- shopping-list.txt
        |-- zip
```

Your shell script should accept one argument on the command line, a directory name. If a user does not pass any arguments or passes the argument "`--help`", your script should display help about the usage of the script. Your script should make good use of comments, functions, loops and well named variables to make the script easy to read.

For this question, you can ignore symbolic links.

Answer:

A good solution to this question requires some non-computer time thought. Its possible to get a solution simply by repeatedly bash-ing your way through, but the end result will not be pretty. More importantly it will be long and not particularly readable or maintainable.

The solution uses *recursion*. Recursion caused a lot of confusion on the forums, however, there is nothing magical about recursive functions... they are simply functions that call themselves. Of course, this can result in an infinite number of function calls unless we are careful to ensure that, soon or later, the function will simply return a value rather than calling itself again. This case is called the base case and when we develop our solution, we want to ensure every recursive call is "closer" to the base case, otherwise the recursion will never terminate.

When looking for a solution to this problem, notice how the initial part of each line (with the bars and spaces) is exactly the same as the previous line. Each time you descend down a directory, all that changes is the initial part of each line - it becomes whatever it was before plus a bar and three spaces. In fact, the only difficulty that arises is because the descending bars stop at the last file in each directory. In this case, its the same as what it was before plus four spaces. This means the solution has two parts, descending when you're handling the last file in a directory, and descending when you're handling any other file!

That's really it!

For marks:

- Displays usage
- Comments and style
- Indents correctly
- Draws the bars correctly
- Handles the case where no more bars should be drawn after the last file in a directory.

```
#!/bin/bash
#  viztree.sh --
#  Author          : Jasvir Nagra <jas@cs.auckland.ac.nz>
#  Created On      : Sun Aug 27 01:37:55 2006
#  Description     : Displays a visual representation of a directory tree

# Takes a directory and an indentation and recursively displays every
# file in that directory but starts each line with "indentation"
displayDir() {

  local indent=$1
  local nthFile=0
  local totalFiles=`ls | wc -l`

  for file in *; do

  # We count the files as we display them...
  (( nthFile++ ))

  echo "$indent|--␣$file"
  if [ -d "$file" -a "$file" != "." ]; then
    cd $file

  # ...because we need to handle the last entry specially
    if [ "$nthFile" -eq "$totalFiles" ]; then
      displayDir "$indent␣␣␣␣"
    else
      displayDir "$indent|␣␣␣"
    fi
    cd ..
  fi
done
}

# Displays usage information then quits
usage () {
  echo "Usage:␣viztree.sh␣[--help]␣DIR"
  echo "Displays␣a␣visual␣tree␣representation␣of␣DIR."
  exit 1
}

# Main program
[ $# -eq 0 -o "$1" = "--help" ] && usage
cd $1                           || usage
echo $1
displayDir ""
```