

CS306: Introduction to Perl

Pattern Matching and Regular Expressions

U. of Alabama at Birmingham
Dept. of Computer & Information Sciences

Slide 1

What are Regular Expressions?

- A regular expression is a template which defines a pattern of text which you then compare to other text looking for a match. The result is either “match” or “doesn't match”.
- Regular expression syntax is its own mini-language which you use to write the mini programs, or templates.

Slide 2

What are Regular Expressions? Con't

- Not unique to perl - UNIX utilities like the shell, sed, awk, egrep and other programming languages also support regular expressions
- Perl has the best implementation of regular expressions

Slide 3

Finding a Word

- Say we want to find the string “hello” in some text. One way to do it:
- ```
my $text = "Before I could say hello she was gone.";
foreach $word (split ' ', $text) {
 if ($word = "hello") {
 $found = 1;
 last;
 }
}
if ($found) { print "We found the word 'hello'"; }
```

Slide 4

## Finding a Word

- Disadvantages of this method
  - Messy, lots of code
  - slow
  - inflexible – would not find “hello!”
- Same problem with regular expressions...
  - if (\$text =~ /hello/) { print “Found the word 'hello'”;}
  - Much easier!

Slide 5

## Finding a Word

- The /hello/ is the regular expression. It looks for “hello” **anywhere** in a string.
- \$text2 = “First she said hello, then she said bye”;  
# this one matches  
if (\$text2 =~ /hello/) { print “Matched hello”; }  
# this one does not match – must match ALL chars  
if (\$text2 =~ /hello then/) {  
    print “Matched hello then”;  
}

Slide 6

## More Examples

- \$text = “The Eagles beat the Cowboys.”;  
if (\$text =~ /boys/) { ... }  
if (\$text =~ / boys/) { ... }  
if (\$text =~ /eagles/) { ... }  
if (\$text =~ /Eagles/) { ... }  
if (\$text =~ /Eagles/i) { ... }  
if (\$text !~ /fish/) { ... }

Slide 7

## Interpolation

- \$text = “The Gators beat LSU this weekend.”;  
\$search = “Gators”;  
if (\$text =~ /\$search/) { ... } # Matches

Slide 8

## Pattern Behavior

- A pattern by default tries to match anywhere in the string.
- You can think of it as starting at the front of the string and “sliding” along the string until it matches or reaches the end.
- **The Leftmost Longest Rule.** Given a choice, a pattern will make the leftmost, longest match it can.

Slide 9

## Metacharacters

- Metacharacters allow us to be less literal in our patterns
- `.` - anything except `\n`
- `*` - match preceding thing zero or more times
- `+` - match preceding thing one or more times
- `?` - match preceding thing zero or one times
- The `\` turns metacharacters back into regular ones

Slide 10

## Metacharacter examples

- `/Fran.Fabrizio/`

Slide 11

## Metacharacter examples

- `/Fran.Fabrizio/`
  - `FranLFabrizio`
  - `Mr. Fran Fabrizio`
  - `FranFabrizio`
  - `Fran\Fabrizio`
  - `Francis Fabrizio`

Slide 12

## Metacharacter examples

- `/Fran.Fabrizio/`
  - `FranLFabrizio`
  - `Mr. Fran Fabrizio`
  - `FranFabrizio` # No match
  - `Fran\Fabrizio`
  - `Francis Fabrizio` # No match

Slide 13

## Metacharacter Examples

- `/Hello *World/`

Slide 14

## Metacharacter Examples

- `/Hello *World/`
  - `Hello World Leaders`
  - `Hello` World
  - `Helloo World`
  - `HelloWorld`

Slide 15

## Metacharacter Examples

- `/Hello *World/`
  - `Hello World Leaders`
  - `Hello` World
  - `Helloo World` # No match
  - `HelloWorld`

Slide 16

## Metacharacter Examples

- `/books?/`
  - Matches `book` and `books`
- `/Echo+/`
  - Matches `Echo`, `Echoo`, `Echooo`, `Echoooo` etc...

Slide 17

## Metacharacter Combinations

- `/hey.*there/`
  - “Hey, followed by zero or more anything characters, followed by there”
  - Which match?
    - hey there
    - hey Joe, how's the weather there in Phoenix?
    - they are late again, and therein lies the problem

Slide 18

## Metacharacter Combinations

- `/hey.*there/`
  - “Hey, followed by zero or more anything characters, followed by there”
  - Which match?
    - hey there
    - hey Joe, how's the weather there in Phoenix?
    - they are late again, and therein lies the problem

Slide 19

## Anchors

- *Anchors* force the pattern to match at a specific place, rather than sliding down the string looking for a match
- `^` - match at the beginning of a string.
- `$` - match at the end of a string

Slide 20

## Anchor Examples

- `/^Subject/` - look for a line that begins “Subject”
- `/phone home$/` - look for a line that ends “phone home”.
  - A note here: `$` matches either at the end of a string or at a newline at the end of a string. Therefore, it will match “phone home” and “phone home\n”

Slide 21

## Character Classes

- A *character class* appears between `[ ]` and matches any single character in the class. **It still is only matching one character in the text.**
- `[abcdxyz]` - match any of those seven characters.
- `[a-zA-Z]` - you can use ranges in character classes
- `/0x[0-9A-Fa-f]+/` - match hexadecimal numbers

Slide 22

## Saying “not” in Character Classes

- Use the `^` at the beginning to say “not”. Sometimes that's easier.
- `[^aeiou]` - will match any one character EXCEPT lowercase vowels

Slide 23

## Magic Character Class Shortcuts

- Some character classes are so common they have magic shortcuts
- `\d` is short for `[0-9]` (“digit” character)
- `\w` is short for `[0-9a-zA-Z_]` (“word” character)
- `\s` is short for `[\f\t\n\r ]` (“whitespace” character)
- Note that these abbreviations eliminate the `[ ]` too. So, `/cs[0-9]+/` becomes `/cs\d+/`

Slide 24

## Everything but the shortcuts...

- `\d` means “any digit character”. `\D` means “anything but a digit character”.
- `\w` means “any word character”. `\W` means “anything but a word character.”
- You see the pattern. `\S` is “any non-whitespace character”.

Slide 25

## “Or” in Patterns

- Use the pipe `|` character
- `/Perl|Java|C/`
  - What's the English translation?
- `/Fran( \t)+Fabrizio/`
  - What's the English translation?
- `/Fran( +\t+)Fabrizio`
  - What's the English translation?

Slide 26

## Saying “or” in Patterns

- Use the pipe `|` character
- `/Perl|Java|C/`
  - “Perl or Java or C”
- `/Fran( \t)+Fabrizio/`
  - “Fran, followed by one or more spaces or tabs in any combination, followed by Fabrizio”
- `/Fran( +\t+)Fabrizio`
  - now it has to be all spaces or all tabs, not mixed

Slide 27

## Grouping Patterns

- Use `()` to group characters
- `/(fred)+/`
  - One or more “fred”s
  - Matches:
    - fred
    - fredfredfred
    - barneyfredbambam
    - Alfred, pull the Batmobile around to the front please.

Slide 28

## Grouping Patterns

- `/ye(slt)/`
- `/th(islatle other)/`

Slide 29

## Quantifiers

- We've seen some already:
  - \* - zero or more
  - + - one or more
  - ? - zero or one
- Others:
  - `{3}` - exactly 3 times. `\w{3}/`
  - `{3,}` - at least 3 times `/(fred){3,}/`
  - `{,3}` - DOESN'T WORK!

Slide 30

## The `m//` match operator

- This is what we've been doing all along today, with the `/pattern/` syntax. We use the match operator `m/pattern/` to test a pattern on a string.
- Just like `qw//`, the `m/pattern/` can use other delimiters besides `//`. For instance, `m#pattern#` or `m!pattern!`.
- However, when using the `m/pattern/` form we can omit the `m`, leaving just `/pattern/`

Slide 31

## pattern matching and `$_`

- ```
while (my $line = <>) {  
    if ($line =~ /^Subject:/) {  
        print "Found the subject line: $line\n";  
    }  
}
```

Slide 32

pattern matching and \$_

- while (<>) {
 if (/^Subject:/) {
 print "Found the subject line: \$_\n";
 }
}

Slide 33

Remembering Matches

- () have previously been used for grouping patterns, like /(fred)+/ is one or more “fred”s
- () are also *memory parentheses*. When you use the (), the regular expression engine remembers the substring that matched that part of the pattern.
- So, if /d+/ matches one or more digits, /(d+)/ also matches one or more digits, and remembers which digits caused the match

Slide 34

Memory Example

- /^Subject: (.*)/ - will match from the beginning of the string “Subject: “ and then zero or more anything and remember those zero or more anything. i.e. it will grab the rest of the line.
- So, “Subject: Pizza Party Friday”, the memory would be “Pizza Party Friday”

Slide 35

Multiple Memories

- /(\d{3})-(\d{3})-(\d{4})/ - phone number, remembering each piece separately
- /(\w+) (\d+), (\d+)/ - Month, day, year
 - Jan 6, 2005
 - January 06, 2005

Slide 36

Accessing Memory - Match Variables

- ```
print "Enter phone no. (XXX-XXX-XXXX): ";
chomp(my $number = <STDIN>);
if ($number =~ /\d{3}-\d{3}-\d{4}/) {
 my ($areacode, $prefix, $num) = ($1, $2, $3);
} else {
 print "Phone number invalid.\n";
}

```
- \$1, \$2, \$3, etc... are the match variables

Slide 37

## Match Variable Shortcut

- ```
print "Enter phone no. (XXX-XXX-XXXX): ";
chomp(my $pn = <STDIN>);
unless (my ($ac, $pf, $num) = $pn =~ /\d{3}-\d{3}-\d{4}/) {
    print "Phone number invalid.\n";
}

```
- In list context, the regular expression returns a list of the memory vars, enabling syntax above.

Slide 38

Persistence of Match Variables

- The match variables get populated on successful match, and stay populated until next -successful-match.
- This implies that you should always check for success when doing a pattern match, otherwise you might not have what you think in \$1, \$2, ...
- So, patterns almost always in if() or while()

Slide 39

Substitution with the s/// operator

- “Search and Replace”
- ```
my $str = "Today's class is on Wednesday";
$str =~ s/Wednesday/Thursday/;

my $str = "I'm playing racquetball with Hari
today.\n";
$str =~ s/with ([A-Z]\w+)/against $1/;

```

Slide 40

## Global Search and Replace

- Use the g modifier
- my \$str = "I play racquetball with Hari today and with Vijay tomorrow.\n";

```
$str =~ s/with ([A-Z]\w+)/against $1/g;
```

```
Now str is "I play racquetball against Hari
today and against Vijay tomorrow.\n";
```

Slide 41

## Using patterns with split()

- We've already seen things like:  
my \$str = "The cow jumped over the moon.";  
my @words = split //, \$str;
- The // is a pattern. So this also works:  
my \$str = "Fee FiFo\nFum";  
my @words = split /\s+/, \$str;

Slide 42