Perl-Praxis

Reguläre Ausdrücke

Jörn Clausen
joern@TechFak.Uni-Bielefeld.DE

Übersicht

- Reguläre Ausdrücke
- Muster suchen
- Muster finden

zur Erinnerung

- Perl := Practical Extraction and Report Language
- Text-Dateien zeilenweise einlesen
- Zeilen
  - nach Mustern durchsuchen
  - in Worte zerlegen
- Textstücke ersetzen

Reguläre Ausdrücke

- beschreiben einfache Sprachen
- „schwächste“ Chomsky-Grammatik
- in Perl: *regular expressions* (RE)
- REs in Perl deutlich mächtiger als Reguläre Sprachen

Die Chomsky-Hierarchie:

Typ-0-Sprachen	allgemeine Regelsprachen
Typ-1-Sprachen	kontextsensitive Sprachen
Typ-2-Sprachen	kontextfreie Sprachen
Typ-3-Sprachen	reguläre Sprachen

regular expressions

- Suchen:

```
m/Gold/
```

- Kurzform:

```
/Gold/
```

- Ersetzen:

```
s/Blei/Gold/
```

- RE an Ausdruck binden: =~ (*match operator*)

```
$story = 'In a hole in the ground there lived a boggit.';
if ($story =~ /ground/) { ... }
$story =~ s/boggit/hobbit/;
```

Wenn der Suchausdruck mit `m` eingeleitet wird, kann (fast) jedes beliebige andere Zeichen anstatt des *slash* als „Klammer“ verwendet werden. Dies sind alles äquivalente Ausdrücke:

```
m#Gold#
m|Gold|
m!Gold!
```

Dies ist vor allem dann Hilfreich, wenn der slash Teil des Suchmusters ist, z.B. wenn Verzeichnispfade oder URLs verarbeitet werden sollen. Sonst müßte ein slash, der nicht Begrenzer des Suchmusters ist, als `\/` maskiert werden. Analog gilt dies für die Ersetzung mit `s///`.

modifier

- beeinflussen Verhalten von `m//` und `s///`
- nur die zwei wichtigsten:

```
m/gold/i       # case insensitive
s/Blei/Gold/g  # global
```

Aufgaben

- Wir verwenden im weiteren Verlauf die Dateien `romeo.txt` und `eric.txt`, um Texte zu suchen und zu manipulieren. Es handelt sich um Theaterstücke von Wayne Anthoney. Siehe

  http://www.dramex.org

- Wieviele Zeilen von `romeo.txt` enthalten das Wort „Gold“?
- Ersetze in `eric.txt` das Wort „Estragon“ durch „Basil“ und „Vladimir“ durch „Ilyich“.

- Nach Gold suchen:

```
$count = 0;
open(DRAMA, 'romeo.txt') || die "can't open romeo.txt: $!\n";
while ($line = <DRAMA>) {
    $count++ if $line =~ /gold/i;
}
close(DRAMA);
print "Found $count lines of pure gold\n";
```

Die Datei enthält sieben mal das Wort „gold“ und neun mal das Wort “Gold“.

- Namen ersetzen:

```
open(DRAMA, 'eric.txt') || die "can't open eric.txt: $!\n";
while ($line = <DRAMA>) {
    $line =~ s/Estragon/Basil/g;
    $line =~ s/Vladimir/Ilyich/g;
    print $line;
}
close(DRAMA);
```

regular expressions, cont.

- Alternativen:

```
m/Huey|Dewey|Louie/
```

- Gruppierung:

```
m/(Hu|Dew)ey|Louie/
```

- Quantoren

```
m/ab?a/         # aa, aba
m/ab*a/         # aa, aba, abba, abbba, abbbba, ...
m/ab+a/         # aba, abba, abbba, abbbba, ...
m/ab{3,6}a/     # abbba, abbbba, abbbbba, abbbbbba
m/a(bab)+a/     # ababa, ababbaba, ababbabbaba, ...
```

regular expressions, cont.

- Zeichenklassen

```
m/hello\s+world/   # whitespace
m/es ist \d+ Uhr/  # digits
m/name: \w+/       # letters (words)
```

- „Gegenteile“: `\S`, `\D`, `\W`
- selbstgemachte Zeichenklassen

```
m/M[ea][iy]er/     # Meier, Meyer, Maier, Mayer
m/[a-z]{2,8}/      # Account-Namen
m/[A-Z][^0-9]+/
```

- paßt auf alles: `.`

Aufgaben

- Lies die Datei „`/etc/services`“ zeilenweise ein.
  - Gib alle Zeilen aus, die sich auf das Protokoll „TCP“ beziehen.
  - Gib alle Zeilen aus, die einen Service definieren (also keine Kommentarzeilen sind).
  - Gib alle Zeilen aus, die eine vier- oder fünfstellige Port-Nummer enthalten.

- Protokoll „TCP“:

```
while ($line = <SERV>) {
    # print $line if $line =~ /tcp/;
    print $line if $line =~ /\d+\/tcp/;
}
```

Im Prinzip reicht es, nach dem Text “tcp“ zu suchen. Zur Sicherheit sollte aber nach der Kombination “Portnummer/tcp“ gesucht werden.

- Kommentarzeilen herausfiltern:

```
while ($line = <SERV>) {
    print $line if $line =~ /\w+\s+\d+\/(tcp|udp)/;
}
```

Es reicht nicht, einfach nach “#“ zu suchen, da Kommentare auch in Zeilen vorkommen, die einen Service definieren. Daher wird das Muster einer Service-Definition zur Suche verwendet: Name, Leerzeichen, Ziffern, Slash, “tcp“ oder “udp“.

- Vier- oder fünfstellige Portnummer:

```
while ($line = <SERV>) {
    print $line if $line =~ /\d{4,5}\/(tcp|udp)/;
}
```

Anker

- Muster an bestimmte Position binden
- Zeilenanfang, Zeilenende:

```
m/^From: .+/      # mail header
s/\s+$//          # remove trailing whitespace
m/^\d+ \d+ \d+$/  # 3d coords
```

- Wortzwischenraum:

```
m/\bmit\b/        # "nicht mit mir", "Kommen Sie mit!"
m/\bmit\B/        # "mittendrin", nicht "vermitteln"
```

Aufgaben

- Vereinfache das Skript, das alle Kommentarzeilen aus `/etc/services` herausfiltert.
- Extrahiere alle Zeilen aus `romeo.txt`, in denen die Worte „club“ oder „clubs“ vorkommen, aber nicht „clubroom“.

- Suche nach „#“, die am Zeilenanfang stehen:

```
while ($line = <SERV>) {
    next if $line =~ /^#/;
    print $line;
}
```

- „club“ und „clubs“, aber nicht „clubroom“:

```
while ($line = <DRAMA>) {
    print $line if $line =~ /clubs?\b/;
}
```

pattern capturing

- bis jetzt: Muster kommt in Text vor!
- aber: Wie sah der Treffer aus?
- interessanten Bereich markieren:

```
m/^From: (.+)/
m/^(\d+) (\d+) (\d+)$/
```

- Treffer landen in `$1`, `$2`, ...

```
$line =~ m/^(\d+) (\d+) (\d+)$/;
print "point at $1,$2,$3\n";
```

- Quantoren richtig setzen: `(\w)+` vs. `(\w+)`

Aufgaben

- In `romeo.txt` finden sich Regieanweisungen der Form

  ```
  ROMEO enters
  ```

  Extrahiere die Namen der Personen, die auf diese Weise die Bühne betreten.

- Regieanweisungen:

```
while ($line = <DRAMA>) {
    if ($line =~ /([A-Z]+\s*[A-Z\.]+)\s+enters/) {
        print "$1\n";
    }
}
```

Bei einfacheren Mustern werden z.B. „DOC LAWRENCE“ oder „MR.J“ nicht gefunden.

pattern capturing, cont.

- etwas eleganter: mit Zuweisung

```
($x, $y, $z) = ($line =~ m/^(\d+) (\d+) (\d+)$/);
```

- Muster muß vollständig passen

```
if (defined($x)) {
  print "point at $x,$y,$z\n";
}
```

- Unterschied zwischen grouping und capturing:

```
($dir, $who) = ($header =~ m/^(From|To): (.+)/);
      ($who) = ($header =~ m/^(?:From|To): (.+)/);
```

Aufgaben

- Lies `/etc/services` ein. Stelle die Informationen über die Dienste etwas umgangssprachlicher dar. Für die Zeile

  ```
  ftp             21/tcp
  ```

  soll folgende Ausgabe erzeugt werden:

  ```
  der Dienst "ftp" verwendet TCP auf Port 21
  ```

  Eventuelle weitere Informationen (Alias-Namen oder Kommentare) sollen ignoriert werden.

- Wie häufig kommen die Worte „Estragon“ und „Vladimir“ jeweils in `eric.txt` vor?

- Service-Liste formatieren:

```
while ($line = <SERV>) {
    ($serv, $port, $prot) = ($line =~ /^(\S+)\s+(\d+)\/(tcp|udp)/);
    if ($serv) {
        print "der Dienst \"$serv\" verwendet ",uc($prot),
              " auf Port $port.\n";
    }
}
```

- Namen zählen:

```
($estragon, $vladimir) = (0, 0);
while ($line = <DRAMA>) {
    @estragon = ($line =~ /Estragon/gi);
    $estragon += @estragon;
    @vladimir = ($line =~ /Vladimir/gi);
    $vladimir += @vladimir;
}
print "Found $estragon Estragons and $vladimir Vladimirs\n";
```

greedy matches

- Was passiert, wenn pattern nicht eindeutig ist?

```
$text = "aaaaaaaaaa";
($w1, $w2) = ($text =~ /(a+)(a+)/);
```

- ausprobieren:

```
/(a+)(a*)/
/(a*)(a+)/
/(a*)(a*)/
/(a?)(a*)/
/(a{2,4})(a*)/
```

- Setze ? hinter den ersten quantifier:

```
+?  *?  ??  {2,4}?
```

- greedy:

```
+ +:      >>aaaaaaaaa<<   >>a<<
+ *:      >>aaaaaaaaa<<   >><<
* +:      >>aaaaaaaaa<<   >>a<<
* *:      >>aaaaaaaaa<<   >><<
? *:      >>a<<   >>aaaaaaaaa<<
{2,4} *:  >>aaaa<<   >>aaaaaa<<
```

- non-greedy:

```
+? +:     >>a<<   >>aaaaaaaaa<<
+? *:     >>a<<   >>aaaaaaaaa<<
*? +:     >><<   >>aaaaaaaaaa<<
*? *:     >><<   >>aaaaaaaaaa<<
?? *:     >><<   >>aaaaaaaaaa<<
{2,4}? *: >>aa<<   >>aaaaaaaa<<
```

Text zerteilen

- `join`: Array-Elemente zu String vereinen
- entgegengesetzte Operation: `split`
- Trennung durch pattern:

```
$story = "In a hole in the ground there lived a hobbit.";
@words = split(/\s/, $story);
```

Aufgaben

- Trenne den Satz

  „**In a hole in the ground there lived a hobbit.**“

  mit den folgenden Mustern. Welche „Worte“ entstehen dabei?

```
/ /
//
/\s*/
/\b/
/\B/
```

  Wie „groß“ sind die patterns, an denen getrennt wird?

- verschiedene Trennmuster:

```
/ /     In|a|hole|in|the|ground|there|lived|a|hobbit.
//      In| |a| |h|o|l|e| |i|n| |t|h|e| |g|r|o|u|n|d|
                 |t|h|e|r|e| |l|i|v|e|d| |a| |h|o|b|b|i|t|.
/\s*/   In|a|h|o|l|e|i|n|t|h|e|g|r|o|u|n|d|
                 t|h|e|r|e|l|i|v|e|d|a|h|o|b|b|i|t|.
/\b/    In| |a| |hole| |in| |the| |ground| |there|
                 |lived| |a| |hobbit|.
/\B/    I|n a h|o|l|e i|n t|h|e g|r|o|u|n|d t|h|e|r|e
                 l|i|v|e|d a h|o|b|b|i|t.
```

Das pattern `/ /` ist ein Zeichen groß. Die patterns `//`, `/\b/` und `/\B/` haben keine Ausdehnung. Das pattern `\s*` hat variable Größe.