Week 02 Tutorial Questions

1. Imagine that we have just typed a shell script into the file `my_first_shell_script.sh` in the current directory. We then attempt to execute the script and observe the following:

   ```
   $ my_first_shell_script.sh
   my_first_shell_script.sh: command not found
   ```

   Explain the possible causes for this, and describe how to rectify them.

2. Implement a shell script called `seq.sh` 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:

   - `seq.sh` *`LAST`* writes all numbers from 1 up to *`LAST`*, inclusive. For example:

     ```
     $ ./seq.sh 5
     1
     2
     3
     4
     5
     ```

   - `seq.sh` *`FIRST LAST`* writes all numbers from *`FIRST`* up to *`LAST`*, inclusive. For example:

     ```
     $ ./seq.sh 2 6
     2
     3
     4
     5
     6
     ```

   - `seq.sh` *`FIRST INCREMENT LAST`* writes all numbers from *`FIRST`* to *`LAST`* in steps of *`INCREMENT`*, inclusive; that is, it writes the sequence *`FIRST`*, *`FIRST + INCREMENT`*, *`FIRST + 2*INCREMENT`*, ..., up to the largest integer in this sequence less than or equal to *`LAST`*. For example:

     ```
     $ ./seq.sh 3 5 24
     3
     8
     13
     18
     23
     ```

3. Write a shell script, `no_blinking.sh`, which removes all HTML files in the current directory which use the blink element:

   ```
   $ no_blinking.sh
   Removing old.html because it uses the <blink> tag
   Removing evil.html because it uses the <blink> tag
   Removing bad.html because it uses the <blink> tag
   ```

4. Modify the `no_blinking.sh` shell script to instead take the HTML files to be checked as command line arguments and, instead of removing them, adding the suffix **.bad** to their name:

   ```
   $ no_blinking.sh awful.html index.html terrible.html
   Renaming awful.html to awful.html.bad because it uses the <blink> tag
   Renaming terrible.html to terrible.html.bad because it uses the <blink> tag
   ```

5. Write a shell script, **list_include_files.sh**, which for all the C source files (`.c` files) in the current directory prints the names of the files they include (`.h` files), for example

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

6. Consider the following columnar (space-delimited) data file containing (fictional) contact for various CSE academic staff:

```
G Heiser       Newtown      9381-1234
S Jha          Kingsford    9621-1234
C Sammut       Randwick     9663-1234
R Buckland     Randwick     9663-9876
J A Shepherd   Botany       9665-4321
A Taylor       Glebe        9692-1234
M Pagnucco     North Ryde   9868-6789
```

Note: This data is fictitious. Do not ring these phone numbers. I have no idea whether they are real or not, but they are certainly not the correct phone numbers for the academic staff mentioned.

The data is currently sorted in phone number order. Can we use the Ýr wǼǍǬ filter to re-arrange the data into "telephone-book" order? If not, how would we need to change the file in order to achieve this?

7. Consider this Unix password file (usually found in `/etc/passwd`):

```
root:ZHolHAHZw8As2:0:0:root:/root:/bin/bash
jas:iaiSHX49Jvs8.:100:100:John Shepherd:/home/jas:/bin/bash
andrewt:rX9KwSSPqkLyA:101:101:Andrew Taylor:/home/andrewt:/bin/cat
postgres::997:997:PostgreSQL Admin:/usr/local/pgsql:/bin/bash
oracle::999:998:Oracle Admin:/home/oracle:/bin/bash
cs2041:rX9KwSSPqkLyA:2041:2041:COMP2041 Material:/home/cs2041:/bin/bash
cs3311:mLRiCIvmtI9O2:3311:3311:COMP3311 Material:/home/cs3311:/bin/bash
cs9311:fIVLdSXYoVFaI:9311:9311:COMP9311 Material:/home/cs9311:/bin/bash
cs9314:nTn.JwDgZE1Hs:9314:9314:COMP9314 Material:/home/cs9314:/bin/bash
cs9315:sOMXwkqmFbKlA:9315:9315:COMP9315 Material:/home/cs9315:/bin/bash
```

Provide a command that would produce each of the following results:

a. display the first three lines of the file

b. display lines belonging to class accounts (assuming that their login name starts with either "cs", "se", "bi" or "en", followed by a digit)

c. display the user name of everyone whose shell is `/bin/bash`

d. create a tab-separated file `passwords.txt` containing only login name and password for all users

8. The following shell script emulates the ùl ǼǍǬ command using the built-in shell commands ẃǴ ĶǍǬ and ǴùĎŕ ǍǬ

```
#!/bin/sh
while read line
do
    echo "$line"
done
```

a. What are the differences between the above script and the real ùl ǼǍǬ command?

b. modify the script so that it can concatenate multiple files from the command line, like the real ùl ǼǍǬ

(Hint: the shell's control structures — for example, `if`, `while`, `for` — are commands in their own right, and can form a component of a pipeline.)

9. The þŽɏ ǍǬ command compresses a text file, and renames it to *`filename`*`.gz`. The Żùl ǼǍǬ command takes the name of a single compressed file as its argument and writes the original (non-compressed) text to its standard output.

Write a shell script called `zshow` that takes multiple `.gz` file names as its arguments, and displays the original text of each file, separated by the name of the file separated by the name of the file.

Consider the following example execution of `zshow`:

```
$ zshow a.gz b.gz bad.gz c.gz
===== a =====
... original contents of file "a" ...
===== b =====
... original contents of file "b" ...
===== bad =====
No such file: bad.gz
===== c =====
... original contents of file "c" ...
```

10. Consider the marks data file from last week's tutorial; assume it's stored in a file called `Marks`:

```
2111321 37 FL
2166258 67 CR
2168678 84 DN
2186565 77 DN
2190546 78 DN
2210109 50 PS
2223455 95 HD
2266365 55 PS
...
```

Assume also that we have a file called `Students` that contains the names and student ids of for all students in the class, e.g.

```
2166258 Chen, X
2186565 Davis, PA
2168678 Hussein, M
2223455 Jain, S
2190546 Phan, DN
2111321 Smith, JA
2266365 Smith, JD
2210109 Wong, QH
...
```

Write a shell script that produces a list of names and their associated marks, sorted by name:

```
67 Chen, X
77 Davis, PA
84 Hussein, M
95 Jain, S
78 Phan, DN
37 Smith, JA
55 Smith, JD
50 Wong, QH
```

Note: there are many ways to do this, generally involving combinations of filters such as [illegible], [illegible], [illegible], [illegible], etc. Try to think of more than one solution, and discuss the merits of each.

11. Implement a shell script, `grades.sh`, that reads a sequence of (studentID, mark) pairs from its standard input, and writes (studentID, grade) pairs to its standard output. The input pairs are written on a single line, separated by spaces, and the output should use a similar format. The script should also check whether the second value on each line looks like a valid mark, and output an appropriate message if it does not The script can ignore any extra data occurring after the mark on each line.

Consider the following input and corresponding output to the program:

Input

```
2212345 65
2198765 74
2199999 48
2234567 50 ok
2265432 99
2121212 hello
2222111 120
2524232 -1
```

Output

```
2212345 CR
2198765 CR
2199999 FL
2234567 PS
2265432 HD
2121212 ?? (hello)
2222111 ?? (120)
2524232 ?? (-1)
```

To get you started, here is a framework for the script:

```
#!/bin/sh
while read id mark
do
    # ... insert mark/grade checking here ...
done
```

Note that the `read` shell builtin assumes that the components on each input line are separated by spaces. How could we use this script if the data was supplied in a file that used commas to separate the (studentID, mark) components, rather than spaces?

Revision questions

The following questions are primarily intended for revision, either this week or later in session. Your tutor may still choose to cover some of these questions, time permitting.

19. Write a shell script, `time_date.sh`, that prints the time and date once an hour. It should do this until a new month is reached.

    Reminder the Ǩl ǍĢǬǍǬ command produces output like this:

    ```
    Friday 5 August  17:37:01 AEST 2016
    ```

20. Consider a scenario where we have a directory containing two LaTeX files, `a.tex` and `b.tex`. The file `a.tex` is 20 lines long, and `b.tex` is 30 lines long. What is the effect of each of the commands below? How will their output differ?

    ```
    $ wc -l *.tex
    $ echo `wc -l *.tex`
    ```

21. Write a shell script that displays the name and size of all files in the current directory that are bigger than, say, 100,000 bytes.

    (Hint: use ỹ ǜ ǬǍǬ to do the counting, and capture its output using back-ticks. How do you get rid of the file name and/or line and word counts?)

22. What is the output of each of the following pipelines if the text

    ```
    this is big Big BIG
    but this is not so big
    ```

    is supplied as the initial input to the pipeline?

    a. `tr -d ' ' | wc -w`
    b. `tr -cs 'a-zA-Z0-9' '\n' | wc -l`
    c. `tr -cs 'a-zA-Z0-9' '\n' | tr 'a-z' 'A-Z' | sort | uniq -c`

23. Consider the fairly standard “split-into-words” technique from the previous question —

    ```
    tr -c -s 'a-zA-Z0-9' '\n' < someFile
    ```

    Explain how this command works. What does each argument do?

24. Assume that we are in a shell where the following shell variable assignments have been performed, and ǨŸǬǍǬ gives the following result:

    ```
    $ x=2  y='Y Y'  z=ls
    $ ls
        a       b       c
    ```

    What will be displayed as a result of the following ǴǜĐ ǬǍǬ commands:

    a. `$ echo a    b    c`
    b. `$ echo "a    b    c"`
    c. `$ echo $y`
    d. `$ echo x$x`
    e. `$ echo $xx`
    f. `$ echo "$y"`
    g. `$ echo '$y'`
    h. ``$ echo `$y` ``
    i. ``$ echo `$z` ``
    j. ``$ echo `echo a b c` ``

25. The following C program and its equivalent in Java both aim to give precise information about their command-line arguments.

```c
// Display command line arguments, one per line, in C
#include <stdio.h>
int main (int argc, char *argv[])
{
    printf ("#args  = %d\n", argc - 1);
    for (int i = 1; i < argc; i++)
        printf ("arg[%d] = \"%s\"\n", i, argv[i]);
    return 0;
}
```

```java
// Display command line arguments, one per line, in Java
public class args {
    public static void main(String args[]) {
        System.out.println("#args  = " + args.length);
        for (int i = 0; i < args.length; i++)
            System.out.println("arg[" + (i+1) + "] = \"" + args[i] + "\"");
    }
}
```

Assume that these programs are compiled in such a way that we may invoke them as `args`. Consider the following examples of how it operates:

```
$ args a b c
#args  = 3
arg[1] = "a"
arg[2] = "b"
arg[3] = "c"
$ args "Hello there"
#args  = 1
arg[1] = "Hello there"
```

As with the previous question, assume that we are in a shell where the following shell variable assignments have been performed, and [illegible] gives the following result:

```
$ x=2  y='Y Y'  z=ls
$ ls
    a       b       c
```

What will be the output of the following:

a. `$ args x y   z`

b. ``$ args `ls` ``

c. `$ args $y`

d. `$ args "$y"`

e. ``$ args `echo "$y"` ``

f. `$ args $x$x$x`

g. `$ args $x$y`

h. `$ args $xy`