ITE315 Module 3 Part B - Programming in PERL: Pattern Matching And Regular Expressions

Athens State University

Let's Review...

Perl Regular Expressions

Using Regexs In Perl

Use Cases You Need to Know

Pattern Matching

- **Pattern matching**: Searching through text to find strings that match a pattern
- **Regular expressions**: A text string that describes a particular search pattern
- We've seen that modern operating system and modern text editors provide ways to use regular expressions to search for text in data
  - The `grep` command in the operating system
  - Pattern-based search and replace in `vim` and `emacs`

Don't Let Regular Expressions Intimidate You

- A way of specifying a *pattern* of characters to be matched in a string
- You'll hear me say "regexp" a lot; it's just an abbreviation for "regular expression"
- Lots of really neat computer science theory behind regexps

Perl and Regular Expressions

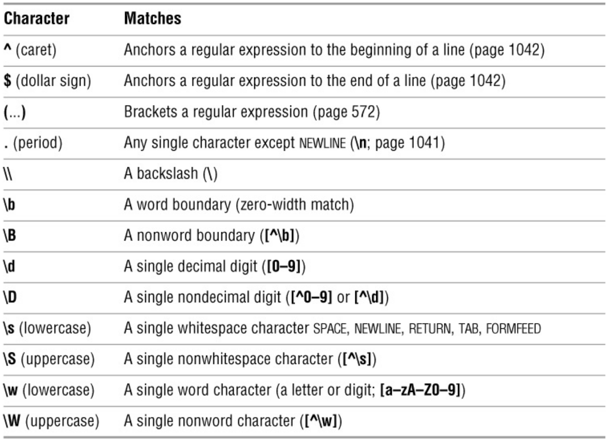
- As a language with a strong focus on text processing, Perl has extensive support for regular expression based pattern matching
- Perl's language for defining regular expressions is an extension of the language used by `grep` and the editors
- The differences between defining regular expressions for `grep` and Perl can be irritating
  - It's much like what happens when a person who speaks Spanish hears Portuguese or Italian
    - The similarity is just enough to where you think you understand but in fact...

Basic Perl Regex Syntax

- Regular expressions are delimited by the forward slash character
- Three classes of regular expressions:

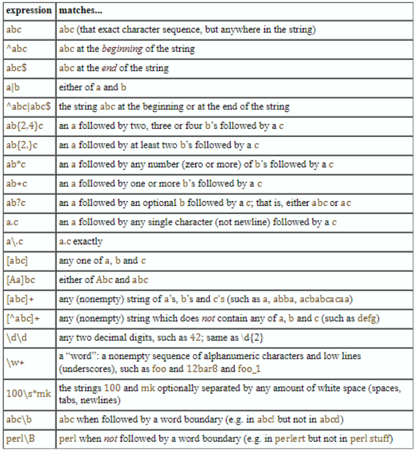
Match	m/abc/
Substitute	s/abc/def/
Translate	tr/abc/def/

Perl Regular Expressions Metacharacters



Character	Matches
^ (caret)	Anchors a regular expression to the beginning of a line (page 1042)
$ (dollar sign)	Anchors a regular expression to the end of a line (page 1042)
(...)	Brackets a regular expression (page 572)
. (period)	Any single character except NEWLINE (**\n**; page 1041)
\\	A backslash (\\)
\b	A word boundary (zero-width match)
\B	A nonword boundary (**[^\b]**)
\d	A single decimal digit (**[0–9]**)
\D	A single nondecimal digit (**[^0–9]** or **[^\d]**)
\s (lowercase)	A single whitespace character SPACE, NEWLINE, RETURN, TAB, FORMFEED
\S (uppercase)	A single nonwhitespace character (**[^\s]**)
\w (lowercase)	A single word character (a letter or digit; **[a–zA–Z0–9]**)
\W (uppercase)	A single nonword character (**[^\w]**)

Perl Regex Examples



expression	matches...
abc	abc (that exact character sequence, but anywhere in the string)
^abc	abc at the *beginning* of the string
abc$	abc at the *end* of the string
a|b	either of a and b
^abc|abc$	the string abc at the beginning or at the end of the string
ab{2,4}c	an a followed by two, three or four b's followed by a c
ab{2,}c	an a followed by at least two b's followed by a c
ab*c	an a followed by any number (zero or more) of b's followed by a c
ab+c	an a followed by one or more b's followed by a c
ab?c	an a followed by an optional b followed by a c; that is, either abc or ac
a.c	an a followed by any single character (not newline) followed by a c
a\.c	a.c exactly
[abc]	any one of a, b and c
[Aa]bc	either of Abc and abc
[abc]+	any (nonempty) string of a's, b's and c's (such as a, abba, acbabcacaa)
[^abc]+	any (nonempty) string which does *not* contain any of a, b and c (such as defg)
\d\d	any two decimal digits, such as 42; same as \d{2}
\w+	a "word": a nonempty sequence of alphanumeric characters and low lines (underscores), such as foo and 12bar8 and foo_1
100\s*mk	the strings 100 and mk optionally separated by any amount of white space (spaces, tabs, newlines)
abc\b	abc when followed by a word boundary (e.g. in abc! but not in abcd)
perl\B	perl when *not* followed by a word boundary (e.g. in perlert but not in perl stuff)

The Binding Operator

```perl
my $str = 'The black cat jumped from the green tree';
if ($str =~ m/cat/) {
    print "There is a cat\n";
}
```

- The binding operator is matching a scalar string against a regular expression
- In this case, applying the operator returns `$true` as the string cat is in the variable

Regular Expression Variables

```
#!/usr/bin/perl
$string = "The food is in the salad bar";
$string =~ m/foo/;
print "Before: $`\n";
print "Matched: $&\n";
print "After: $'\n";
```

The Substitution Operator

```
#/user/bin/perl
$string = "The cat sat on the mat";
$string =~ s/cat/dog/;
print "$string\n";
```

- This is an extension of the match operator that replaces the matched text with the new text
- The simularity with search and replace in `vim` is intentional

The Substitution Operator

```
#/user/bin/perl
$string = "The cat sat on the mat";
$string =~ tr/cat/dog/;
print "$string\n";
```

- Translate works on a per character basis rather a per string basis

Matches And Replacements Return A Quantity

- The match and substitute operators return the number of matches or replacements made by the action

```
if ( $str =~ /Diggle|Shelley/ ) {
   print "We found Pete or Steve!\n";
}

if ( my $n = ($str =~ s/this/that/g) ) {
   print qq{Replaced $n occurrence(s) of "this"\n};
}
```

Capture Variables

```perl
my $str = 'Perl 101 rocks.';
if ( $str =~ /(\d+)/ ) {
    print "Number: $1"; # Prints "Number: 101";
}

if ( $str =~ /(Python|Ruby)/ ) {
    print "Language: $1"; # Never gets here
}
```

Capture variable Don't Behave As You Expec

```perl
BAD: Not checked, but at least it "works".
my $str = 'Perl 101 rocks.';
$str =~ /(\d+)/;
print "Number: $1"; # Prints "Number: 101";

# WORSE: Not checked, and the result is not what you'd
     expect
$str =~ /(Python|Ruby)/;
print "Language: $1"; # Prints "Language: 101";
```

Funky Capture Variable Use Cases

```perl
#!/usr/bin/env perl
# n2 - extract forename and surname
print "please enter your name ";
chop ($name = <STDIN>);
if ($name =~ /^\s*(\S+)\s+(\S+)\s*$/) {
 print "Hi $1. Your Surname is $2.";
} else {
 print "no match";
}
print "\n";in{lstlisting}[language=perl]
```

Chop and Chomp

- The `chop()` and `chomp()` functions are used for parsing input from strings and files
- `chop()`: Remove the last character of a string and returns that character. If passed a list of arguments, perform the operation on each one and return the last character chopped
- `chomp()`: Removes characters at the end of strings corresponding to the input line separator

Chop and Chomp

```perl
#chomp() EXAMPLES
$a = "abcdefghij";
chomp($a);
#would return exact string... nothing to remove
print $a;

$a = "abcdefghij\n";
chomp($a);
#would return 'abcdefghij', removed newline
print $a;

$a = "abcdefghij\n";
$b = chomp($a);
#would return 1, it did remove something for sure
print $b;
```

Chop and Chomp

```
#chop() EXAMPLES
$a = "abcdefghij";
chop($a);
#this would return 'abcdefghi'
print $a;

$a = "abcdefghij";
$b = chop($a);
#this would return 'j'
print $b;
```