Demystifying the syntax of regular expressions

Jun 5, 2002
Shelley Doll

Regular expressions (REs) are often wrongly perceived as mystical unknowns that only a true guru can understand. They're ugly to look at, and if you don't know the syntax, they can seem like cryptic strings of garbage you might get from a core dump. The reality is that regular expressions are very simple and can be easily picked up with the help of a handy cheat sheet. This article will help you amaze your friends by walking you though the most commonly used regular expressions.

Back up for a second

Mathematician Stephen Kleene first introduced regular expressions in 1956, as a result of his work with recursion sets in natural language. They were created as a set of syntax used to match patterns in strings, which he later helped apply to emerging information technology to facilitate automation. Since that time, regular expressions have been through a number of iterations, and the current standard is kept by the ISO (International Standards Organization) and defined by The Open Group, a collaborative effort of various technical non-profit organizations.

Regular expressions do not constitute a language in and of themselves, but are a defined standard for searching and replacing text within a file or any string. Two standards are maintained: basic regular expressions (BRE) and extended regular expressions (ERE). ERE includes BRE functionality, plus additional concepts.

Many applications use regular expressions, including xsh, egrep, sed, and vi, among others on UNIX platforms, and they have been adopted in one form or another by most programming languages. Just as HTML and XML are both subsets of SGML, these adaptations often represent a subset of the entire standard.

More common than you think

With the migration of regular expressions into cross-platform programming languages, their power has been utilized in more ways than may be obvious. Internet search engines use them; e-mail programs also use them, specifically to figure out what messages go in your inbox. Even if you're not a UNIX programmer, you can use regular expressions to simplify your applications, and save yourself a lot of time in parsing arrays and juggling variables.

Regular expressions 101

Much regular expression syntax should look familiar, because you've probably used them before even if you didn't realize what they were. Wildcards are one type of RE construct—*repetition operators*. Let's walk through the basic types of syntax that make up the most commonly used portions of the ERE

standard. In order to provide quality examples of specific usage, I've used a variety of applications.

Character matching

The crux of regular expressions is stating what you want to search for or match. Without this concept, REs are useless. Every expression will contain some instruction about what to look for; see **Table A**.

Table A: Character-matching regular expressions

Operator	Description	Sample	Result
.	Match any one character	grep .ord sample.txt	Will match "ford", "lord", "2ord", etc. in the file sample.txt.
[ ]	Match any one character listed between the brackets	grep [cng]ord sample.txt	Will match only "cord", "nord", and "gord"
[^]	Match any one character not listed between the brackets	grep [^cn]ord sample.txt	Will match "lord", "2ord", etc. but not "cord" or "nord"
		grep [a-zA-Z]ord sample.txt	Will match "aord", "bord", "Aord", "Bord", etc.
		grep [^0-9]ord sample.txt	Will match "Aord", "aord", etc. but not "2ord", etc.

Repetition operators

Repetition operators, or *quantifiers*, describe how many times to search for a specified string. They are used in conjunction with character-matching syntax to search for multiple characters; see **Table B**. Here's where support in different applications starts to vary, so you should check the documentation for the application you're using if your pattern doesn't work as expected.

Table B: Regular expression repetition operators

Operator	Description	Sample	Result
?	Match any character one time, if it exists	egrep "?erd" sample.txt	Will match "berd", "herd", etc. and "erd"
*	Match declared element multiple times, if it exists	egrep "n.*rd" sample.txt	Will match "nerd", "nrd", "neard", etc.
+	Match declared element one or more times	egrep "[n]+erd" sample.txt	Will match "nerd", "nnerd", etc., but not "erd"
{n}	Match declared element exactly n times	egrep "[a-z]{2} erd" sample.txt	Will match "cherd", "blerd", etc. but not "nerd", "erd", "buzzerd", etc.
{n,}	Match declared element at least n times	egrep ".{2,}erd" sample.txt	Will match "cherd" and "buzzerd", but not "nerd"
{n,N}	Match declared element at least n times, but not more than N times	egrep "n[e]{1,2} rd" sample.txt	Will match "nerd" and "neerd"

Anchors

Anchors describe where to match the pattern; see **Table C**. They can come in handy when you're searching for common string combinations. For some of these examples, I've used the vi line editor command*:s*, which stands for *substitute*. The basic syntax for that command is *s/pattern_to_match/pattern_to_substitute/*.

Table C: Regular expression anchors

Operator	Description	Sample	Result
^	Match at the beginning of a line	s/^/blah /	Inserts “blah “ at the beginning of the line
$	Match at the end of a line	s/$/ blah/	Inserts “ blah” at the end of the line
\<	Match at the beginning of a word	s/\</blah/	Inserts “blah” at the beginning of the word
		egrep “\<blah” sample.txt	Matches “blahfield”, etc.
\>	Match at the end of a word	s/\>/blah/	Inserts “blah” at the end of the word
		egrep “\>blah” sample.txt	Matches “soupblah”, etc.
\b	Match at the beginning or end of a word	egrep “\bblah” sample.txt	Matches “blahcake” and “countblah”
\B	Match in the middle of a word	egrep “\Bblah” sample.txt	Matches “sublahper”, etc.

Alternation
Another handy device in REs is the *alternation* or *infix* operator. Essentially, this operator is equivalent to an inclusive OR statement and is represented by the | symbol. The following statement will return all instances of the words “nerd” and “merd” in the file sample.txt:
egrep “(n|m)erd” sample.txt

Alternation can be a powerful tool when you’re looking for variable spellings in a file, but you can accomplish the same thing in this example with:
egrep “[nm]erd” sample.txt

Alternation’s real usefulness becomes apparent when you use it in conjunction with some of the more advanced features of REs.

Reserved characters
One last important concept in using basic REs is reserved characters (also called *special characters*). For example, if you want to look for the strings “ne*rd” and “ni*rd”, the pattern-matching statement “n[ei] *rd” will match variations of “neeeeerd” and “nieieierd” but not the strings you’re looking for. Because ‘*’ (the Kleene star) is a reserved character, you have to escape it using a backslash (\) in your pattern, like so: “n[ei]*rd”. Other reserved characters include:

- ^ (carat)
- . (period)
- [(left bracket}
- $ (dollar sign)
- ((left parenthesis)
-) (right parenthesis)
- | (pipe)
- * (asterisk)
- + (plus symbol)
- ? (question mark)

- { (left curly bracket, or left brace)
- \ backslash

Once you begin including these characters in your search strings, it's no wonder REs are more difficult to read than write. Consider this partial code, which calls the *eregi* engine in PHP to validate that an email address has been correctly formed:
eregi("^[_a-z0-9-]+(\.[_a-z0-9-]+)*@[a-z0-9-]+(\.[a-z0-9-]+)*$",$sendto)

Or how about this vi substitution to replace a global URI with a local filename:
s/http:\/\/www\.somedomain\.com\/somedir\/index\.html/\.\.\/somedir\/index\.html/

As you can see, it can become difficult to determine what exactly is going on. But if you don't escape special characters, you'll dramatically change the meaning of your pattern.

Summary
In this article, I've attempted to demystify regular expressions and offer a helpful listing of the more commonly used syntax of the ERE standard. If you're interested in looking at the complete description of The Open Group's guidelines, you can find them in the specification available online: Regular Expressions. Please feel free to post your questions or comments in the discussion area below, or send us an email with your feedback.