THE UNIVERSITY OF SYDNEY

LAB **TWO**

Regular Expressions

2.1 Summary

RE	TYPE	COMMENTS
`a`	characters	use **`grep -i`** to get case insensitive
`.`	any character	matches against every character except newline
`^`	beginning of line	
`$`	end of line	
`*`	Kleene star	match *zero or more* instances of the previous RE
`+`	Kleene plus	match *one or more* instances of the previous RE
		requires **`grep -E`** or **`egrep`**
`?`	option	match *zero or one* instances of the previous RE
		requires **`grep -E`** or **`egrep`**
`|`	disjunction	match one RE or another
		matches the first matching RE from left to right
		requires **`grep -E`** or **`egrep`**
`[abc]`	character set	matches one of the characters in the set
`[a-z]`	ranges	matches one of the character inside the range
		endpoints must be in accending order e.g. not `[z-a]`
		you can specify multiple ranges in one char set e.g. `[a-zA-Z]`
`[^abc]`	complement	matches one of the characters not in the set
`(abc)`	grouping	group a RE
		allows it to be treated as a single RE
		e.g. `(abc)?` means *zero or one* instances of `abc`
		requires **`grep -E`** or **`egrep`**
`{m,n}`	multiplier	match between **`m`** and **`n`** instances of the previous RE
`{m,}`		match **`m`** *or more* instances of the previous RE
`{,n}`		match **`n`** *or less* instances of the previous RE
		these are only supported by GNU and BSD **`grep -E`** and **`egrep`**
`\n`	backreference	match the *n*-th group
		this is only supported by GNU and BSD **`grep -E`** and **`egrep`**
		and it is also supported by **`sed`**
`&`	match	match the match to this point
		this is only supported by **`sed`**

To match any of the special characters as themselves, escape them using the backslash `\` or put them in a character set , e.g. to match a period character (`.`) use `\.` or `[.]`

The different variants of regular expressions each version of **`grep`** supports makes it difficult sometimes. For example, many versions don't support multipliers (`{m,n}`). Others, e.g. GNU **`grep`**, support the **`grep -E`** options without the **`-E`** if they are escaped. For example, in GNU **`grep`**

```
% grep '\(abc\)\+'
```

is the same thing as:

```
% grep -E '(abc)+'
```

In general, it is best to represent your pattern using the most basic RE syntax you can, to avoid problems with the different versions.

2.2 Experimenting with Regular Expressions

Because **grep** and other commands can still accept input from the keyboard through standard input, it is easy to test your regular expression before running it on a file. You need to type in the bits in the **??script** font:

```
% egrep 'x(abc)?y'
xy
xy
xay
xabcy
xabcy
xabcdy
<Ctrl-D>
%
```

Press **<Ctrl-D>** to end the keyboard input. **grep** only prints out the lines that match the RE.

This is very helpful when you are about to run your RE on a large file. Imagine there are no matches – is that because there *are* no matches in the file, or because the RE doesn't match what you expect it to. One way of testing this is to play with it first.

So, what do the following regular expressions match? You can try them against **/usr/dict/words** and also using the keyboard. Remember to use the correct version of **grep** otherwise some won't work.

1. `.`
2. `.*`
3. `a.*z`
4. `[a-z]`
5. `^[a-zA-Z]+$`
6. `^[aeiou]+$`
7. `^[^aeiou]+$`
8. `^0[xX][0-9a-fA-F]+$`
9. `ax?b`
10. `axyz?b`
11. `a(xyz)?b`
12. `a(xyz)*b`
13. `a(xyz)+b` (what's the difference between the last four?)
14. `^a|b`
15. `^a|^b`
16. `^(a|b)` (and the last three?)

And finally, what's funny about the following?:

```
% grep -i '^[a-zA-Z]'
```

2.3 Tasks

Here are some examples to test out your Unix and regular expression skills.

2.3.1 Word lists

This exercise is designed to work on the file `/usr/dict/words` which is present on many Unix systems. You can download a copy from our website
`http://www.physics.usyd.edu.au/ioa/ausvoss/index.php/Main/BasicUnix`

How many lowercase words:

1. contain two u's in a row?
2. only consist of vowels?
3. only consist of consonants?
4. have four vowels in a row?
5. are palindromes? (hint: you need the command **`rev`**)

2.3.2 Analyse Pride and Prejudice

Download a free copy of Jane Austen's *Pride and Prejudice* from http://www.gutenberg.orgProject Gutenberg using the **`wget`** or **`snarf`** Unix programs:

```
% wget 'http://www.gutenberg.org/dirs/etext98/pandp12.txt'
```

or from our website
`http://www.physics.usyd.edu.au/ioa/ausvoss/index.php/Main/BasicUnix`

1. Create a list of all of the words of P&P in order, one word per line, just like **`/usr/dict/words`** with all of the punctuation removed.
2. Using this list, what are the two most popular names from this novel?
3. Ignoring punctuation, what is/are the longest word/s in the novel?