Week 03 Laboratory Sample Solutions

Objectives

- Understanding shell scripting

Preparation

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

Getting Started

Set up for the lab by creating a new directory called `lab03` and changing to this directory.

```
$ mkdir lab03
$ cd lab03
```

There are no provided files for this lab.

EXERCISE:
Balancing numbers

Write a POSIX-compatible shell script, `balancing_numbers.sh`, that reads from standard input and writes to standard output.
It should:

- map all digit characters whose values are less than 5 into the character ' `<` '.
- map all digit characters whose values are greater than 5 into the character ' `>` '.
- leave the digit character '5', and all non-digit characters, 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 5 questions`
	`I can think of <<<'s`
	`of other things I'd rather`
	`be doing than these 5 questions`

`A line with lots of numbers:`
`123456789123456789123456789`
`A line with all zeroes`
`000000000000000000000000000`
`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 1 ...`
	`Input with absolutely < digits in it`
	`Well ... apart from that < ...`

`1 2 4 8 16 32 64 128 256 512 1024`
`2048 4096 8192 16384 32768 65536`
	`< < < > <> << >< <<> <5> 5<< <<<<`
	`<<<> <<>> ><>< <><>< <<>>> >55<>`

HINT:

Make the first line of your shell-script `#!/bin/dash`

tr will be very useful.

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

```
$ 2041 autotest balancing_numbers
```

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

```
$ give cs2041 lab03_balancing_numbers balancing_numbers.sh
```

before **Monday 19 June 12:00 (midday)** (2023-06-19 12:00:00) to obtain the marks for this lab exercise.

SOLUTION:

Sample solution for `balancing_numbers.sh`

```
#! /usr/bin/env dash

tr '0123456789' '<<<<<5>>>>'
```

EXERCISE:

Can you hear my echo

Write a POSIX-compatible 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 1 goodbye
goodbye
$ ./echon.sh 5 hello
hello
hello
hello
hello
hello
$ ./echon.sh 0 nothing
```

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>
```

Your script should print exactly the error message below 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:

Make the first line of your shell-script #!/bin/dash

You will need to use:

- `if`
- `while`
- `exit`
- `$(())` (arithmetic)
- [(test)

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 lab03_echon echon.sh
```

before **Monday 19 June 12:00 (midday)** (2023-06-19 12:00:00) to obtain the marks for this lab exercise.

SOLUTION:

Sample solution for `echon.sh`

```
#! /usr/bin/env dash

program_name="$0"

# check the number of command-line args
case $# in
  2)
    number_of_lines="$1"
    text="$2"
    ;;
  *)
    echo "Usage: ${program_name} <number of lines> <string>" >& 2
    exit 1
    ;;
esac

# stderr redirected because `test` will print a warning message if `$number_of_lines` is not an
integer
if [ "${number_of_lines}" -ge 0 ] 2> /dev/null; then
  :
else
  echo "${program_name}: argument 1 must be a non-negative integer" >& 2
  exit 1
fi

line_count=0
while [ "${line_count}" -lt "${number_of_lines}" ]; do
  echo "${text}"
  line_count=$((line_count + 1))
done

exit 0
```

EXERCISE:

Categorising sizes

Write a POSIX-compatible shell script, `file_sizes.sh`, which prints the names of the files in the current directory splitting them into three categories:

- *small* if the file contains less than 10 lines
- *medium-sized* if the file contains between 10 and 100 lines
- *large* if the file contains 100 or more lines

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 directory 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:

Make the first line of your shell-script `#!/bin/dash`

You can use the command *wc* to discover how many lines are in a file.
You probably want to use a subshell `$()`, `if` statements, 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 lab03_file_sizes file_sizes.sh
```

before **Monday 19 June 12:00 (midday)** (2023-06-19 12:00:00) to obtain the marks for this lab exercise.

SOLUTION:

Sample solution for `file_sizes.sh`

```
#! /usr/bin/env dash

# `*` finds non-hidden file system objects
# `.[!.]*` and `..?*` finds hidden file system objects except for `.` and `..`

for file in * .[!.]* ..?*; do
  if [ ! -f "$file" ]; then
    continue
  fi

  lines=$(wc -l < "${file}")

  if [ "${lines}" -lt 10 ]; then
    small_files="${small_files} ${file}"
  elif [ "${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
```

EXERCISE:

Scraping JSON APIs with the shell

Write a POSIX-compatible shell script, `scraping_courses.sh`, which prints a list of UNSW courses with the given prefix by extracting them from the UNSW handbook webpages.

In 2019 UNSW changed to much prettier format but also added an API that scripts can extract information from.
(It's probably not really meant for our scripts, but they haven't blocked us yet)

For this exercise we'll use the 2019-2023 handbook pages via the API

For example:

```
$ ./scraping_courses.sh 2021 OPTM
OPTM2133 The Clinical Environment
OPTM2190 Introduction to Clinical Optometry
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 and 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
OPTM7107 Ocular Therapy 1
OPTM7108 Research Skills in Optometry
OPTM7115 Visual Neuroscience
OPTM7117 Ocular Therapy 2
OPTM7203 Behavioural Optometry 2
OPTM7205 Specialty Contact Lens Studies
OPTM7208 Research Skills in Optometry
OPTM7213 Ocular Therapy
OPTM7218 Research Project
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
OPTM7611 Introduction to Myopia
OPTM7612 Myopia Management
OPTM7621 Clinical Myopia Management
OPTM8511 Clinical Paediatrics, Low Vision and Colour Vision
OPTM8512 Clinical Optometry 5A
OPTM8513 Clinical Ocular Therapy 5A
OPTM8514 Optometry Research Project
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 2020 MATH | wc
    125     601    5029
$ ./scraping_courses.sh 2019 COMP | grep -F "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
COMP6447 System and Software Security Assessment
COMP6452 Software Architecture for Blockchain Applications
COMP9044 Software Construction: Techniques and Tools
COMP9322 Software Service Design and Engineering
COMP9323 Software as a Service Project
COMP9434 Robotic Software Architecture
$ ./scraping_courses.sh 2023 MINE | grep -F "Rock"
MINE3630 Rock Breakage
MINE5010 Fundamentals of Rock Behaviour for Underground Mining
MINE5030 Mining Excavations in Rock
```

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

```
$ ./scraping_courses.sh
Usage: ./scraping_courses.sh <year> <course-prefix>
```

Also get your script to print this error message if its first argument isn't in the allowable range:

```
$ ./scraping_courses.sh 2000 COMP
./scraping_courses.sh: argument 1 must be an integer between 2019 and 2023
```

Your script must access the handbook API and extract the information from the returned JSON data.

HINT:

Make the first line of your shell-script `#!/bin/dash`

The program curl can be used to access the API, by running it as **curl -sL *<url>***
or
The program wget can be used to access the API, by running it as **wget -qO- *<url>***

This task is tricky just using line-based tools such as, `sort`, and `uniq`.

An extra tool called jq can make the task much easier.

But you'll need to figure out how to use **`jq`** - understanding new tools is a skill worth developing!

The UNSW handbook uses separate pages for undergraduate and postgraduate courses.
These two web pages would need to be downloaded (eg, for all COMP courses in 2021):

```
https://www.handbook.unsw.edu.au/api/content/render/false/query/+unsw_psubject.implementationYear:2021%20
```

and

```
https://www.handbook.unsw.edu.au/api/content/render/false/query/+unsw_psubject.implementationYear:2021%20
```

Any sequence of whitespace characters in the *course title* should be replaced with a single space.

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

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 lab03_scraping_courses scraping_courses.sh
```

before **Monday 19 June 12:00 (midday)** (2023-06-19 12:00:00) to obtain the marks for this lab exercise.

SOLUTION:

Sample solution for `scraping_courses.sh`

```
#! /usr/bin/env dash

min_year=2019
max_year=2023
program_name="$0"

case $# in
  2)
    YEAR=$1
    CODE=$2
    ;;
  *)
    echo "Usage: ${program_name} <year> <course-prefix>" >& 2
    exit 1
esac

if [ "${YEAR}" -ge "${min_year}" ] 2> /dev/null && [ "${YEAR}" -le "${max_year}" ] 2> /dev/null;
then
  :
else
  echo "${program_name}: argument 1 must be an integer between ${min_year} and ${max_year}" >& 2
  exit 1
fi

# `env printf` uses the binary `printf` not the shell builtin
# this is because different shells have different builtin `printf` commands
# dash `printf` doesn't support `\u`
SPACES=$(env printf "%b"
"\u00A0\u1680\u2000\u2001\u2002\u2003\u2004\u2005\u2006\u2007\u2008\u2009\u200A\u2028\u2029\u202F\u2

(
  curl -Ls
"https://www.handbook.unsw.edu.au/api/content/render/false/query/+unsw_psubject.implementationYear:$
  curl -Ls
"https://www.handbook.unsw.edu.au/api/content/render/false/query/+unsw_psubject.implementationYear:$
) |
# `.["contentlets"]` will extract the object with the key `contentlets`
# `[]` will extract all the elements of the array
# `|` will pipe the output of the previous command to the next command
# with the preceding `[]` this will pipe each element of the array to the next command one at a time
# `.code + " " + .title` creates a string with the course code and title separated by a space
# `gsub("[[:space:]${SPACES}]"; " ")` replaces all whitespace characters with spaces
# the `gsub` must be done here and not later with `sed`
# as once we finish the `jq` command we can no longer distinguish between
# newlines within the course title and newlines between courses
# [:space:] (within the C locale) only matches the ASCII whitespace characters
# ${SPACES} as defined above matches any Unicode whitespace characters
jq -r '
  .["contentlets"][] |
  (
    .code + " " + .title |
    gsub("[[:space:]'"${SPACES}"']"; " ")
  )
' |
tr -s ' ' | # squeeze whitespace (all whitespace will just be spaces after the previous command)
sed 's/ $//' | # remove trailing whitespace (there can only be one trailing space after the previous
command)
# remove duplicate lines
sort | uniq
```

CHALLENGE EXERCISE:

Scraping Badly Designed JSON APIs with the shell

Write a POSIX-compatible shell script, `advanced_scraping_courses.sh` , which works exactly the same as `scraping_courses.sh` but can also retrieve data from the legacy handbook.

In 2005-2018 the handbook uses a different API so you will need to write a new (and more complicated) `jq` command.

The legacy handbook API returns a lot of extra information.
We only want undergraduate and postgraduate courses.

Same as the non-challenge exercise, your script should print a suitable error message when necessary.

HINT:

Make the first line of your shell-script `#!/bin/dash`

Due to an issue with the legacy handbook API, you will need to use the a slightly different `curl` command.
curl --cipher 'DEFAULT:!DH' -sL *<url>*
(It has had a broken SSL configuration for over a year now, so we need to disable SSL verification)
or
You can still use the same `wget` command
wget -qO- *<url>*

You can find the legacy handbook API here (eg, for all courses (and more) in 2005):
`https://legacy.handbook.unsw.edu.au/assets/json/search/2005data.json`

If a course doesn't have a *course code* listed perhaps there is another way to get this information

If a course doesn't have a *course title* listed it should simply be left blank.

Any sequence of whitespace characters in the *course title* should be replaced with a single space.

Remember we only want to print the *course code* and *course title* of undergraduate and postgraduate courses.

Opening the API URL in firefox will give a nice interface for you to examine the returned data (other browsers might provide similar functionality with extensions).

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

```
$ 2041 autotest advanced_scraping_courses
```

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

```
$ give cs2041 lab03_advanced_scraping_courses advanced_scraping_courses.sh
```

before **Monday 19 June 12:00 (midday)** (2023-06-19 12:00:00) to obtain the marks for this lab exercise.

SOLUTION:

Sample solution for `advanced_scraping_courses.sh`

```
#! /usr/bin/env dash

min_year=2005
switch_year=2019
max_year=2023
program_name="$0"

case $# in
  2)
    YEAR=$1
    CODE=$2
    ;;
  *)
    echo "Usage: ${program_name} <year> <course-prefix>" >& 2
    exit 1
esac

SPACES=$(env printf "%b"
"\u00A0\u1680\u2000\u2001\u2002\u2003\u2004\u2005\u2006\u2007\u2008\u2009\u200A\u2028\u2029\u202F\u2

if [ "$YEAR" -ge "${min_year}" ] 2> /dev/null && [ "$YEAR" -lt "${switch_year}" ] 2> /dev/null;
then
  curl --cipher 'DEFAULT:!DH' -Ls
"https://legacy.handbook.unsw.edu.au/assets/json/search/${YEAR}data.json" |
  # This API gives no top level object, so we immediately pipe the array `[]` to the next command
  # `slect()`` will filter the array based on the condition provided
  # as we are only looking for undergraduate and postgraduate courses, we filter based on the `type`
and `career` fields
  # this stops us finding `research` courses, as well as `program` and `plan` types
  # unlinke the new API we can't filter directly in the query,
  # so we have to do it with another `select()`
  # (OK we could just pass the results to grep, but that's not as fun)
  # The second `select()` filters based on the course code
  # if we dont have a `code` field, we use the `filename` field and remove the `.html` extension
  # because the filename field always exists, while the code field doesn't for some reason
  # `if` allows us to select a value based on a condition
  # in this case the condition is `has("code")` which checks if a field exists called `code`
  # `test()` is a regex match function to return true if the string matches the regex
  # finally, just as in the previous activity, we create a string with the course code and title
  # but we need to use a `if` again so that if the `code` field doesn't exist we use the `filename`
field
  # and if the `shortdescription` field doesn't exist we use an empty string
  jq -r '
    .[] |
    select(
      .type == "courses"
      and
      (
        .career == "undergraduate"
        or
        .career == "postgraduate"
      )
    ) |
    select(
      (
        if
          has("code")
        then
          .code
        else
          .filename |
          gsub("\\.html$"; "")
        end
      ) |
      test('"\"^${CODE}\""')
    ) |
    (
      (
        if
          has("code")
        then
```

```
          .code
        else
          .filename |
          gsub("\\.html$"; "")
        end
      )
      + " " +
      (
        if
          has("shortdescription")
        then
          .shortdescription
        else
          ""
        end
      ) |
      gsub("[[:space:]'"${SPACES}"']"; " ")
    )
  ' |
  tr -s ' ' |
  sed 's/ $//' |
  sort | uniq
elif [ "$YEAR" -ge "${switch_year}" ] 2> /dev/null && [ "$YEAR" -le "${max_year}" ] 2> /dev/null;
then
  # This is the same as the previous activity
  (
    curl -Ls
"https://www.handbook.unsw.edu.au/api/content/render/false/query/+unsw_psubject.implementationYear:$
    curl -Ls
"https://www.handbook.unsw.edu.au/api/content/render/false/query/+unsw_psubject.implementationYear:$
  ) |
  jq -r '
    .["contentlets"][] |
    (
      .code + " " + .title |
      gsub("[[:space:]'"${SPACES}"']"; " ")
    )
  ' |
  tr -s ' ' |
  sed 's/ $//' |
  sort | uniq
else
  echo "${program_name}: argument 1 must be an integer between ${min_year} and ${max_year}" >& 2
  exit 1
fi
```

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 **Week 4 Monday 12:00:00 (midday)** to submit your work.

You cannot obtain marks by e-mailing your code 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, using test cases different to those `autotest` runs for you. (Hint: do your own testing as well as running `autotest` .)

After automarking is run by the lecturer you can view your results here. The resulting mark will also be available 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
```