

Perl Regular Expressions



UVic SEng 265

Daniel M. German

Department of Computer Science

University of Victoria

September 25, 2002 Version: 1.00

Perl Regular Expressions



- ❖ Perl is renowned for its excellence in text processing.
- ❖ Regular expressions are a big factor behind this fame.
- ❖ Mastering even the basics will allow you to manipulate text with ease.
- ❖ Regular expressions have a strong formalism (finite state automata, which you will learn in future CS courses)

The Basics: Simple String matching

- ❖ The simplest reg exp is a word

```
if ("Hello World" =~ /World/) {  
    print "It matches\n";  
} else {  
    print "It doesn't match\n";  
}
```

- ❖ It does variable interpolation (double quotes style)

```
$greeting = "Hello";  
if ("Hello World" =~ /$greeting/) {  
    print "It matches\n";  
} else {  
    print "It doesn't match\n";  
}
```

Variations of the same theme

- ❖ If you're matching against the special default variable `$_`, `$_ =~` can be omitted:

```
$_ = "Hello World";  
if (/World/) {  
    print "It matches\n";  
} else {  
    print "It doesn't match\n";  
}
```

- ❖ If a regexp matches in more than one place in the string, perl will always match at the earliest possible point in the string:

```
"Hello World" =~ /o/;           # matches 'o' in 'Hello'  
"That hat is red" =~ /hat/;    # matches 'hat' in 'That'
```

Metacharacters

- ❖ Meta characters have special meaning.

`{ } [] () ^ $. | * + ? \`

- ❖ To match one of the metacharacters, put a backslash in front:

```
"2+2=4" =~ /2+2/;      # doesn't match, + is a metachar
"2+2=4" =~ /2\+2/;     # matches, \+ is an ordinary +
"The interval is [0,1)." =~ /[0,1)./ # syntax error!
                                   # (more later)
"The interval is [0,1)." =~ /\[0,1\)\/ # matches
"/usr/bin/perl" =~ /\usr\/local\/bin\/perl/; # matches
'C:\WINDOWS' =~ /C:\\WINDOWS/;   # matches
```

An example of simple searching

- ❖ This program searches for a pattern in the dictionary file

```
#!/usr/bin/perl
$regexp = shift;
while (<>) {
    if (/ $regexp/) {
        print $_;
    }
}
```

with result:

```
$ ./simple_grep.pl pter /usr/dict/words
chapter
copter
diopter
helicopter
pterodactyl
teleprompter
```

Where the match occurs

❖ You can anchor the search to certain locations:

❖ `^` matches at the beginning of a string

❖ `$` matches at the end of string or before newline

```
"housekeeper" =~ /keeper/;      # matches
```

```
"housekeeper" =~ /^keeper/;    # doesn't match
```

```
"housekeeper" =~ /keeper$/;    # matches
```

```
"housekeeper\n" =~ /keeper$/;  # matches
```

```
"keeper" =~ /^keep$/;          # doesn't match
```

```
"keeper" =~ /^keeper$/;        # matches
```

```
" " =~ /^$/;                   # ^$ matches an empty string
```

Character Classes

- ❖ A character class allows a set of possible characters, rather than just a single character, to match at a particular point in a regexp.

```
/cat/;          # matches 'cat'
/[bcr]at/;      # matches 'bat', 'cat', or 'rat'
/item[0123456789]/; # matches 'item0' or ... or 'item9'
"abc" =~ /[cab]/; # matches 'a'
/[yY][eE][sS]/;  # match 'yes' in a case-insensitive
                  # way 'yes', 'Yes', 'YES', etc.
/yes/i;          # simpler way, using i modifier
/(?i)yes/;       # same
/[\\]c]def/;     # matches ']def' or 'cdef'
$x = 'bcr';
/[$x]at/;        # matches 'bat', 'cat', or 'rat'
/[\\$x]at/;      # matches '$at' or 'xat'
/[\\\$x]at/;     # matches '\at', 'bat', 'cat', or 'rat'
```


Range operator within Character Classes

- ❖ With ranges, the ugly [0123456789] and [abc...xyz] become the slender [0-9] and [a-z]

```
/item[0-9]/;    # matches 'item0' or ... or 'item9'
/[0-9bx-z]aa/;  # matches '0aa', ..., '9aa',
                  # 'baa', 'xaa', 'yaa', or 'zaa'
/[0-9a-fA-F]/;  # matches a hexadecimal digit
/[a-z]/i;       # matches a "word" character,
                  # like those in assignment 2 and 3
```

- ❖ If '-' is the first or last character in a character class, it is treated as an ordinary character:

```
/[-ab]/;
/[ab-]/;
/[a\-b]/;      # all are equivalent. Match a, b or -
```

Negated character classes

- ❖ The special character `^` in the first position of a character class denotes a negated character class
- ❖ Matches any character but those in the brackets

```
/[^a]at/;    # doesn't match 'aat' or 'at', but matches  
              # all other 'bat', 'cat', '0at', '%at', etc.  
/[^0-9]/;    # matches a non-numeric character  
/[a^]at/;    # matches 'aat' or '^at'; here '^' is ordinary
```

Matching *any* character

- ❖ The period '.' matches any character but "\n"
- ❖ A period is a metacharacter, it needs to be escaped to match as an ordinary period.

```
/.rt/;      # matches any two chars, followed by 'rt'
/end\./;    # matches 'end.'
/end[.]/;   # same thing, matches only end.
""          =~ /.;/;    # doesn't match; needs a char
""          =~ /^.$/;   # doesn't match; needs a char
"\n"        =~ /^.$/;   # doesn't match; needs a char other than \n
"a"          =~ /^.$/;   # matches
"a\n"        =~ /^.$/;   # matches, ignores the \n
```

Matching this or that



- ❖ We would like to match different possible words or character strings
- ❖ We use the alternation character | (pipe)

```
"cats and dogs" =~ /cat|dog|bird/; # matches "cat"
```

```
"cats and dogs" =~ /dog|cat|bird/; # matches "cat"
```

Grouping Things Together

❖ Sometime we want alternatives for just part of a regexp.

```
/(a|b)b/;      # matches 'ab' or 'bb'
/(ac|b)b/;     # matches 'acb' or 'bb'
/(^a|b)c/;     # matches 'ac' at start of string or
               # 'bc' anywhere
/(a|[bc])d/;   # matches 'ad', 'bd', or 'cd'

/house(cat|)/;  # matches either 'housecat' or 'house'
/house(cat(s|)|)/; # matches either 'housecats' or
                  # 'housecat' or 'house'.
                  # Note groups can be nested.
```

Extracting Matches

- ❖ The grouping metacharacters () also serve another completely different function: they allow the extraction of the parts of a string that matched.
- ❖ For each grouping, the part that matched inside goes into the special variables \$1, \$2, etc.

```
# extract hours, minutes, seconds
$time =~ /(\d\d):(\d\d):(\d\d)/; # match hh:mm:ss format
                                # \d is equivalent to [0-9]

$hours = $1;
$minutes = $2;
$seconds = $3;
# More compact code, equivalent code
($hours, $minutes, $seconds) =
    ($time =~ /(\d\d):(\d\d):(\d\d)/);
```

Matching Repetitions

- ❖ We would like to be able to match multiple times:
 - ❖ $a?$ = match 'a' 1 or 0 times
 - ❖ a^* = match 'a' 0 or more times, i.e., any number of times
 - ❖ a^+ = match 'a' 1 or more times, i.e., at least once
 - ❖ $a\{n,m\}$ = match at least n times, but not more than m times.
 - ❖ $a\{n,\}$ = match at least n or more times
 - ❖ $a\{n\}$ = match exactly n times

```
$year =~ /\d{2,4}/; # make sure year is at least 2 but
                    # not more than 4 digits
/[a-z]+\d*/i;      # match a word and any number of digits
/y(es)?/i;         # matches 'y', 'Y',
                    # or a case-insensitive 'yes'
```

Search and Replace



- ❖ Regular expressions also play a big role in search and replace operations in perl
- ❖ Search and replace is accomplished with the `s///` operator
- ❖ General form: `s/regexp/replacement/modifiers`

```
$x = "Time to feed the cat!";  
if ($x =~ s/cat/hacker/) {  
    # $x contains "Time to feed the hacker!"  
    print "Found the cat, replaced with hacker\n";  
}
```


More Search and Replace Commands

```
$y = "'quoted words'";  
$y =~ s/^(.*)'$/<<$1>>/; # strip single quotes,  
                           # $y contains "<<quoted words>>"
```

```
$x = "I batted 4 for 4";  
$x =~ s/4/four/;      # doesn't do it all:  
                       # $x contains "I batted four for 4"  
$x = "I batted 4 for 4";  
$x =~ s/4/four/g;     # /g modifier does it all:  
                       # $x contains "I batted four for four"
```