CITS4407 Open Source Tools and Scripting
Semester 1, 2021
Week 5 workshop – Text and regular expressions

Before starting this workshop, make sure you've reviewed the recommended reading for weeks 1–4, and completed the lab sheets for weeks 2–4.

0. Revision

In your reading from previous weeks, you should have read about the `sort`, `uniq`, `wc`, `head` and `tail` commands. (If not – refer back to chapter 6, "Redirection", of *Shotts*.)

Answer the following questions. It is suggested you try these problems on your own and discuss with a partner – then compare your answers with the rest of the class.

Counting

```
$ echo a b c
```

What is a pipeline we can add to the end of this command (i.e., the pipe character, "`|`", plus further commands) which will display how many words the `echo` command has printed? How does it work?

One solution:

```
$ echo a b c | wc -w
```

The `wc` command counts words, lines, or characters on standard input; `wc -w` will count the number of words.

Duplicates

Executing the following

```
ls /usr/bin /bin
```

will list the contents of the `/usr/bin` and `/bin` directories, which contain executable programs for external commands.

Traditionally on a Unix-like system, `/bin` contains just a subset of the typical Unix commands – it contains only those commands needed by the system for booting and potentially repairing the system. Commands in `/usr/bin/`, on the other hand, are not needed for booting or repair, and thus could be kept on a separate disk and loaded later.

Some programs appear only in `/bin`, some only in `/usr/bin`, and some appear in both directories.

What is a pipeline we can add to the command shown, which will print the programs found in both directories? How does it work?

One solution:

```
ls /usr/bin /bin | sort | uniq -d
```

Here, `uniq -d` is used to print duplicate lines in its input. It requires the input to be sorted, however, so we filter the output of the `ls` command through `sort`.

Challenge exercise: How can we print the files that are in `/bin`, but not `/usr/bin`?

1. Shell expansions

It is suggested you try these problems on your own and discuss with a partner – then compare your answers with the rest of the class.

Hidden files

Try running the following in your Bash terminal:

```
$ touch .hidden-file
$ ls -a
$ echo *
```

What do you expect to see, and why? Can you suggest a filename expansion that you can put after "`echo`", that will print `.hidden-file` (and possibly other dot-files in the working directory)?

One solution:

```
$ echo .*
```

Arithmetic expansion

Look at the following commands:

```
$ touch file1
$ myvar="INSERT SOMETHING HERE"
$ echo file$((4 - myvar))
```

If you run these, you will just get an error. But assigning the correct value to `$myvar` will result in `file1` being printed by the `echo` command. What value do you need to insert, and why does this work?

One solution:

```
$ myvar="3"
```

Command substitution

Run the following commands:

```
$ echo "some content" > file1
$ echo "$(cat file1)"
```

From previous weeks, you should know what the first line is doing – it is writing the output of the `echo` command to `file1`.

The next line is performing a type of expansion called *command substitution*. The dollar sign tells bash that the string between the parentheses should be interpreted as commands to run, and the dollar and parentheses should be *replaced* by the output of those commands.

```
$ echo "some content" > file1
$ echo "$(cat file1)"
```

Look at the following command:

```
$ echo "The location of the cp command is: XXX"
```

Suppose we wish to replace the `XXX` with the location of the `cp` command – what should we put there, and why?

One solution:

```
$ echo "The location of the cp command is: $(which cp)"
```

The `which` command is used here to show the location on the filesystem of the `cp` command.

Quoting

Which of the following will print a single asterisk to the screen? Why?

a. `echo *`
b. `echo "*"`
c. `echo '*'`

Option (c) – the others will print the names of all the (non-hidden) files in the working directory.

This is because putting a string in single quotes suppresses all shell expansions – refer to *Shotts* chapter 7, "Seeing the World as the Shell Sees It", under "Quoting".

2. Scripts

If you haven't tried the exercises in *Shotts* chapter 20 ("Text processing"), attempt those first before doing this section.

It is suggested you try these problems on your own and then discuss with a partner – then compare your answers with the rest of the class.

Running scripts

Try running the following commands:

```
$ echo "echo hello" > my-script.sh
$ ./my-script.sh
```

You should see an error message. What is the problem here, and how do we fix it?

The file `my-script.sh` does not have *executable* permissions set, so Bash refuses to run it as a script.

We can fix this by using the `chmod` command – for instance, by executing:

```
$ chmod u+x my-script.sh
```

before running the script.

Manipulating text

Clone the Git repository at `<https://github.com/cits4407/workshop04>` onto your computer. The file contained in it stores data on enrolments at Australian universities, and is in *tab-separated* format – each line contains several "fields", and the fields are separated by tab characters. Files like this are often given the extension `.tsv`.

(**Challenge exercise:** Can you find a way of downloading the `.tsv` file without using Git? Investigate the `wget` and `curl` commands, and take a look at the Github repository in your browser to see what you can access via the browser.)

Write a script that uses the `sort` command to print the line of the file which contains a record of the university with the highest total enrolment. (Hint: you might want to use the `head` command, as well – check the documentation for it.)

Then extend your script to also print the lines for the universities with the highest and lowest local enrolment.

Finally, add pipelines in your script, using the `cut` command to print just the *names* of the universities, not the whole line.

Downloading the `.tsv` file:

If you go to https://github.com/cits4407/workshop04 on GitHub in your browser, you'll see you can click on the `australian-universities.tsv` file, to bring up information about the file, and a Web version of its content (nicely displayed in a table).

Clicking on the button marked "`raw`" gives us the "raw" file content – the content exactly as it is found in the text file.

And copying and pasting the URL from our browser into a terminal allows us to run a command like the following:

```
$ wget https://raw.githubusercontent.com/cits4407/workshop04/master/
  ↪ australian-universities.tsv
```

which will download the `.tsv` file to our computer, without having to use `git`.

Print the highest total enrolment:

The following is one solution:

```
tail -n +2 australian-universities.tsv  | sort --field-separator $'\t'
   ↪ -n -k 4 | tail -n 1
```

First, we use `tail -n +2` to remove the header line – we don't want that in our result.

Next we sort the lines by the 4th field (the "total" field), using `sort --field-separator $'\t' -n -k 4`. The `-n` means "sort numerically", the "`-k 4`" means "sort by the 4th column".

By default, `sort` assumes that columns are separated by whitespace. But this would mean for instance that the string "`Monash University`" is incorrectly interpreted as being two columns.

So we use the `--field-separator` option for `sort`, and tell it to use the tab character as a separator. The dollar sign in front of

```
$'\t'
```

tells Bash to interpret *escape sequences* – ways of representing "invisible" or "control" characters like tab and return ("newline").

`$\t` expands to a tab character, and `$\n` to a newline character.

Check out section 3.1.2 "ANSI-C Quoting" of the Bash manual for a full list of these escape sequences.

Finally, `tail -n 1` says to only print the very last line of the output.

So if we run this command pipeline, we should get the following output:

```
$ tail -n +2 australian-universities.tsv | sort --field-separator $'\t'
   ↪  -n -k 4 | tail -n 1
Monash University   42339   22140   64479
```

A challenge question: how might we double-check the answer to this question? What software can we use which will also allow us to quickly see the university with highest total enrolment?

Print the highest and lowest local enrolment:

For the highest local enrolment, we use the same pipeline as before, but sorting on field 2:

```
tail -n +2 australian-universities.tsv  | sort --field-separator $'\t'
    ↪ -n -k 2 | tail -n 1
```

So we now give the arguments `-k 2` to `sort`.

For the lowest local enrolment, we put the pipeline `head -n 1` at the end, rather than `tail -n 1` – this gives us the *first* line:

```
tail -n +2 australian-universities.tsv  | sort --field-separator $'\t'
-n -k 2 | head -n 1
```

Print just the names:

We can put our three commands in a script like the following:

```
#!/bin/bash

u_highest_total="$(tail -n +2 australian-universities.tsv  | sort --
    ↪ field-separator $'\t' -n -k 4 | tail -n 1 | cut -d $'\t' -f 1)"
u_highest_local="$(tail -n +2 australian-universities.tsv  | sort --
    ↪ field-separator $'\t' -n -k 2 | tail -n 1 | cut -d $'\t' -f 1)"
u_lowest_local="$(tail -n +2 australian-universities.tsv  | sort --
    ↪ field-separator $'\t' -n -k 2 | head -n 1 | cut -d $'\t' -f 1)"

echo "University with the highest total enrolment is: $u_highest_total"
echo "With the highest local enrolment is: $u_highest_local"
echo "With the lowest local enrolment is: $u_lowest_local"
```

We have added at the end of all the pipelines the command invocation `cut -d $'\t' -f 1`.

The `cut` command "slices" out columns from standard input. The `-d $'\t'` says to use the tab character "`\t`" as a delimeter, and the `-f 1` means to print only the *first* column.