More on Linux Tools

Objective

To practice using the Linux terminal and get more familiar with *regular expressions* and their use in Linux commands/tools such as:

- 'grep' (global regular expression print, or *egrep* for extended grep)
- 'awk' (the initials of the three authors Aho, Weinberger and Kernighan)

Description

You can always write programs (C, C++, Java, ….) to find data, process and manipulate it but this will take you time and effort. Any changes to the input data, may force you to rewrite parts of your code. Linux/Unix comes with many generic tools that can help you load and process data with very little programming needed. These tools rely heavily on RegExp (regular expressions).

Regular expressions:

A regular expression (regex or regexp) is a pattern that describes/defines a class of strings. A string can then be tested to see if it belongs to this class or not. Regular expressions go back to automata and formal language theories where a finite automaton (finite state machine - FSM) can be built to recognize valid strings defined by a formal language.

Example: given the alphabet {a,b}, the FSM in figure 1 has 3 states, Q0, Q1, and Q2 where Q0 is the initial state and Q2 is the end/accepting state. The machine describes a pattern of strings that have an even number (0,2,4,6,…) of the letter 'a' and will accept the following strings: "bbbb", "aba", "aabaa", "baabaabaa", "aaabbaabbbaaba", "bbbabbaababbb" etc.

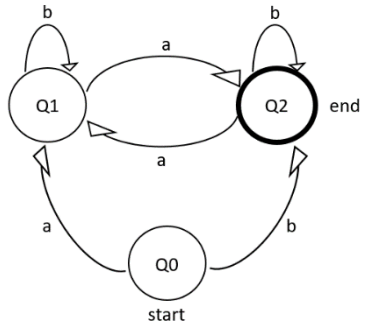

Figure 1 An FSM that accepts strings with even number of 'a's

Regex grammar defines the notation used to describe a regular expression including:

- ***Alternation***: vertical bar '|' is used to describe alternating choices among two or more choices. **'(a|b|c)'** indicates that a or b or c can be part of a string.
- ***Grouping***: parenthesis '( )' can be used to describe the scope and precedence of operators. The regex **'(c|s)at'** describes strings that begin with c or s and immediately followed by "at" such as "cat" or "sat"
- ***Quantification***: is the notation used to define the number of symbols that could appear in the string. The most common quantifiers are: ?, *, +
    - **?** mark indicates that there is **zero or one** of the expression to its left. The regex **'aab?b'** describe strings such as "aab" and "aabb", …etc.

  - ***** mark indicates that **zero or more** of the expression to its left. The regex **'a(ab)*b'** describe strings such as "ab", "aabb", "aababb", … etc.
  - **+** mark indicates **at least one** of the expression to its left. The regex **'go+gle'** describe strings such as **"gogle", "google" , "gooogle",** … etc.
- ***Other important rules to remember:***
  - '.' matches a single character
  - '.*' matches any string
  - '[a-zA-Z]*' matches any string of alphabetic characters
  - '[ag].*' matches any string that starts with a or g
  - '[a-d].*' matches any string that starts with a,b,c or d
  - '^ab' matches any string that begins with ab.
  - 'ab$' matches any string that ends with ab.
  - {n,m} at least n but not more than m times
  - {n,} – match at least n times
  - {n} – match exactly n times

Here are references on regular expressions:

- https://regexone.com/
- https://pythex.org/
- https://www.gnu.org/software/grep/manual/html_node/Regular-Expressions.html#Regular-Expressions
- https://ryanstutorials.net/regular-expressions-tutorial/regular-expressions-cheat-sheet.php
- https://www.regular-expressions.info/quickstart.html

Regex and Linux:

are used in Linux utilities to match a possible set of strings for output or future processing. Text editors like 'vi', filters such as 'grep' and scripting languages such as 'awk' are few examples.

'grep': is a tool that searches input files for lines containing a match to a given pattern and prints the matching lines. The general synopsis of the 'grep' command line is:

> grep *options pattern input_file_names*

```
NAME
    grep, egrep, fgrep, rgrep - print lines matching a pattern

SYNOPSIS
    grep [OPTIONS] PATTERN [FILE...]
    grep [OPTIONS] [-e PATTERN]...  [-f FILE]...  [FILE...]

DESCRIPTION
    grep searches the named input FILEs for lines containing a match to the given PATTERN.  If no files are specified,
or if the file "-" is given, grep  searches  standard  input.   By default, grep prints the matching lines.

    In addition, the variant programs egrep, fgrep and rgrep are the same as grep -E, grep -F, and grep -r,
respectively.  These variants are deprecated, but are provided  for  backward compatibility.

Source: Linux manual
```

For detailed examples, check the following links:

- https://www.digitalocean.com/community/tutorials/using-grep-regular-expressions-to-search-for-text-patterns-in-linux
- https://www.howtoforge.com/tutorial/linux-grep-command/
- https://javarevisited.blogspot.com/2011/06/10-examples-of-grep-command-in-unix-and.html

'awk': is not just a command but a scripting language designed for data processing and extraction. The basic function of 'awk' is to search files for lines that contain certain patterns. When a line matches one of the patterns, 'awk' performs specified actions on that line. It continues to process input lines until it reaches the end of the input files.
There are several ways to run an 'awk' program:

- If the program is short, it is easiest to include it in the command that runs 'awk':
  > awk *'program' input-file1 input-file2 …*
  Examples:
  $ awk '{print "Welcome to CIS 370 Lab "}'
  $ echo "Greetings from Mars" | awk '{$4="and"; $5="Earth"; print $0}'

- When the program is long, it is usually more convenient to put it in a program file and run it using the following command:
  > awk -f *"program-file" input-file1 input-file2 …*

For detailed examples, check the following links:
- https://likegeeks.com/awk-command/
- https://linuxhint.com/awk-command-examples/
- https://www.gnu.org/software/gawk/manual/gawk.html#Running-gawk
- http://www.grymoire.com/Unix/Awk.html

Submission:

One compressed file that contains your shell/awk/HLL (high-level language) programs and all the output for each of the following parts (*10 points for each part*):

1. **P1.sh** uses regular expressions with 'grep/egrep' to perform the following text parsing operations (use **UMassWhois.txt** as your input file):
   - Display all dates in the file.
   - Display all the all-upper-case words (with spaces on both sides).
   - Display all domain extensions in the document (.edu or .EDU, .com or .COM, .biz or .BIZ,...). Display the matches and the line numbers.
   - Display all the numbers that have at least 2 digits and are delimited by at least one space (e.g. the number 31 in NS31 does not fulfill the condition).
   - Display all root domain names (i.e. the words that come right before the .com, .edu, .biz,... of a URL)

   Please print blank lines between the previous 5 outputs to increase readability.

2. **P2.sh** uses 'awk' to perform the following information parsing operations (use **gradeBook.csv** as your input file):
    - Count the students in each of the three sections and then the total number of students.
    - Calculate the average score for Exams 1 and 2.
    - Print the names of students whose Exam 2 grades were less than their Exam 1 grades and count their number.
3. Perform the following text parsing operations (use **sampleLogFile.txt** as your input file):
    - **P3.(c|java|py)** uses a high-level language (C, Java, or Python) to parse the input file and print all log lines of type INFO that occurred in February 2024.
    - **P3.sh** uses Linux tools/commands to do the same task.

Special Characters for Regex:

abc…	Letters
123…	Digits
\d	Any Digit
\D	Any Non-digit character
.	Any Character
\.	Period
[abc]	Only a, b, or c
[^abc]	Not a, b, nor c
[a-z]	Characters a to z
[0-9]	Numbers 0 to 9
\w	Any Alphanumeric character
\W	Any Non-alphanumeric character
{m}	m Repetitions
{m,n}	m to n Repetitions
*	Zero or more repetitions
+	One or more repetitions
?	Optional character
\s	Any Whitespace
\S	Any Non-whitespace character
^…$	Starts and ends
(…)	Capture Group
(a(bc))	Capture Sub-group
(.*)	Capture all
(abc|def)	Matches abc or def

Standard character class names for Regex:

[:alnum:] – Alphanumeric characters.

[:alpha:] – Alphabetic characters

[:blank:] – Blank characters: space and tab.

[:digit:] – Digits: '0 1 2 3 4 5 6 7 8 9'.

[:lower:] – Lower-case letters: 'a b c d e f g h i j k l m n o p q r s t u v w x y z'.

[:space:] – Space characters: tab, newline, vertical tab, form feed, carriage return, and space.

[:upper:] – Upper-case letters: 'A B C D E F G H I J K L M N O P Q R S T U V W X Y Z'.