Regular Expressions Exercises (1)

Regular Expressions Overview and Definition

Regular expressions (regex) are a powerful tool used in programming and shell scripting for searching, matching, and manipulating text based on specific patterns. They enable developers and system administrators to perform complex text processing tasks efficiently. Regular expressions can be used in various programming languages and command-line tools, and utilities like `grep`, `sed`, and `awk` in the Unix/Linux shell.

For this exercise you'll want to refer often to a manual for `grep`. You can access one on the commandline by invoking `man grep`. You've already tackled some problems involving regular expressions in a previous exercise. Here are some more advanced questions that will require you to understand more about `grep`, its options, and how regular expression syntax works.

1. Study the documentation for the `w` option.

    ```
    -w, --word-regexp
                 Select only those lines containing matches that
                 The  test  is  that  the  matching  substring m
                 beginning of the  line,  or  preceded  by  a  n
                 character.   Similarly, it must be either at th
                 followed by a  non-word  constituent  characte
    ```

```
                characters  are  letters,  digits, and the unde
                has no effect if -x is also specified.
```

Contrive a file such that `grep PATTERN FILE` returns two different lines but `grep -w PATTERN FILE` returns only one line.

grep_practise.text:

```

silent
lent
```

```
grep 'lent' grep_practise.txt

OUTPUT:

silent
lent


grep -w 'lent' grep_practise.txt


OUTPUT:

lent
```

2. You'll have seen beforehand that you can count the results of a search with `grep PATTERN FILE | wc -l`. However, `grep` also has a `c` option which counts matches. Can you find the situation where the `wc -l` approach and the `c` approach produce different results? Can you explain why?

`grep -c` counts the number of lines that match the pattern passed to grep. `wc -l` will count the number of lines that contain the pattern

REVISIT PERHAPS??????

3. Some words have different spelling between British English and American English. For example, 'encyclopaedia' is valid in British English but not American. Can you write a regular expression that would match both of these words, but nothing else? How about matching both 'color' (American) and 'colour' (British)?

The `?` operator would look for either 0 or 1 occurences of the preceeding element, i.e. the element behind it.

Tp use `?` you need to tell grep to look for **extended regular expressions** using the `-E` flag

Example text file:

```
encyclopaedia
silent
color
lent
colour
encyclopedia
```

```
grep -E 'colou?r' grep_practise.txt

OUTPUT:
color
```

```
colour


grep -E 'encyclopa?edia' grep_practise.txt

OUTPUT:
encyclopaedia
encyclopedia
```

4. UK postcodes follow a general schema of two letters followed by one number, followed by an optional space, then another number, followed by two more letters. Can you write a regular expression that would match such sequences?

Example input file:

```
TQ8 5SQ
TQ95UJ
TQ9 5UJ
TQQ 5UJ
SL94EX
9AA 4EX
```

```
grep -Ei '^[A-Z]{2}\d[[:space:]]?\d[a-z]{2}' regex_postcodes.tx
```

- `i` makes grep case insensitive
- `{2}` checks for 2 instances of the preceeding character, in this case 2 alphabnetical characters

5. In practice, the above is a simplified version of the system, and a better UK postcode validator regex is known to be `^(([A-Z]{1,2}[0-9][A-Z0-9]?|ASCN|STHL|TDCU|BBND|[BFS]IQQ|PCRN|TKCA) ?[0-9][A-Z]{2}|BFPO ?[0-9]{1,4}|(KY[0-9]|MSR|VG|AI)[ -]?[0-9]{4}|[A-Z]{2} ?[0-9]{2}|GE ?CX|GIR ?0A{2}|SAN ?TA1)$`. Try

breaking apart this monster to understand what is being tested, and find an example that would match the schema described for the fourth question but fail to match this expression.