Monash University
Faculty of Information Technology
2nd Semester 2018

FIT2014 Lab 0 Introductory Linux

SOME SOLUTIONS

This file gives solutions to the exercises in Lab 0. These are just sample solutions. In each case, there are many other ways to do the specified task.

Some of the later exercises on frequency counts require some tidying-up of the files produced before passing them to the next stage; this document does not discuss such tidying-up in great detail.

1 Input, output, redirection, pipes

Exercise: Suggest two ways to concatenate two files, placing the combination into a single new file.

The first way puts the concatenation into a new file, *file3*.

```
$ cat file1 file2 > file3
```

The second way appends the second file to the first file, so that *file2* is unchanged but *file1* becomes the concatenation of the two files.

```
$ cat file2 >> file1
```

A variation on this that puts the concatenation in a completely new file is:

```
$ cp file1 file3
$ cat file2 >> file3
```

2 The character translator `tr`

Exercise: How would you use `tr` to concatenate all the lines of a file so that the file just contains one single long line (with all line breaks removed)?

```
$ tr '\n' ' ' < file
```

Here, the first string consists just of the newline character, and the second just has the space character, so every newline is replaced by a space.

Try piping the output to `wc`. How many lines does the output have? How do you explain the answer `wc` gives to this question?

3 The stream editor `sed`

Exercise: How would you use `sed` to remove all characters that are not letters?

```
$ sed 's/[^a-zA-Z]//g' file
```

If you wanted to keep spaces and newlines as well, you could do:

```
$ sed 's/[^a-zA-Z \n]//g' file
```

Exercise: Suppose you have a file of prices in dollars and cents, all under $100. Use `sed` to remove all cents from the prices. Then consider how, instead, to round all prices to the nearest dollar.

Truncation:

```
$ sed 's/\.[0-9][0-9]//g' file
```

If you want to still show the cents for the truncated prices, even though they're all zero, then instead you could use

```
$ sed 's/\.[0-9][0-9]/.00/g' file
```

Notice the backslash in the pattern in each of these two examples. It ensures that it is indeed the full stop that is matched. Without the backslash, the full stop in a `sed` pattern matches any character except newline. The backslash is not necessary in the replacement string, as it's not a pattern, and full stop has its ordinary meaning.

The global flag, `g`, can be dropped if your file only has one price per line.

Rounding:

This is tricky! The only way I know is to treat the round-down and round-up cases separately. You can combine the two using pipes. The most serious problem is that, in the round-up case, digits may need to be carried, which pattern replacement in `sed` is not well suited to.

Rounding down:

```
$ sed 's/\.[0-4][0-9]/.00/g' file
```

Rounding up:

```
$ sed 's/\.[5-9][0-9]/.RR/' file | sed '/\.RR/ s/9\./CZ./;/\.RR/ s/8\./N9./;/\.RR/ s/7\./N8./;/\.RR/
s/6\./N7./;/\.RR/ s/5\./N6./;/\.RR/ s/4\./N5./;/\.RR/ s/3\./N4./;/\.RR/ s/2\./N3./;/\.RR/
s/1\./N2./;/\.RR/ s/0\./N1./;/\.RR/ s/9C/X/;/\.RR/ s/8C/9/;/\.RR/ s/7C/8/;/\.RR/ s/6C/7/;/\.RR/
s/5C/6/;/\.RR/ s/4C/5/;/\.RR/ s/3C/4/;/\.RR/ s/2C/3/;/\.RR/ s/1C/2/;/\.RR/ s/0C/1/;/\.RR/
s/ C/1/;/\.RR/ s/^C/1/;/\.RR/ s/N//;/\.RR/ s/Z/0/;/\.RR/ s/X/10/;s/\.RR/.00/;'
```

The semicolons separate successive `sed` scripts, and ensure that the scripts are all applied in the given order.

Interpretation of some of the symbols used during the working:

RR	used to indicate that this dollar amount is to be rounded up. Gets replaced by 00 at the end. In the meantime, it's used to select those amounts that are to be rounded up, and ensure that the rounding, carrying etc. is only done to them.
Z	Zero digit. To be replaced near the end. Needed to ensure that it isn't replaced by 1 by some later substitution.
X	Ten. To be replaced by 10 near the end.
N	No carry to the left here.
C	Carry 1 to the left here.

Exercise: Use `sed` to insert a space between every pair of adjacent letters in a file.

```
$ sed 's/\([a-zA-Z]\)/\1 /g' file
```

The following does *not* do this, as it misses every second pair of letters. But it could be useful in working with digraphs later.

```
$ sed 's/\([a-zA-Z]\)\([a-zA-Z]\)/\1 \2/g' file
```

4 Frequency count

Exercises:

1. From your input file, derive one with the same words, in the same order, but with each word on a separate line.

   ```
   $ sed 's/ */\n/g' file
   ```

   If that doesn't work (in particular, if it replaces the space by the letter '`n`' instead of a newline, try

   ```
   $ sed 's/ */\
   /g' file
   ```

   . . . where you actually press the Return key at the end of the first line (*immediately* after the backslash), and then type the second line as given (pressing Return at the end, of course).

   If you want to replace every stretch of "white space" — i.e., spaces *and tabs* — by a newline, try the following variation:

   ```
   $ sed 's/[ \t]+/\
   /g' file
   ```

   It may make sense to use `sed` to remove punctuation first, provided the punctuation isn't part of the word. So you'd like to keep apostrophes if they are part of the word (as in, `doesn't` . . . and what about `fishin'` or `'fraid`?!), but not if they're part of quotation marks . . . Could get fiddly! Don't be too fussy about this.

2. From this file, find one in which each word appears only once, and is accompanied by its frequency (i.e., the number of times it occurs in the file).

   ```
   $ sort fileFromAbove | uniq -c
   ```

3. Sort the file of word frequencies in order of decreasing frequency.

   Pipe the result of the above frequency count into `sort` with appropriate option(s) for `sort` to ensure reverse numerical order.

4. Now do a frequency count of *letters* in the file. You should give the frequency count in two separate files: one with the letters in alphabetical order, the other with the letters ranked by frequency.

   Use earlier methods to put spaces between all the letters, then replace the spaces by newlines, then use `sort` and `uniq -c` as above.

5. A *digraph* is a pair of consecutive letters. Do a frequency count of all *digraphs* in a file. (Overlapping digraphs are still counted separately. For example, if the file just consists of the single word `dodo`, then we have three digraphs, namely `do`, `od`, and `do`, so the frequency count should show that `do` has frequency 2 and `od` has frequency 1.)

   The tricky part here is splitting the file up into digraphs without losing overlapping digraphs. You can use . . .

   ```
   $ sed 's/\([a-zA-Z][a-zA-Z]\)/\1 /g' file
   ```

   . . . for half the digraphs, and . . .

   ```
   $ sed 's/\([a-zA-Z]\)\([a-zA-Z]\)/\1 \2/g' file
   ```

   . . . for the other half. Then concatenate the two outputs. Then do frequency count.

   This exercise is somewhat simpler if you remove all spaces first. You may also need to remove all non-digraphs (including isolated single letters) from the file of digraphs you get, before doing the frequency count.