Shell Scripts

1. Cletus has an extreme phobia of two periods ('..') appearing consecutively anywhere, stemming from an early childhood trauma too embarrassing to relate. He decided to write a shell script 'cd_up.sh' aimed at changing the working directory to the level up, so he didn't have to go through the distress of regularly seeing '..':

   ```
   #!/bin/bash
   cd ..
   ```

   But when it ran his script, it didn't seem to work:

   ```
   $ pwd
   $ ./cd_up.sh
   $ pwd
   ```

   Why not, and can you fix Cletus' shell script?

   A shell script is executed by a separate shell so changes to its working directory don't affect its parent shell. Similarly the shell script 'set_var.sh' contains this:

   ```
   #!/bin/sh
   variable=42
   ```

   It won't change the variable in the parent shell, for example:

   ```
   $ variable=21
   $ ./set_var.sh
   $ echo $variable
   21
   ```

   You can indicate that the commands in a shell script are to be run by the parent shell, like this:

   ```
   $ variable=21
   $ . ./set_var.sh
   $ echo $variable
   42
   ```

2. The course code for COMP2041 has been changed to COMP2042 and the course code for COMP9041 has been changed to COMP9042. Write a shell script, `update_course_code.sh` which appropriately changes the course_code in all the files it is given as argument.

   Sample solution for update_course_code.sh

   ```
   #!/bin/sh

   for file in "$@"
   do
       temporary_file="$file.tmp.$$"
       if test -e "$temporary_file"
       then
           echo "$temporary_file" already exists
           exit 1
       fi
       sed 's/COMP2041/COMP2042/g;s/COMP9041/COMP9042/g' $file >$temporary_file &&
       mv $temporary_file $file
   done
   ```

   Alternatively a single line Perl solution:

   ```
   $ perl -pi -e 's/(COMP[29]04)1/${1}2/ig' file1 file2 ...
   ```

3. Modify `update_course_code.sh` so if given a directory as argument it updates the course codes in files found in that directory and its sub-directories.

   Sample solution for update_course_code.sh

```
#!/bin/sh

# doesn't handle pathnames containing white space
for file in `find "$@" -type f`
do
    temporary_file="$file.tmp.$$"
    if test -e "$temporary_file"
    then
        echo "$temporary_file" already exists
        exit 1
    fi
    sed 's/COMP2041/COMP2042/g;s/COMP9041/COMP9042/g' $file >$temporary_file &&
    mv $temporary_file $file
done
```

Sample solution for update_course_code.sh

```
#!/bin/sh
# doesn't hande pathnames containing new lines
find "$@" -type f|
while read file
do
    temporary_file="$file.tmp.$$"
    if test -e "$temporary_file"
    then
        echo "$temporary_file" already exists
        exit 1
    fi
    sed 's/COMP2041/COMP2042/g;s/COMP9041/COMP9042/g' $file >$temporary_file &&
    mv $temporary_file $file
done
```

4. Write a shell script, `is_business_hours` which exits with a status of 0 if the current time is between 9am & 5pm, and otherwise exits with a status of 1.
   Hint: the date command prints the current time in a format like this:

```
$ date
Sun Mar 18 12:57:08 EST 2012
```

Sample solution for is_business_hours.sh

```
#!/bin/sh
# exits with a status of 0  if the current time is between 9am & 5pm
# otherwise exit with a status 1
#
# date format looks like this Sun Mar 18 12:57:08 EST 2012

current_hour=`date|cut -d\  -f4|cut -d: -f1`
if test $current_hour -ge 9 -a $current_hour -lt 17
then
    exit 0
else
    exit 1
fi
```

Another sample solution for is_business_hours.sh

```
#!/bin/sh
# exits with a status of 0  if the current time is between 9am & 5pm
# otherwise exit with a status 1
# date output looks like this 'Sun Mar 18 12:57:08 EST 2012'
# relies on the exit status being the exit status of last command
# when there isn't an explicit exit

current_hour=`date|cut -d\  -f4|cut -d: -f1`
test $current_hour -ge 9 -a $current_hour -lt 17
```

5. CSE systems have a command, `mlalias`, which prints information about a specified mail alias. For example:

```
$ mlalias COMP2041-list
        alias: COMP2041-list
  description: Udb alias list
    addresses:
               blix573
               mhuz728
               .......
               sngx602
               andrewt
       owners: udb, cs2041
authorised posters: @Employee, @Subject_Utility, @Wheel
    Moderator: udb
       Status: system, closed, moderated, virtual, and public
```

Convert the output of the mlalias command into a new line separated list of CSE usernames, like this:

```
blix573
mhuz728
.......
sngx602
andrewt
```

```
mlalias COMP2041-list|grep -v :|sed 's/^ *//'
```

6. CSE system have a command, `acc`, which prints information about a specified user. For example:

```
$ acc mzhou
           User Name : mzhou              Aliases : myzh046
                 Uid : 25068
              Groups : cs1917
             Expires : 31 Aug 2012
        User classes : 3978_Student, COMP2041_Student[15jul2012]
                     : COMP2121_Student[15jul2012], COMP2911_Studen
                     : COMP1917_Tutor[16jul2012], COMP3901_Student[
        Misc classes : WirelessAccess[23jun2012]
                Name : Michael Yang Zhou
Password last changed : 2011/03/02.21:23:19
      Home Directory : /import/adams/1/mzhou
     Waste Basket UID : 65619
 Printer Usage Status : Pre-census Allocation       2045 (More will
                     : Used                           35
                     : Available                    2010
                     : set at 12:05 AM 19/Mar/2012
      Daily IP Quota : 350.0MB
    Session IP Quota : 700.0MB (more will be available from 31 Marc
    Session IP Usage : 275.3MB
```

Write a pipeline which converts the output of acc into a new line separated list of courses the person is enrolled in, like this:

```
COMP2041
COMP2121
COMP2911
COMP3901
```

```
acc mzhou | tr -s ', ' '\n' | grep '[A-Z][A-Z][A-Z][A-Z][0-9][0-9][0-9][0-9]_Student' | cut -c1-8
```

7. Use the pipeines from the above 2 questions to write shell commands which print a list of courses taken by COMP2041 students with counts of how many COMP2041 students take each, like this:

```
     55 COMP2911
     37 COMP2121
     17 COMP3311
     10 COMP2111
      9 COMP3331
      ..........
```

```
mlalias COMP2041-list|
grep -v :|
sed 's/^ *//'|
while read cseusername
do
    acc $cseusername|
    tr -s ', ' '\n' |
    grep '[A-Z][A-Z][A-Z][A-Z][0-9][0-9][0-9][0-9]_Student'|
    cut -c1-8
done|
sort|
uniq -c|sort -rn
```

8. COMP2041 student Shruti has a 'friends' subdirectory in her home directory that contains images of her many friends. Shruti likes to view these images often and would like to have them appear in other directories within her CSE account so she has written a shell script to symbolically link them to the current directory:

```
for image_file in `ls ~/friends`
do
    ln -s "~/friends/$image_file" .
done
```

The links created by Shruti's script are broken. Why? How can she fix her script?

The shell does not replace tilde (~) with the user's home directory inside double-quotes, and does not handle spaces in filenames correctly. For example:

```
$ echo ~
/home/shruti
$ echo "~"
~
$ touch a\ b
$ for f in `ls`; do echo $f; done
a
b
```

This should work for Shruti:

```
for image_file in ~/friends/*
do
    ln -s "$image_file" .
done
```

9. Implement a shell script called `iseq` for writing sequences of integers onto its standard output, with one integer per line. The script can take up to three arguments, and behaves as follows:
   - `iseq` *n* writes all numbers from 1 up to *n*, inclusive
   - `iseq` *m n* writes all numbers from *m* up to *n* inclusive
   - `iseq` *k m n* writes the sequence *k*, *m*, *m+(m-k)*, *m+2(m-k)* up to *p*
     (where *p* is the largest integer in this sequence that is less than or equal to *n*)

   Examples of output:

```
$ iseq 5       $ iseq 2 6       $ iseq 3 3 16
    1                  2                   3
    2                  3                   6
    3                  4                   9
    4                  5                   12
    5                  6                   15
```

Sample solution for iseq.sh

```
#!/bin/sh
# Write the sequence of integers specified by command-line arguments
# Check and handle command-line args

case $# in
1) lo=1   hi=$1  inc=1 ;;
2) lo=$1  hi=$2  inc=1 ;;
3) lo=$1  hi=$3  inc=`expr $2 - $1` ;;
*) cat <<EOI
Usage:
  $0 hi         ... prints numbers in range 1..hi
  $0 lo hi      ... prints numbers in range lo..hi
  $0 lo 2nd hi  ... prints numbers lo,2nd,..hi
                    (using difference between lo and
                     2nd as the increment)
EOI
   exit 1 ;;
esac

# Generate integer sequence

i=$lo
while test $i -le $hi
do
    echo $i
    i=`expr $i + $inc`
done
```

Another sample solution for iseq.sh

```
#!/bin/sh
# Write the sequence of integers specified by command-line arguments
# The bash-specific syntax ((..)) is used for arithmetic

if (($# == 1))
then
    lo=1
    hi=$1
    inc=1
elif (($# == 2))
then
    lo=$1
    hi=$2
    inc=1
elif (($# == 3))
then
    lo=$1
    hi=$3
    inc=$(($2 - $1))
else
    cat <<EOI
Usage:
  $0 hi         ... prints numbers in range 1..hi
  $0 lo hi      ... prints numbers in range lo..hi
  $0 lo 2nd hi  ... prints numbers lo,2nd,..hi
                    (using difference between lo and
                     2nd as the increment)
EOI
   exit 1
fi

i=$lo
while (($i <= $hi))
do
    echo $i
    i=$(($i + $inc))
done
```

10. Write a shell script named `isprime` which given an integer as argment, tests whether it is prime and prints a suitable message:

```
$ isprime 42
42 is not prime
$ isprime 113
113 is prime
```

Your script should exit with a non-zero exit status if its argument is not prime.

Write a second script named `primes` which uses the first script to print all primes less than a specified value,e.g:

```
$ primes 100
2
3
5
7
11
13
17
...
79
83
89
97
```

Another sample solution for isprime.sh

```
#!/bin/sh
# test whether the specified integer is prime

if test $# != 1
then
    echo "Usage: $0 <number>"
    exit 1
fi

n=$1

i=2
while test $i -lt $n
do
    if test `expr $n % $i` -eq 0
    then
        echo "$n is not prime"
        exit 1
    fi
    i=`expr $i + 1`
done
echo "$n is prime"
```

Yet another solution for isprime.sh

```
#!/bin/sh
# test whether the specified integer is prime
# written by Han Zhao

if test $# != 1
then
        echo "Usage: $0 <number>"
        exit 1
fi

n=$1
if [ $1 -eq 1 ]
then
        echo "$n is not prime"
        exit 1
fi

i=2
while test $(expr $i \* $i) -le $n
do
        if test `expr $n % $i` -eq 0
        then
                echo "$n is not prime"
                exit 1
        fi
        i=`expr $i + 1`
done
echo "$n is prime"
exit 0
```

Sample solution for primes.sh

```
#!/bin/sh
# print the prime numbers less than the specified argument

case $# in
1) ;;
*) echo "Usage: $0 <number>"; exit 1
esac
limit=$1

p=2
while test $p -lt $limit
do
    if isprime.sh $p >/dev/null
    then
        echo $p
    fi
    p=`expr $p + 1`
done
exit 0
```

11. Write a shell script, `list_include_files` , which given a list of C source files (`.c` files) as arguments, prints the names of the files they include (`.h` files), reporting each file only once, e.g.:

```
$ list_include_files count_words.c get_word.c map.c
ctype.h
get_word.h
map.h
stdio.h
stdlib.h
time.h
```

Sample solution for is_business_hours.sh

```
#!/bin/sh
# list the files included by the C sources files included as arguments
if test $# = 0
then
    echo "Usage: $0 <files>"
    exit 1
fi

egrep '^#include' "$@"|  # find '#include lines
sed 's/[">][^">]*$//'|  # remove the last '"' or '>' and anything after it
sed 's/^.*["<]//'|      # remove the last '"' or '>' and anything before it
sort|                   # sort the file names
uniq                    # remove any duplicates
```

12. COMP2041 student Big Bad Barry tries to impress a girl at a party by betting her she can't work out what this shell script:

```
#!/bin/sh
IFS=abc
echo "$*"
```

prints when run like this:

```
$ ./script.sh mount inside
```

What does the script print?

Will the girl go out with Big Bad Barry?

The script will do this:

```
$ ./script.sh mount inside
mountainside
```

This is because `IFS` is a special internal shell variable which indicates the argument separators. The first character from IFS is uses to separate the argument when expanding $*.

Big Bad Barry won't get a date. No one is impressed by knowledge of hacky&obscure shell features. Good programmers avoid quirky little-known language features.