CSE 390, Spring 2010
Homework 7: Regular Expressions
Due Tuesday, May 18, 2010, 12:30 PM

This assignment focuses on using regular expressions and related commands such as `sed` and `egrep`. Turn in a file named `homework7.txt` using the Homework section of the course web site.

(This week's homework is more 1-line shell commands. We'll go back to more complex shell scripts next assignment.)

Task 1: Bash Shell Commands with `egrep`:

For each item below, **determine a single `bash` shell statement that will perform the operation(s) requested**. Each solution must be a one-line shell statement, but you may use input/output redirection operators such as >, <, and `|`. Write these commands into your `homework7.txt` file, one per line. In your file, **write the command** that will perform the task described for each numbered item; don't write the actual output that such a command would produce.

> *(Self-Discovery)* The command `locate` searches an index database of all filenames on an entire Unix/Linux system to quickly find the file(s) that have a given name. Given proper parameters, locate can be given a regular expression for the file name pattern to search for. (See its `man` page.)
>
> *(Self-Discovery)* A file `words` exists on most Unix/Linux systems and contains a large list of English words, one per line. We will use this file as an input test file for the shell commands on this part of the assignment. But first, you have to discover where the file is! Simply typing `locate words` will output several file names, one of which is the right one...

1. Write a shell command that runs `locate` using a regular expression to output only a single line: the location of the words dictionary file. Hint: Search for files named exactly "words", not a longer filename containing "words" such as "keywords.xsl". You may assume that there is only one words file, and that it is not in the root directory; in other words, that its name is preceded by a `/` . You can rely on the fact that the line will end with `words` .

   For the rest of the problems, write a command that uses `grep -E` or `egrep` to list the given words from the `words` file you found, one per line. Use regular expressions as appropriate. Use a single `grep` call per command.

2. All words that contain a 'z', followed by any single character, followed by an 'x'.
   (On `attu` there are 22 of them, from "azox" to "zaxes").
3. All words that contain the text "banana" or "mango".
   (On `attu` there are 21 of them, from "banana" to "mangour").
4. All words that contain a string of at least 5 vowels (a, e, i, o, or u) in a row.
   (On `attu` there are 9 of them, from "cadiueio" to "queueing").
5. All words that end with "never". (The word must *end* with "never", not just contain the substring "never".)
   (On `attu` there are 4 of them, from "minever" to "whenever").
6. All words that are exactly 25 letters long, in reverse ABC order.
   (On `attu` there are 8 of them, from "superincomprehensibleness" to "antidisestablishmentarian").
7. All words that start with either 'q' or 's', and that also contain a double z ('zz') later in the word.
   (On `attu` there are 66 of them, from "quizzability" to "swizzling".
8. All 5-letter words in the file `~stepp/390/words` that start/end with the same 2-letter substring.
   (This one takes a long time to run on the default `words` file on `attu`, so use the shorter file listed above.
   In that file there are 8 of them, from "edged" to "salsa".

Hints: Recall the **regular expression syntax** shown in class, such as a `.` for any character, `|` for "or," `[]` for character classes, `{}`, `*`, and `+` for quantities, `^` and `$` for specifying the start/end of a line, and `\<` and `\>` for specifying the start/end of a word. Also note that `grep` can use "back-references" to refer to sub-patterns captured previously between `(` and `)`, such as `\1` for the first captured sub-pattern, `\2` for the second, etc.

Task 2: Bash Shell Commands with `sed`:

For each of the next few problems, write a command that uses `sed` (preferably with the `-r` command-line argument to enable full regular expressions) to search and replace text based on regular expressions. For some problems, you may need to combine `grep`/`egrep` and `sed` using `|` . Each command should use at most one call to `sed` , but you may use input/output redirection operators such as >, <, and `|` to combine it with other commands as needed.

Write these commands into your `homework7.txt` file, one per line. In your file, **write the command** that will perform the task described for each numbered item; don't write the actual output that such a command would produce. Your commands should not create any **temporary files** during their execution.

9. Output the contents of the file `~stepp/390/email.txt`. with all spaces replaced by underscores.
10. Output the contents of the file `~stepp/390/Questions.java`. with all occurrences of the word "public" replaced with the word "private".
    (Don't match words that have "public" as a substring, such as "publicly" or "republic".)
11. Suppose that a writer named Victoria uses too many exclamation points. Output the contents of the file `~stepp/390/v.txt` but with all occurrences of one or more `!` marks in a row replaced with a single period.
    (For example, `!` would become `.` and `!!!!!!` would all be replaced by a single `.` .)
12. Java programs can contain single-line `//` comments and multi-line `/*` ... `*/` comments. Sometimes a programmer uses a multi-line comment syntax, but the comment only occupies a single line.
    Write a command that finds `/*` ... `*/` comments in `~stepp/390/Questions.java` that occupy a single line and replaces them with a `//` comment. For example, `/* hello there */` would become `// hello there` .
    (Your command doesn't need to modify comments where the `/*` isn't on the same line as the `*/` . You may assume that any given line contains at most one comment.)
13. Europeans format their dates differently than Americans. Where we would write a date such as "May 12, 2010", they would write it as "12 May 2010". Output the contents of file `~stepp/390/dates.txt` but with all dates changed from USA format to European format.
14. Convert all 10-digit phone numbers in the file `~stepp/390/phone.txt` to 5-digit internal extension numbers.
    (For example, `Abba, Cadabra 206-685-x67890` and `Timss, Aaron 206-685-x62859` .)
15. Give a modified version of #7 that displays the phone extension first, then 3 spaces, then the person's name.
    (For example, `x67890  Abba, Cadabra` and `x62859  Timss, Aaron` .)
16. All 12-character words in the `words` file that start with 'p' through 's' and end with "ker", in *Pig Latin*.
    In other words, show all of the letters of the word other than the first letter,
    followed by a dash `-`, followed by the first letter followed by "ay".
    (On `attu`, there are 25 of them, from "atternmaker-pay" to "witchbacker-say".)
    (*Hint:* Use `grep` or `egrep` first to reduce the file to the words of interest, before running `sed` on those words.)

Hints: The regular expression syntax hints from the previous section on `grep` also apply to `sed`. Also recall that some special characters must be escaped by a `\` backslash to be used in a regex pattern. Remember to put a 'g' at the end of your pattern, such as `s/oldpattern/newtext/g`, to process all matches.