Regex for people who should know Regex, but do not. PART 1

written by dbr/Ben
http://neverfear.org

Table of Contents

Part 1

Regular expressions are very useful. If you have to deal with text at all, you will find them indispensable. In programming, they can be used to verify user input (e.g check an email is valid - although remember, this is *hard* to do correctly), sanitize input (make sure the user only used allowed characters), remove sections of the string (remove all non-alpha-numeric characters). And not just for programming! You can use regular expressions in many text editors to edit the current file. You can use them in the grep command to find very specific lines, or in the sed "stream editor" to edit out bits of a file..

As you can tell from the previous paragraph, they are very versatile. Most implementations are more or less compatible with the Perl regex syntax, so this is what the guide is about

Ohno, scary line-noise

Regular Expressions (regex for short) are really very simple.. They may look horrendous (for example, the email to validate an email address correctly http://www.ex-parrot.com/~pdw/Mail-RFC822-Address.html), but they are essentially variations of the following:

```
[set of characters]{number of characters to match}
```

For example, [a-z]{2} will match two letters, from a-z.

In the previous [a-z] is the characters to look for, {2} is how many characters to match.

..that's it! Regular expressions have a really basic syntax, they just look like the result of someone kicking their keyboard for a while.

Matching letters

Character sets

Now, lets say you would to match more than just a-z. Maybe upper case, numbers, and a question-mark.. Easy:

```
[a-zA-Z0-9?]{2}
```

You can add different ranges or characters to the `[ ]` character-class.

But.. say you want to match the] symbol too! If you have any experience programming, you could probably guess - you escape the symbol with a backslash:

```
[a-z\]]{2}
```

The symbols you will almost always have to escape (as they have other meanings in regexs) are:

```
. + * [ ]  ( ) ^ $ \
```

You may also have to escape (significant) space, depending on the implementation, for example when using the verbose flag available in some implementations (which ignores any whitespace, which allows you to format the regex more readably)

Strings

Something I should definitely point out - you don't have to use character classes to match letters - you can match a normal string of letters very easily:

```
    some thing[!]{1,5}
```

That will match "some thing" followed by one to five exclamation marks. Just remember to escape any characters like [ ] * as discussed in the previous section

Shortcuts

Instead of typing every possible range of letters and symbols, the decimal-point can refer to "any single character". For example, to match any 5 characters:

```
    .{5}
```

There are also other "shortcuts" you can use inside [ ] sets, such as:

- \A is the same as [a-zA-Z]
- \S will match white-space (spaces and tabs)
- \w matches alpha-numeric (a-zA-Z0-9 plus underscore)
- Upper-case \W matches non-word characters (opposite of above)
- \d matches numbers
- \D matches non-numbers

There are plenty more, but those are the most useful I find. To see the complete list, the best place is probably the Perl Regex doc-page (http://perldoc.perl.org/perlre.html), as many languages implement "Perl-compatible regex" (PHP, Ruby and Python all do)

These can be combined into character classes. Say you want to match spaces and tabs, and question marks:

```
    [\S\?]{1,10}
```

You just shove them in with the other characters and ranges.

I generally find I use the dot "any characters", and character classes most often.

Number of characters

Now I've covered character sets (the things enclosed in [ ], remember?), the next important bit is the number of characters to match. For these examples, I will just match lower-case a-z.

There are a few different wildcard matchers, mainly ? * and +

? matches one or zero characters. The ? quantifier is often useful for matching a "optional character", like a trailing slash, or a trailing "s":

```
some thing[s]?
```

That will match both "some thing" and "some things"

And the two most useful:

To match one or more of a symbol:

```
[a-z]+
```

And match zero or more:

```
[a-z]*
```

Say I wanted to match any exclamation marks in a string, and if any are found, remove them:

```
[!]*
```

You would use your languages Regex substitute function to replace any matches with an empty string

Now say I want to check if the user has used any exclamation marks in their string:

```
[!]+
```

If that matches, there is at least one exclamation mark in the string, whereas using * would match even if there was none.

Groups

Groups are very very useful for extracting bits of a string. Basically you enclose a section of the regex in brackets. Then, depending on how the language deals with them, you can get the contents of these groups as variables.

Say I have a string "Viewing: |Some title|". I want to grab the title, without the "Viewing: ||" bit:

```
    Viewing: \|([a-zA-Z ]*)\|
```

I just wrote that, and already it's hard for me to read.. But I know it will work! Regular expressions are far easier to write than they are to read..

Basically I typed in the string "Viewing:", escaping the two |'s

```
  Viewing: \|\|
```

Next, I matched zero or more [a-zA-Z], inside the two | symbols.

```
  Viewing: \| [a-zA-Z ]* \|
```

And finally, so I can grab the title, I put the [a-zA-Z]* inside a group

```
  Viewing: \| ( [a-zA-Z ]* ) \|
```

(Note: Spaces added to improve readability)

That's about it

Using the above (if you can make sense of it), you can do a *lot* with regular expressions.

The second part of this little guide will give some practical examples on using regex, and I'll also show how to use them in various languages..
So, visit neverfear.org and subscribe to the RSS feed by clicking that orange icon in the address bar. Any questions, join the IRC channel at irc.neverfear.org #neverfear (port 7777, or 6697 for SSL)