REGULAR EXPRESSIONS FOR COMMON UNIX UTILITIES : A TUTORIAL

1. Regular Expressions (POSIX standard)

A *regular expression* is a string that describes a whole set of strings according to certain syntax rules. These expressions are used by many text editors, utilities and web search engines to search bodies of text for certain patterns. Today, the most popular one is certainly *Google*. In the Unix community, very popular utilities are *grep* and *find*. Unfortunately there exists no standard for the syntax rules, but the basic ideas are the same. Here we will give a non exhaustive description for common Unix utilities.

In general, any character appearing in a regular expression matches that character in the text. For example, the regular expression 'pattern' matches the string 'pattern'. However certain characters are used to specify variable patterns and are therefore special. Useful special characters are:

.	The dot matches any single character.
[ ]	A set of characters in square brackets matches any single character from the set. Example: [abc] matches a, b, or c. [a-z] matches any lowercase letter.
[^]	Matches any single character that is not in the set. Example: [^a-z] matches any single character that isn't a lowercase letter.
^	A caret ^ at the beginning of an outermost regular expression matches the beginning of a line. Example: ^[0-9] matches any lines that begin with a digit.
$	A dollar sign at the end of an outermost regular expression matches the end of the line.
*	A single character expression followed by '*' matches to zero or more copies of the expression. Example: [xy]* matches to '', 'x', 'y', 'xx', 'xy', 'yx', and so on.
\	The backslash quotes the character after it. Example: '\.' matches the dot '.'

Examples

'.at'	matches any three-letter word ending with 'at'.
'[hc]at'	matches 'hat' and 'cat'.
'[^b]at'	matches any three-letter word ending with 'at' and not beginning with 'b'.
'^[hc]at'	matches 'hat' and 'cat' but only at the beginning of a line.
'[hc]at$'	matches 'hat' and 'cat' but only at the end of a line.

Exercise 1.1 What is the meaning of the following four expressions : 1) .a 2) \.a 3) [a]* 4) [^a]*

2. Extended Regular Expressions (POSIX standard)

In most modern Unix tools, the special characters have been extended as follow (not exhaustive):

|	Two regular expressions separated by '|' match an occurrence of either of them, i.e. the '|' operator acts as an OR. Example : 'a | b' means either a OR b, i.e. matches a and b.
( )	Parentheses are used for grouping regular expressions, so (exp1)(exp2) acts as a concatenation, i.e. '(a|b)c' matches 'ac' and 'bc'
*	An atom followed by '*' matches zero or more copies of the atom. Roughly speaking, an atom is a single character matching or a regular expression enclosed in `()'
+	Same as *, but matches only one or more copies of the previous atom.
?	Same as +, but matches only zero or one copy of the previous atom.

Examples

(a|b)(c|d)	matches 'ac', 'ad', 'bc' and 'ad'
[cC]at | [dD]og	matches 'cat', 'Cat', 'dog' and 'Dog'
(little)?cat	matches 'cat' and 'little cat'

3. grep

The Unix utility `grep` prints lines matching a given pattern. The command line has the form:

```
    % grep [options] pat [file]
```

where `pat` is a regular expression to be searched for and `[file]` is a list of files to be searched. If no files are specified, the program searches standard input.

In addition, two variant programs `egrep` and `fgrep` are available which are the same as `grep` with the option `-E`, respectively `-F`.

Some useful *options* are:

`-c`	Print the number of matching lines
`-i`	Ignore case distinctions in both `pat` and the input files
`-l`	Print the names of files having at least one match; the normal output is suppressed
`-n`	Print the line number and the line
`-r`	Read all files under each directory, recursively, and print all lines matching `pat`
`-s`	Suppress *error* messages about nonexistent or unreadable files
`-v`	Print all lines but those matching `pat`
`-w`	Print only those lines containing matches that form whole words
`-E`	Interpret `pat` as an extended regular expression
`-F`	Interpret `pat` as a list of fixed strings, separated by newlines, any of which is to be matched
`-H`	Print the filename for each match, in addition to the normal output

Examples

```
% egrep -l MAX *.c             # Lists the names of the C source files .c that contain the pattern MAX.
% egrep -r MAX ~/MyCProg/*     # Finds MAX in any file, recursively in ~/MyCProgr/  and all its
                               # subdirectories
% egrep -v '^[0-9]' report     # Finds all lines in the file report that do not begin with a digit.
% egrep -w live              # Finds only live in the input file(s); does not find liver, lives, lived, and so on.
% egrep '(little|big)(cat|dog)'   # Matches lilltlecat, littledog, bigcat and bigdog
% egrep '(little|big) (cat|dog)'  # Matches lilltle cat, little dog, big cat and big dog
```

Remark. Don't confuse the meaning of the asterisks '*' appearing in a regular expression (matches zero or more copies of the preceding single character expression or atom) and in a filename appearing in a Unix command (matches every string, including the empty string).

Exercise 3.1 What is the meaning of the following line commands:

```
(1)      % egrep -n  '{|}'  hello.c
(2)      % egrep -n  '.'  hello.c
(3)      % egrep -n  ''  hello.c
(4)      % egrep -c  ';$'  hello.c
(5)      % egrep -c  '[^;]$'  hello.c
(6)      % ls -R | egrep  '.c'
(7)      % ls -R | egrep  '\.c'
(8)      % ls -R | egrep  '\.c$'
(9)      % egrep -rl 'my_stack\.h>' ~/MyCProg/*
```

and which of the above patterns are extended regular expressions? Test your answer on machine!

Tip for debugging regular expressions. If you don't pass a filename to grep, it will read standard input, allowing you to enter lines of text to see which match, and grep will echo back only matching lines. Try this out, e.g. with `% egrep '(little )?cat'`

Exercise 3.2 Write a command allowing you to display the names of all *readable* files of your home and its subdirectories, recursively, containing (without quotes ") :

1. The pattern "Hello World" (search in all your files)
2. The word "Pattern" (search in all your files, with case non sensitive)
3. The pattern "Hello World" (search in all your source .c files)

Hint: Command (9) of exercise 3.1 is a good start. Question 3 is a bit more tricky.

4. Google Search

For pattern search, Google uses regular expression, but with its own syntax, see http://www.google.com/help/cheatsheet.html.

Exercise 4.1 Write a Google line search for finding web pages containing following words or patterns (without the quotes ") :

a. "University" and "Fribourg"
b. "University" or "Fribourg"
c. "University Fribourg"
d. "University Fribourg" or "University Bern"
e. "University Fribourg" or "University Bern", but not "University Geneva"

Note that google search is not case sensitive.

5. About Hard and Soft Quotes in the Bash Shell

Quoting is used to remove the special meaning of certain characters or words to the shell, e.g. quoting can be used to disable special treatment for special characters, to prevent reserved words from being recognized as such, and to prevent parameter expansion.

There are three quoting mechanisms: the escape character \ (which preserves the literal value of the next character that follows), single quotes, and double quotes.

<u>Single-quotes are hard-quotes</u>: Enclosing characters in single quotes preserves the literal value of each character within the quotes. A single quote may not occur between single quotes, even when preceded by a backslash.

<u>Double-quotes are soft-quotes</u>: Enclosing characters in double quotes preserves the literal value of each character within the quotes, with the exception of $, ', and \. The character $ and ' retain their special meaning within double quotes. The backslash retains its special meaning only when followed by one of the following characters: $, ', ", \, or <newline>.

Following commands illustrate this behaviour:

```
% touch '3&4.txt'     # touch the file '3&4.txt'
% ls -l '3&4.txt'     # the filename '3&4.txt' is displayed
% ls -l "3&4.txt"     # the filename '3&4.txt' is displayed
% ls -l 3\&4.txt      # the filename '3&4.txt' is displayed
% ls -l 3&4.txt       # strange: try to explain. Hint: what is the meaning of & ?
```

Exercise 5.1 What will the following command produce:

```
% echo $USER "$USER" '$USER'
```

More Readings *(Regular expressions)*

[1] % man re_format *-- Complete and rigorous description of Unix extended regular expressions*
[2] http://en.wikipedia.org/wiki/Regular_expression *-- Traditional_Unix_regular_expressions*
[3] http://www.regextester.com/ *-- A GUI allowing to test interactively regular expression via a browser*
[4] KRegExpEditor *-- A Linux editor allowing you to generate regular expressions with a graphical user interface*

More Readings *(grep)*

[5] % man grep

More Readings *(Quotes)*

[6] % man bash *(see section 'Quoting')*