COMP(2041|9044) 19T2

Week 02 Laboratory Sample Solutions

COMP(2041|9044) 19T2

Objectives

- More practice with UNIX filters
- Introduction to writing Shell Scripts

Preparation

Before the lab you should re-read the relevant lecture slides and their accompanying examples.

Getting Started

Create a new directory for this lab called `lab02` by typing:

```
$ mkdir lab02
```

Change to this directory by typing:

```
$ cd lab02
```

Exercise: Counting UNSW Classes with Unix Filters (individual)

This is an individual exercise to complete by yourself.

There is a template file named `counting_classes_answers.txt` which you must use to enter the answers for this exercise.
Download counting_classes_answers.txt here, or copy it to your CSE account using the following command:

```
$ cp -n /web/cs2041/19T2/activities/counting_classes/counting_classes_answers.txt .
```

The autotest scripts depend on the format of `counting_classes_answers.txt` so just add your answers don't otherwise change the file. In other words edit `counting_classes_answers.txt`:

```
$ gedit counting_classes_answers.txt &
```

The file `classes.txt` contains a list of CSE classes downloaded from myUNSW. Download classes.txt here, or copy it to your CSE account using the following command:

```
$ cp -n /web/cs2041/19T2/activities/counting_classes/classes.txt .
```

1. Write a shell pipeline to print how many classes there are.
   Hint: the output of the pipeline should be:

   ```
   441
   ```

   Each line in the file represents a class. Therefore, the number of classes is equal to the number of lines in the file.

   ```
   wc -l <classes.txt
   ```

2. Write a shell pipeline to print how many different courses have classes.
   Hint: `cut` with the `-f` option will be useful here.

Hint: the output of the pipeline should be:

```
35
```

Approach: extract just the course codes, then sort them into groups of identical course codes, then compress each group to size one, giving one line for each group (i.e. each course). Then count the number of lines.

```
cut -f1 classes.txt | sort | uniq | wc -l
```

3. Write a shell pipeline which will print the course with the most classes (and no other courses) and how many classes are in this course.
   Hint: the output of the pipeline should be:

```
      31 COMP1521
```

```
cut -f1 classes.txt | sort | uniq -c|sort -n|tail -1
```

4. Write a shell pipeline that prints the room most frequently-used room by CSE classes and how often it is used. Don't include the CSE lab rooms.

   Hint: the output of the pipeline should be:

```
      26 Quad 1042
```

Approach: extract the non-lab classes, cut out the room names, then sort them into groups, then count the number of entries in each group, then sort numerically and grab the last line.

```
egrep -v 'LAB' classes.txt|cut -f5|sort|uniq -c|sort -n|tail -1
```

The last two steps in the above pipelines could be changed to `sort -nr | head -1`

5. Write a shell pipeline that prints the most common day and hour in the week for classes to start and how many classes start at that time.
   Hint: `cut` has a `-c` option.

   Hint: the output of the pipeline should be:

```
      20 Fri 11
```

```
cut -f4 classes.txt|cut -d- -f1|sort|uniq -c|sort -n|tail -1
```

The last two steps in the above pipelines could be changed to `sort -nr | head -1`

6. Challenge: Write a shell pipeline that prints a list of the course codes (only) of COMP courses that run 2 or more classes of the same type starting at the same time on the same day (e.g. three tut-labs starting Monday at 10:00).
   Hint: this should be the output of your pipeline:

```
COMP1000
COMP1511
COMP1521
COMP2041
COMP2511
COMP2521
COMP3331
COMP6441
COMP6841
COMP9044
COMP9311
COMP9313
COMP9331
COMP9417
```

This question needs a little thinking. The idea is that we're interested in any course where two classes are running on the same day at the same time. The info we want is contained in fields 1, 3 and 4 of each line. If we cut these lines out, then sort the file, then count the size of each group, we could extract the final result by picking out groups whose size was different to '1'. The following pipeline uses this approach:

```
grep '^COMP' classes.txt|cut -f1,3,4 |uniq -d|cut -f1|sort|uniq
```

When you think your program is working you can use `autotest` to run some simple automated tests:

```
$ 2041 autotest counting_classes
```

When you are finished working on this exercise you must submit your work by running **give**:

```
$ give cs2041 lab02_counting_classes counting_classes_answers.txt
```

You must run give before **Tuesday 18 June 17:59:59** to obtain the marks for this lab exercise. Note, this is an individual exercise, the work you submit with **give** must be entirely your own.

Sample solution for `counting_classes_answers.txt`

```
This file is automarked.

Do not add extra lines to this file, just add your answers.

For example if your answer to Q1 is: egrep Andrew words.txt
Change the line that says Q1 answer to:

Q1 answer: egrep Andrew words.txt


1) Write a shell pipeline to print how many classes there are.

Q1 answer: wc -l <classes.txt


2) Write a shell pipeline to print how many different courses have classes.

Q2 answer: cut -f1 classes.txt | sort | uniq | wc -l


3) Write a shell pipeline which will print the course with the most classes (and no
other courses) and how many classes are in this course.

Q3 answer: cut -f1 classes.txt | sort | uniq -c|sort -n|tail -1


4) Write a shell pipeline that prints the room most frequently-used room by CSE classes and how ofter

Q4 answer: egrep -v 'LAB' classes.txt|cut -f5|sort|uniq -c|sort -n|tail -1


5) Write a shell pipeline  that prints the most popular time-of-day for classes to
start and how many classes start at that time.

Q5 answer: cut -f4 classes.txt|cut -d- -f1|sort|uniq -c|sort -n|tail -1


6) Challenge: Write a shell pipeline that prints a list of the course codes (only) of COMP courses th
run 2 or more  classes of the same type starting at the same time on the same day (e.g. three tut-lab

Q6 answer: grep '^COMP' classes.txt|cut -f1,3,4 |uniq -d|cut -f1|sort|uniq

Q7 answer:
```

Exercise: Mapping Digits (individual)

This is an individual exercise to complete by yourself.

Write a program `digits.sh` that reads from standard input and writes to standard output mapping all digit characters whose values are less than 5 into the character '**<**' and all digit characters whose values are greater than 5 into the character '**>**'. The digit character '5' should be left unchanged.

Sample Input Data	**Corresponding Output**
`1 234 5 678 9`	`< <<< 5 >>> >`

`I can think of 100's`
`of other things I'd rather`
`be doing than these 3 questions`
	`I can think of <<<'s`
	`of other things I'd rather`
	`be doing than these < questions`

A line with lots of numbers: 12345678912345678912345678 9 A line with all zeroes 0000000000000000000000000 A line with blanks at the end 1 2 3	A line with lots of numbers: <<<<5>>>>>>>><<<<5>>>>>>>><5>>>> A line with all zeroes <<<<<<<<<<<<<<<<<<<<<<<<<< A line with blanks at the end < < <
Input with absolutely 0 digits in it Well ... apart from that one ...	Input with absolutely < digits in it Well ... apart from that one ...
1 2 4 8 16 32 64 128 256 512 1024 2048 4096 8192 16384 32768 65536	< < < > <> << >< <<> <5> 5<< <<<< <<<> <<>> ><>< <><>< <<>>> >55<>

Hint: `tr` can be used.

When you think your program is working you can use `autotest` to run some simple automated tests:

```
$ 2041 autotest digits
```

When you are finished working on this exercise you must submit your work by running **give**:

```
$ give cs2041 lab02_digits digits.sh
```

You must run give before **Tuesday 18 June 17:59:59** to obtain the marks for this lab exercise. Note, this is an individual exercise, the work you submit with **give** must be entirely your own.

Sample solution for `digits.sh`

```
#!/bin/sh
tr '0123456789' '<<<<<5>>>>>'
```

Exercise: Is There An Echo In Here? (individual)

This is an individual exercise to complete by yourself.

Write a shell script called `echon.sh` which given exactly two arguments, an integer *n* and a string, prints the string *n* times. For example:

```
$ ./echon.sh 5 hello
hello
hello
hello
hello
hello
$ ./echon.sh 0 nothing
$ ./echon.sh 1 goodbye
goodbye
```

Your script should print exactly the error message below if it is not given exactly 2 arguments:

```
$ ./echon.sh
Usage: ./echon.sh <number of lines> <string>
$ ./echon.sh 1 2 3
Usage: ./echon.sh <number of lines> <string>
```

Also get your script to print this error message if its first argument isn't a non-negative integer:

```
$ ./echon.sh hello world
./echon.sh: argument 1 must be a non-negative integer
$ ./echon.sh -42 lines
./echon.sh: argument 1 must be a non-negative integer
```

Although its better practice to print your error messages to `stderr` its OK to print your error messages to stdout for this exercise.

Hint: you'll need to use the shell `if`, `while` and `exit` statements, shell arithmetic and the `test` command.

Discussion: Straight-forward shell programming except for checking that the first argument is an integer
We could use the shell `case` to check the number of command-line args and also to check that the first argument is a non-negative integer.

Note the display of a usage message, which gives useful feedback to the user about they should have done. Note also the use of `exit` to terminate the script if an error is discovered in the command line arguments.

When you think your program is working you can use `autotest` to run some simple automated tests:

```
$ 2041 autotest echon
```

When you are finished working on this exercise you must submit your work by running **give**:

```
$ give cs2041 lab02_echon echon.sh
```

You must run give before **Tuesday 18 June 17:59:59** to obtain the marks for this lab exercise. Note, this is an individual exercise, the work you submit with **give** must be entirely your own.

Sample solution for `echon.sh`

```
#!/bin/sh

# check command-line args
if test $# != 2
then
    echo "Usage: $0 <number of lines> <string>"
    exit 1
fi

# standard error redirected because test will print
# a warning message if $1 is not an integer

if test "$1" -ge 0 2>/dev/null
then
    :
else
    echo "$0: argument 1 must be a non-negative integer"
    exit 1
fi

number_of_lines=$1
text=$2

line_count=0
while test $line_count -lt $number_of_lines
do
    echo $text
    line_count=$((line_count + 1))
done

exit 0
```

Exercise: How Big is That File? (individual)

This is an individual exercise to complete by yourself.

Write a shell script `file_sizes.sh` which prints the names of the files in the current directory splitting them into three categories: *small*, *medium-sized* and *large*. A file is considered *small* if it contains less than 10 lines, *medium-sized* if contains less than 100 lines, otherwise it is considered *large*.
Your script should always print exactly three lines of output. Files should be listed in alphabetic order on each line. Your shell-script should match character-for-character the output shown in the example below. Notice the creation of a separate direcory for testing and the use of the script from the last question to produce test files. You could also produce test files manually using an editor.

```
$ mkdir test
$ cd test
$ ../echon.sh 5 text >a
$ ../echon.sh 505 text >bbb
$ ../echon.sh 17 text >cc
$ ../echon.sh 10 text >d
$ ../echon.sh 1000 text >e
$ ../echon.sh 0 text >empty
$ ls -l
total 24
-rw-r--r-- 1 andrewt andrewt   25 Mar 24 10:37 a
-rw-r--r-- 1 andrewt andrewt 2525 Mar 24 10:37 bbb
-rw-r--r-- 1 andrewt andrewt   85 Mar 24 10:37 cc
-rw-r--r-- 1 andrewt andrewt   50 Mar 24 10:37 d
-rw-r--r-- 1 andrewt andrewt 5000 Mar 24 10:37 e
-rw-r--r-- 1 andrewt andrewt    0 Mar 24 10:37 empty
$ ../file_sizes.sh
Small files: a empty
Medium-sized files: cc d
Large files: bbb e
$ rm cc d
$ ../echon.sh 10000 . >lots_of_dots
$ ls -l
total 36
-rw-r--r-- 1 andrewt andrewt    25 Mar 24 10:37 a
-rw-r--r-- 1 andrewt andrewt  2525 Mar 24 10:37 bbb
-rw-r--r-- 1 andrewt andrewt  5000 Mar 24 10:37 e
-rw-r--r-- 1 andrewt andrewt     0 Mar 24 10:37 empty
-rw-r--r-- 1 andrewt andrewt 20000 Mar 24 10:39 lots_of_dots
$ ../file_sizes.sh
Small files: a empty
Medium-sized files:
Large files: bbb e lots_of_dots
$
```

Hint: you can use the command `wc` to discover how many lines are in a file. You probably want to use the shell's back quotes, its `if` statement, and the `test` command.
When you think your program is working you can use `autotest` to run some simple automated tests:

```
$ 2041 autotest file_sizes
```

When you are finished working on this exercise you must submit your work by running **give**:

```
$ give cs2041 lab02_file_sizes file_sizes.sh
```

You must run give before **Tuesday 18 June 17:59:59** to obtain the marks for this lab exercise. Note, this is an individual exercise, the work you submit with **give** must be entirely your own.

Sample solution for `file_sizes.sh`

```
#!/bin/sh

for file in *
do
    lines=`wc -l <$file`
    if test $lines -lt 10
    then
        small_files="$small_files $file"
    elif test $lines -lt 100
    then
        medium_files="$medium_files $file"
    else
        large_files="$large_files $file"
    fi
done

echo "Small files:$small_files"
echo "Medium-sized files:$medium_files"
echo "Large files:$large_files"

exit 0
```

Challenge Exercise: Scraping Web Pages with a Shell Script (individual)

This is an individual exercise to complete by yourself.

Write a shell script `scraping_courses.sh` which prints a list of UNSW courses with the given prefix by extracting them from the 2018 UNSW handbook webpages.
This year UNSW has changed to much prettier format but also a format which it is much harder for a script to extract information from.

So for this exercise we'll use the 2018 handbook pages which aren't

For example:

```
$ scraping_courses.sh OPTM
OPTM2111 Optometry 2A
OPTM2133 The Clinical Environment
OPTM2190 Introduction to Clinical Optometry
OPTM2211 Optometry 2B
OPTM2233 Optical Dispensing
OPTM2291 Primary Care Optometry
OPTM3105 Disease Processes of the Eye 1
OPTM3111 Optometry 3A
OPTM3131 Ocular Disease 3A
OPTM3133 Vision Science in the Consulting Room
OPTM3201 Ocular Imaging & Applied Vision Science
OPTM3205 Disease Processes of the Eye 2
OPTM3211 Optometry 3B
OPTM3231 Ocular Disease 3B
OPTM3233 Working in the clinical environment
OPTM4110 Optometry 4A
OPTM4131 Clinical Optometry 4A
OPTM4151 Ocular Therapeutics 4A
OPTM4211 Optometry 4B
OPTM4231 Clinical Optometry 4B
OPTM4251 Ocular Therapeutics 4B
OPTM4271 Professional Optometry
OPTM4291 Optometry, Medicine & Patient Management
OPTM5111 Clinical Optometry 5A
OPTM5131 Specialist Clinical Optometry 5A
OPTM5151 Clinical Ocular Therapeutics 5A
OPTM5171 Research Project 5A
OPTM5211 Clinical Optometry 5B
OPTM5231 Specialist Clinical Optometry 5B
OPTM5251 Clinical Ocular Therapeutics 5B
OPTM5271 Research Project 5B
OPTM6400 Optometric Preclinical Practice
OPTM6411 Contact Lenses
OPTM6412 Clinical Optometry 4A
OPTM6413 Anterior Eye Therapeutics
OPTM6421 Binocular Vision, Paediatrics and Low Vision
OPTM6422 Clinical Optometry 4B
OPTM6423 Therapeutics and the Posterior Eye
OPTM6424 Professional Optometry
OPTM7001 Introduction to Community Eye Health
OPTM7002 Epidemiology & Biostatistics for Needs Assessment
OPTM7003 Epidemiology of Blinding Eye Diseases
OPTM7004 Advocacy and Education in Community Eye Health
OPTM7005 Eye Health Economics and Sustainability
OPTM7006 Eye Care Program Management
OPTM7007 Community Eye Health Project
OPTM7103 Behavioural Optometry 1
OPTM7104 Advanced Contact Lens Studies 1
OPTM7108 Research Skills in Optometry
OPTM7110 Public Health Optometry
OPTM7115 Visual Neuroscience
OPTM7117 Ocular Therapy 2
OPTM7203 Behavioural Optometry 2
OPTM7205 Specialty Contact Lens Studies
OPTM7213 Ocular Therapy
OPTM7301 Advanced Clinical Optometry
```

```
OPTM7302 Evidence Based Optometry
OPTM7308 Research Project
OPTM7444 Business Skills in Optometry
OPTM7511 Advanced Ocular Disease 1
OPTM7521 Advanced Ocular Disease 2
OPTM8511 Clinical paediatrics, low vision and colour vision
OPTM8512 Clinical Optometry 5A
OPTM8513 Clinical Ocular Therapy 5A
OPTM8518 Optometry Research Project A
OPTM8521 Clinical Contact Lenses
OPTM8522 Clinical Optometry 5B
OPTM8523 Clinical Ocular Therapy 5B
OPTM8528 Optometry Research Project B
$ scraping_courses.sh MATH|wc
    126     585    4874
$ scraping_courses.sh COMP|grep Soft
COMP1531 Software Engineering Fundamentals
COMP2041 Software Construction: Techniques and Tools
COMP3141 Software System Design and Implementation
COMP3431 Robotic Software Architecture
COMP4161 Advanced Topics in Software Verification
COMP4181 Language-based Software Safety
COMP6447 System and Software Security Assessment
COMP9041 Software Construction: Techniques and Tools
COMP9181 Language-based Software Safety
COMP9322 Software Service Design and Engineering
COMP9323 Software as a Service Project
COMP9431 Robotic Software Architecture
$ scraping_courses.sh MINE|grep Rock
MINE3630 Rock Breakage
MINE8640 Geotechnical Hazards in Hard Rock Mines
MINE8660 Geotechnical Engineering for Underground Hard Rock
```

Your script must download the handbook web pages and extract the information from them when it is run.

Hints

This task can be done using the usual tools of `egrep`, `sed`, `sort` & `uniq` but the regular expressions take some thought.
The UNSW handbook uses seperate web pages for undergraduate and postgraduate courses. These two web pages would need to be downloaded for the above example (OPTM):
http://legacy.handbook.unsw.edu.au/vbook2018/brCoursesByAtoZ.jsp?StudyLevel=Undergraduate&descr=0 and
http://legacy.handbook.unsw.edu.au/vbook2018/brCoursesByAtoZ.jsp?StudyLevel=Postgraduate&descr=0.

Make sure courses which occur in both postgraduate & undergraduate handbooks aren't repeated.

`cat -A` can be useful to check for non-printing characters.

The command `curl` will download a URL and print it to standard output.

In a script it is best run as `curl --silent` so it doesn't print extra information on standard error.

For example:

```
$ curl --silent "http://legacy.handbook.unsw.edu.au/vbook2018/brCoursesByAtoZ.jsp?StudyLevel=Undergraduate&descr=O"|grep OPTM
                                    <TD class="" align="left">OPTM2111</TD>
                                    <TD class=""><A href="http://www.handbook.unsw.edu.au/undergraduate/courses/2018/OPTM2111.html">Optometry 2A</A></TD>
                                    <TD class="evenTableCell" align="left">OPTM2133</TD>
                                    <TD class="evenTableCell"><A href="http://www.handbook.unsw.edu.au/undergraduate/courses/2018/OPTM2133.html">The Clinical Environment </A></TD>
                                    <TD class="" align="left">OPTM2190</TD>
                                    <TD class=""><A href="http://www.handbook.unsw.edu.au/undergraduate/courses/2018/OPTM2190.html">Introduction to Clinical Optometry </A></TD>
                                    <TD class="evenTableCell" align="left">OPTM2211</TD>
                                    <TD class="evenTableCell"><A href="http://www.handbook.unsw.edu.au/undergraduate/courses/2018/OPTM2211.html">Optometry 2B</A></TD>
                                    <TD class="" align="left">OPTM2233</TD>
                                    <TD class=""><A href="http://www.handbook.unsw.edu.au/undergraduate/courses/2018/OPTM2233.html">Optical Dispensing </A></TD>
...
```

The program wget can be used for the same purpose, by running it as **wget -q -O-** ***url***

When you think your program is working you can use `autotest` to run some simple automated tests:

```
$ 2041 autotest scraping_courses
```

When you are finished working on this exercise you must submit your work by running **give**:

```
$ give cs2041 lab02_scraping_courses scraping_courses.sh
```

You must run give before **Tuesday 18 June 17:59:59** to obtain the marks for this lab exercise. Note, this is an individual exercise, the work you submit with **give** must be entirely your own.

Sample solution for `scraping_courses.sh`

```
#!/bin/sh

# written by andrewt@cse.unsw.edu.au
# Aug 2017 as a COMP2041 programming example

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

year=2018
course_prefix=$1
first_letter=`echo $course_prefix|sed 's/\(.\).*/\1/'`
base_url="http://legacy.handbook.unsw.edu.au/vbook$year/brCoursesByAtoZ.jsp"
ugrad_url="$base_url?StudyLevel=Undergraduate&descr=$first_letter"
pgrad_url="$base_url?StudyLevel=Postgraduate&descr=$first_letter"

curl -s "$ugrad_url" "$pgrad_url"|
egrep "$course_prefix[0-9][0-9][0-9][0-9].html"|
sed "s/.*\($course_prefix[0-9][0-9][0-9][0-9]\)\.html[^>]*> *\([^<]*\).*/\1 \2/"|
sed 's/ *$//'|
sort|
uniq
```

Submission

When you are finished each exercises make sure you submit your work by running **give**.
You can run **give** multiple times. Only your last submission will be marked.

Don't submit any exercises you haven't attempted.

If you are working at home, you may find it more convenient to upload your work via give's web interface.

Remember you have until **Tuesday 18 June 17:59:59** to submit your work.

You cannot obtain marks by e-mailing lab work to tutors or lecturers.

You check the files you have submitted here

Automarking will be run by the lecturer several days after the submission deadline for the test, using test cases that you haven't seen: different to the test cases `autotest` runs for you.

(Hint: do your own testing as well as running `autotest`)

After automarking is run by the lecturer you can view it here the resulting mark will also be available via via give's web interface

Lab Marks

When all components of a lab are automarked you should be able to view the the marks via give's web interface or by running this command on a CSE machine:

```
$ 2041 classrun -sturec
```

The lab exercises for each week are worth in total 1 mark.
The best 10 of your 11 lab marks will be summed to give you a mark out of 9. If their sum exceeds 9 - your mark will be capped at 9.

- You can obtain full marks for the labs without completing challenge exercises
- You can miss 1 lab without affecting your mark.