Week 02 Tutorial Sample Answers

1. 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
```

> DANGER:
>
> This data is fictitious.
> **Do not ring these phone numbers.**

The data is currently sorted in phone number order.
Can we use the *sort* filter to re-arrange the data into “telephone-book” order?
(alphabetically by last name)
If not, how would we need to change the file in order to achieve this?

> ANSWER:
>
> We need to sort based on the family name field, but this occurs in different positions on each line of the file, depending how many initials the person has.
>
> > Thanks JAS.
>
> To achieve this, we need to make sure that the family name occurs in the same "field" on each line.
> One possibility:
> make it the first field in each line:
>
> ```
> Keller G        Newtown      9381-1234
> Wilson W H      Kingsford    9621-1234
> ...
> ```
>
> Then sort by field 1.
>
> Another possibility:
> make sure that the initials form a single field:
>
> ```
> G. Keller       Newtown      9381-1234
> W.H. Wilson     Kingsford    9621-1234
> ...
> ```
>
> Then sort by field 2.

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

```
root:ZHolHAHZw8As2:0:0:root:/root:/bin/dash
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/zsh
cs9311:fIVLdSXYoVFaI:9311:9311:COMP9311 Material:/home/cs9311:/bin/bash
cs9314:nTn.JwDgZE1Hs:9314:9314:COMP9314 Material:/home/cs9314:/bin/fish
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 class accounts have a username that starts with either "cs", "se", "bi" or "en", followed by four digit)

c. Display the username of everyone whose shell is `/bin/bash`

d. Create a tab-separated file `passwords.txt` containing only the username and password of each user

ANSWER:

a.
```
$ head -3 /etc/passwd
```

b.
```
$ grep -E '^(cs|se|bi|en)[0-9]{4}:' /etc/passwd
```

c.
```
$ grep -E ':/bin/bash$' /etc/passwd | cut -d':' -f1
```

d.
```
$ cut -d':' -f1,2 /etc/passwd | tr ':' '\t' > passwords.txt
```

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

```
$ tr -cs 'a-zA-Z0-9' '\n' < someFile
```

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

ANSWER:

- `-c` is complement:
  It replaces everything **not** in string 1 with string 2
- `-s` is 'squeeze repeated characters':
  It replaces any duplicate newlines with just one.
- the string `'a-zA-Z0-9'` is shorthand for the string of all alphanumeric characters:

  ```
  abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789
  ```

  `[:alpha:]` is also shorthand for the string of all alphanumeric characters.

4. 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 '[:lower:]' '[:upper:]' | sort | uniq -c
```

ANSWER:

a. This pipeline deletes all of the spaces between words,
thus compressing each line into a single long word,
which means the number of words is the same as the number of lines (2).

b. This splits the input up so there is one word of input on each line of output;
counting the number of output lines thus also counts the total number of words in the text (11).

c. This splits the input up so there is one word of input on each line of output;
then normalises them (by mapping all words to all upper-case),
then counts the number of occurrences of each distinct word.
The output looks like:

```
4 BIG
1 BUT
2 IS
1 NOT
1 SO
2 THIS
```

5. Consider a file containing (fake) zIDs and marks in COMP1511:

```
4279700|61
4212240|59
4234024|57
4286024|50
4270657|75
4227010|52
4299716|84
4236088|74
4245033|87
4222098|46
4228842|85
4209182|96
4276270|61
4224421|72
4207416|76
```

and another file containing (fake) zIDs and marks in COMP2041:

```
4200549|92
4283960|77
4203704|48
4261741|43
4224421|67
4223809|75
4276270|80
4279700|68
4233865|61
4207416|56
4209669|71
4209182|70
4213591|49
4236221|53
4201259|91
```

a. Can the files be used as-is with the `join` command?
If not, what needs to be changed?

b. Write a `join` command which prints the marks in COMP1511 and COMP2041 of everyone who did *both* courses.

c. Write another `join` command which prints the marks in COMP1511 and COMP2041 of everyone, across both files,
With `--` in the case where a student didn't do a particular subject

d. Write a shell pipeline which prints the marks in COMP1511 and COMP2041 of everyone who did *both* courses,
sorted by their COMP1511 mark in *ascending* order,
then by their COMP2041 mark in *descending* order.

ANSWER:

a. No, they need to be sorted by a common key (in this case, zID).

```
sort -t'|' -k1,1 comp1511-marks.psv > comp1511-marks-sorted.psv
sort -t'|' -k1,1 comp2041-marks.psv > comp2041-marks-sorted.psv
```

b. Assuming that the files are sorted into *comp1511-marks-sorted.psv* and *comp2041-marks-sorted.psv*:

```
join -t'|' comp1511-marks-sorted.psv comp2041-marks-sorted.psv
```

c. Assuming that the files are sorted into *comp1511-marks-sorted.psv* and *comp2041-marks-sorted.psv*:

```
join -t'|' -a1 -a2 -o auto -e'--' comp1511-marks-sorted.psv comp2041-marks-sorted.psv
```

The `-o auto` option is required in this case so that `join` can calculate how many fields are required.

d. Assuming that the files are sorted into *comp1511-marks-sorted.psv* and *comp2041-marks-sorted.psv*:

```
join -t'|' comp1511-marks-sorted.psv comp2041-marks-sorted.psv | sort -t'|' -k2,2 -k3,3r
```

6. Consider a file containing tab-separated benchmarking results for 20 programs, in three different benchmarks, all measured in seconds.

```
program1    08  03  05
program2    14  03  05
program3    17  08  10
program4    15  11  05
program5    16  10  24
program6    15  09  17
program7    15  06  10
program8    17  10  17
program9    12  07  08
program10   09  04  16
program11   11  03  24
program12   16  11  20
program13   16  08  17
program14   08  07  06
program15   06  06  05
program16   12  05  08
program17   09  05  10
program18   06  06  06
program19   14  09  22
program20   16  04  24
```

a. Write a `sort` command which sorts by the results in the second benchmark, then by the results in the first benchmark.

b. Write a `sort` command which sorts by the results in the third benchmark, then by the program number.

c. Write a `sed` command which removes the leading zeroes from the benchmark times.

d. Write a `sed` command which removes the benchmark results from `program2` through `program13` .

ANSWER:

a.
```
$ sort benchmarks -k3,3 -k2,2
```

b.
```
$ sort benchmarks -k4,4 -k1.8,1n
```

c.
```
$ sed -Ee 's/\t0/\t/g' benchmarks
```

d.
```
$ sed -Ee '/^program2\b/,/^program13\b/d' benchmarks
```