Regular Expression Primer

By Cal Henderson

Regular Expressaion (RegExps) are used for matching pieces of text. In perl a RegExp consists of delimiters, a pattern and some optional settings. There are three types of RegExp in perl: match, replace and transliterate; we'll look at each in turn.

The simple matching RegExp checks to see whether a pattern can be found in a string. For the moment we'll use '/' as our delimiter. Here's an example:

```
$my_variable =~ m/hello/i;
```

Lets' break that down and see what we have...

```
$my_variables =~
```

Perform a regexp on $my_variable.

```
m/
```

This is a matching regexp (if '/' is used as the delimiter, the 'm' can be ommited)

```
hello
```

The pattern to match - just a simple string in this case.

```
/i;
```

End of the pattern, and the 'i' option, which means the RegExp will be case insensitive.

Because $_ is the 'default variable' in perl, you can perform a regexp on it as follows:

```
/hello/i;
```

so if you see this sort of thing on it's own, then it's matching against $_.

The format of the pattern doesn't just have to contain alpha numerics. RegExps have 'meta-characters' which are not taken as literals. These are the following characters: "+ ? . * ^ $ ( ) [ ] { } | \". To use one of these characters literally, escape it with a slash. If you want to match "file.txt", use:

```
$var =~ h/file\.txt/;
```

Note: The delimiter you use also becomes a metachar, so always escape it if you want it to be literal! We'll go through each metachar, looking at what it does:

```
 + matches the preceeding group one or more times
* matches the preceeding group zero or more times
? matches the preceeding group zero or one times
{a} matches the preceeding group a times
{a,} matches the preceeding group a or more times
{a,b} matches the preceeding group between a and b times
```

a group is defined as either a single character, or a group of chars surrounded with brackets "()". A

demonstration:

```
 /bob+/ will match bob, bobb, bobbb
/(bob)+/ will match bob, bobbob, bobbob
```

The dot "." meta character matches any character, EXCEPT newlines. So:

```
 /.*/ matches anything, excluding newlines
/a./ matches aa, ab, ac, a*, a:
```

To make the dot match newline characters aswell, use single line 's' mode:

```
 /.*/s matches anything, including newlines
```

The caret "^" and dollar "$" signs mark the beginning and end of a line respectively. So this...

```
 /^the only thing in the string$/
```

...checks to see if the expression matches the entire line.

The square brackets are used to denote a "character class", which is a group of characters to choose from:

```
 /x[abc]x/ matches xax, xbx, xcx
```

A class can be used with other metachars too:

```
 /[abc]+/ matches b, acb, cabcbabcbab
```

The special case for classes is when they start with a caret "^". This negates the class, meaning it matches anything EXCEPT the characters within: `/[^aeiou]/ matches any non vowel character.`

The | metachar is used to give alternatives:

```
 /a|b|c/ matches a, b, c
```

It can also be used with groups, so:

```
 /(bob)|(jack)|[abc]/ matches bob, jack, a, b, c
```

(end of article)