Linux, Knoppix, Mac OS X, Open Source: Vorteile von Unix et al. in Chemie & Biologie

Teil 6: Textsuche mit Schuss: Reguläre Ausdrücke

Röbbe Wünschiers

In Teil 3 der Unix/Linux Serie (CLB 01/04) haben Sie bereits Wildcards kennen gelernt. Sie dienten uns als Jokerzeichen bei der Suche oder dem Kopieren, Verschieben, Umbenennen und Löschen von Dateien. So können wir mit dem Befehl `rm *.txt` alle Dateien im aktuellen Verzeichnis löschen, welche die Endung *.txt* besitzen. In Abbildung 4 in Teil 3 (CLB 01/04) sind die verfügbaren Wildcards aufgelistet. Wildcards sind hervorragend geeignet um Dateinamen zu beschreiben – dafür wurden sie entwickelt. Sie sind aber ungeeignet, wenn Sie nach komplizierten Textmustern innerhalb von Textdateien suchen möchten. Dafür gibt es reguläre Ausdrücke (*regular expressions*), die ich Ihnen in dieser Ausgabe vorstellen möchte.

Wenn Sie sich im Umgang mit regulären Ausdrücken vertraut machen, dann haben Sie eines der mächtigsten Unix-Werkzeuge in der Hand die es gibt. Reguläre Ausdrücke sind universell einsetzbar. Fast jedes Programm und jede Programmiersprache erlaubt den Einsatz von regulären Ausdrücken. Selbst in Anwendungsprogrammen wie OpenOffice (das frei erhältliche Pendant zu MS-Office) erlaubt den Einsatz von regulären Ausdrücken bei der Suchfunktion. Keine Frage, dass auch `vim`, der Texteditor dem wir in der letzten Ausgabe begegnet sind, reguläre Ausdrücke versteht.

Reguläre Ausdrücke sind aus zwei Zeichentypen aufgebaut. Spezialzeichen, wie z.B. der Stern (`*`), heißen *Metazeichen*, alle anderen Zeichen, meist Buchstaben, sind *Literale*. Bei regulären Ausdrücken geht die Verwendung von Metazeichen weit über die Möglichkeiten von Wildcards hinaus. Daher sehen reguläre Ausdrücke meist auch sehr abschreckend aus: `([^□])\1\1` (bitte beachten Sie, dass das Quadrat `(□)` stellvertretend für ein Leerzeichen steht). Dieser reguläre Ausdruck spürt die dreifache Wiederholung eines beliebigen Zeichens mit Ausnahme des Leerzeichens auf: z.B. das Tripel-f in Sauerstoffflasche.

Reguläre Ausdrücke und Programme

Wie bereits gesagt, sind reguläre Ausdrücke universell einsetzbar und werden von den meisten Programmen verstanden. Wir werden uns im Folgenden auf die Anwendung zwei sehr einfacher Programme beschränken: `egrep` und `sed`. In der letzten Folge haben sie in Terminal 3 (CLB 02/04) bereits `grep` kennen gelernt. Mit `grep` können wir in einer Textdatei nach Zeichen oder Wörtern suchen. Alle Zeilen, auf die das Suchmuster zutrifft, werden auf den Bildschirm ausgegeben. `egrep` ist eine Erweiterung von `grep`, die das Suchmuster als regulären Ausdruck interpretiert.

Der Zeilen orientierte Editor `sed` (*stream editor*) bietet nicht nur die Möglichkeit in Dateien nach Text zu suchen, sondern diesen auch zu editieren. Im Gegensatz zu `vim` bietet `sed` kein Editierfenster und lässt die Quelldatei unberührt. Die Ausgabe erfolgt auf den Bildschirm. Der große Vorteil von `sed` ist, dass er rasant schnell ist und den Arbeitsspeicher schont, da die Textdatei zeilenweise bearbeitet wird. Dies interessiert z.B. diejenigen, die mit mehreren Megabyte großen Datendateien (von Messgeräten oder ähnlichem) hantieren müssen. `sed` ist dann das Werkzeug der Wahl um z.B. bei allen Zahlen das Dezimalkomma durch einen Dezimalpunkt oder alle Semikolons durch einen Tabulator zu ersetzten.

Die Auswahl der Programme `egrep` und `sed` ist nicht ganz zufällig getroffen. Vielmehr bilden beide Programme wichtige Meilensteine bei der Anwendung regulärer Ausdrücke in der elektronischen Datenverarbeitung.

Reguläre Ausdrücke und Neurone

Der Keim zu regulären Ausdrücken wurde in den frühen 40er Jahren von den beiden Neurophysiologen Warren McCulloch und Walter Pitts angelegt. Sie entwickelten Modelle für die Funktion des Nervensystems auf der Ebende der Neurone [1]. Der Mathematiker Stephen Kleene formulierte einige Jahre später für diese Modelle eine Algebra, die er reguläre Mengen nannte. Zusätzlich entwarf er eine Notation zu der Algebra und nannte diese regular expressions, also reguläre Ausdrücke. In den 50er und 60er Jahren waren reguläre Ausdrücke ein beliebtes Forschungsobjekt der theoretischen Mathematik. Der wahrscheinlich erste Einsatz regulärer Ausdrücke bei Computern wird in einem Artikel mit dem Titel “Regular Expression Search Algorithm“ von Ken Thompson, neben Dennis Ritchie einem der Entwickler von Unix, beschrieben [2]. Diese Arbeiten führten im weiteren Verlauf zur Entwicklung des ersten Unix Editors, ed

(editor). Auf der Grundlage von ed wurde später der Zeilen orientierte Editor sed entwickelt. Sowohl ed als auch sed kennen einen Befehl, der Zeilen ausgibt, die auf einen regulären Ausdruck passen: g/regulärer ausdruck/p (lies: global/regular expression/print). Diese Funktion wurde so häufig benötigt, das einige Programmierer in den 70er Jahren dafür ein eigenständiges Programm entwickelt haben, grep bzw. egrep.

Die Syntax der regulären Ausdrücke

Die Metazeichen der regulären Ausdrücke lassen sich in bestimmte Klassen einteilen, die nachfolgend aufgeführt sind. Um mit regulären Ausdrücken Spaß zu haben muss man sie ausprobieren. Daran führt kein Weg vorbei. Kochen Sie sich eine Kanne Rooibusch Tee, kreieren Sie sich Zeichenketten und wenden Sie darauf reguläre Ausdrücke an. Eine elegante Hilfe bietet hierfür der Texteditor `vim` (siehe unten).

In der Tabelle 1 finden Sie die wichtigsten Metazeichen aus denen reguläre Ausdrücke aufgebaut sind. In den nachfolgenden Beispielen werden exemplarisch einzelnen Elemente näher erläutert. Als Beispieldatei, in der wir mit `egrep` und `sed` suchen werden, sollten Sie die Datei namens *beispiel.txt* mit folgenden Inhalt erstellen:

Terminal 1

```
01    $ cat beispiel.txt
02    Genomics
03    Genom
04    Proteomics
05    123; 43; 2344;7445; 34,
      2323.912; 4711
06    2374
07    Methan
08    Ethan
09    C6H12O6
10    $
```

In Terminal 1 zeigen wir den Inhalt der Datei *beispiel.txt* auf dem Bildschirm an. Die Datei enthält also die Zeilen 2-9.

Einzelzeichen Metazeichen
Einzelzeichen Metazeichen sind Joker für ein einzelnes Zeichen. Das folgende Beispiel in Terminal 2 verdeutlicht dies.

Terminal 2

```
01    $ egrep 'Ge.om' beispiel.txt
02    Genomics
03    Genom
04    $
```

Alle Zeilen der Datei *beispiel.txt* welche die Zeilenfolge G, e, beliebiges Zeichen, o und m enthalten werden auf den Bildschirm ausgegeben. Achtung: Unix unterscheidet zwischen Groß- und Kleinbuchstaben! In eckigen Klammern können Sie mehrere Zeichen zur Auswahl stellen.

Terminal 3

```
01    $ egrep '[eE]than' beispiel.
      txt
02    Methan
03    Ethan
04    $
```

Quantifizierer
Mittels der Quantifizierer können Sie festlegen, wie oft ein bestimmtes Zeichen (oder eine Zeichenauswahl) auftreten muss.

Terminal 4

```
01    $ egrep '4*' beispiel.txt
02    Genomics
03    Genom
04    Proteomics
05    123; 43; 2344;7445; 34;
      2323.912; 4711
06    2374
07    Methan
08    Ethan
09    C6H12O6
10    $ egrep '4+' beispiel.txt
11    123; 43; 2344;7445; 34;
      2323.912; 4711
12    2374
13    $ egrep '4{2}' beispiel.txt
14    123; 43; 2344;7445; 34;
      2323.912; 4711
15    $
```

In Zeile 1 von Terminal 4 suchen wir alle Zeilen in denen die 4 ein oder kein Mal vorkommt. Das sind alle Zeilen. In Zeile 10 suchen wir nach allen Zeilen in der Datei *beispiel.txt* in denen die Zahl 4 mindestens einmal vorkommt. Dies betrifft zwei Zeilen. Schließlich suchen wir in Zeile 13 alle Zeilen, in denen die 4 genau zweimal in Folge auftritt. Weitere Quantifizierer können Sie der Tabelle 1 entnehmen.

Anker
Sehr hilfreich sind oftmals auch die Anker, die bestimmte Positionen innerhalb einer Zeile festlegen. Insbesondere die Anker für den Beginn und das Ende einer Zeile finden häufig Anwendung. So trifft der reguläre Ausdruck `^$` auf alle leeren und `^.+$` auf alle nicht leeren Zeilen zu.

Escape Zeichen
Von besonderer Bedeutung ist das Escape Zeichen (Backslash). Es wird einem Metazeichen vorangestellt, wenn es als Literal erkannt werden soll. Schauen wir uns ein Beispiel an, das gleichzeitig den Streameditor `sed` vorstellt.

Terminal 5

```
01    $ sed 's/\./,/g' beispiel.txt
02    Genomics
03    Genom
04    Proteomics
05    123; 43; 2344;7445; 34;
      2323,912; 4711
06    2374
07    Methan
08    Ethan
09    C6H12O6
10    $
```

Die Syntax von **sed** gleicht der von **egrep**. Auf den Programmaufruf (**sed**) folgt in einfachen Hochzeichen(**'...'**) ein Befehl. Als letztes folgt der Dateiname auf den der Befehl abgewendet werden soll. Die Ausgabe erfolgt wiederum auf dem Bildschirm, während die bearbeitete Datei unverändert bleibt. Um die Änderungen zu speichern muss die Ausgabe von **sed** in eine Datei umgeleitet werden (>, siehe Terminal 1 in Teil 3, CLB 01/04). Wir werden hier nur einen Befehl von **sed** nutzen: ersetzen (*substitute*). Die Syntax ist: **s/A/B/g** (lies: *substitute A by B globally*). *A* beizeichnet das Suchmuster (einen regulären Ausdruck oder einfach nur Text), der durch *B* (Text) ersetzt wird. Das abschließende *g* steht für *globally* und besagt, dass alle Treffer von *A* durch *B* ersetzt werden sollen. In unserem Falle ist *A* gleich \. und *B* gleich ,. Das bedeutet, dass alle Punkte durch Kommata ersetzt werden. Würden wir dem Punkt kein Escape Zeichen voranstellen, dann würde der Punkt dem regulären Ausdruck „ein beliebiges Zeichen" entsprechen – alle Zeichen würden dann durch Kommata ersetzt werden. Probieren Sie es einmal aus. Der Erstetzungsbefehl von **sed** wird sehr häufig benötigt. Mit ihm können sehr einfach alle Dezimalkommata durch Dezimalpunkte oder Tabulatoren durch Leerzeichen usw. ersetzt werden.

Zwischenspeicher

Wie die Speichertaste eines Taschenrechners, so bieten auch reguläre Ausdrücke die Möglichkeit zur Speicherung von Suchmustern an. Dazu dienen die runden Klammern. Wird ein regulärer Ausdruck in runden Klammern geschrieben, dann wird der übereinstimmende Text gespeichert. Aufgerufen wird der Speicher mit **\1**. insgesamt stehen 9 Speicher zur Verfügung.

Terminal 6

```
01    $ sed 's/\(.*an\)/## \1 ##/g'
      beispiel.txt
02    Genomics
03    Genom
04    Proteomics
05    123; 43; 2344;7445; 34;
      2323.912; 4711
06    2374
07    ## Methan ##
08    ## Ethan ##
09    C6H12O6
$
```

Vermutlich sind Sie etwas verwirrt, weil den runden Klammern in Zeile 1 ein Escape Zeichen vorangestellt ist. Das ist eine Eigenart von **sed**. Eingeschlossen in den runden Klammern ist der reguläre Ausdruck **.*an**, also eine beliebige Anzahl (*) beliebiger Zeichen (.) plus **an**. Der zutreffenden Textstring wird in der Variable 1 gespeichert und über **\1** wieder aufgerufen. Die Ersetzung die wir vornehmen ist eine Formatierung, wie das Ergebnis in den Zeilen 7 und 8 zeigt.

Reguläre Ausdrücke und vim

Tabelle 1 gibt Ihnen eine Übersicht über Metazeichen die in regulären Ausdrücken verwendet werden können. Wie wir gesehen haben, spielen reguläre Ausdrücke bei der Textsuche und -formatierung eine große Rolle und sind unglaublich vielseitig einsetzbar. Um den maximalen Nutzen aus regulären Ausdrücken zu ziehen, müssen Sie sie häufig anwenden. Eine elegante Möglichkeit den Umgang mit regulären Ausdrücken zu üben bietet der Texteditor **vim**, den wir bereits in Teil 4 (CLB 02/04) besprochen haben. Starten Sie **vim** und geben einen kleinen Text zum üben ein. Wechseln Sie nun in den Kommandomodus und geben folgenden Befehl ein: **:set hls** und drücken ENTER. Dadurch wird die Markierung von Suchergebnissen aktiviert (*set highlight search*). Immer noch im Kommandomodus drücken Sie jetzt / und ENTER. Sie sehen am unteren Rand das / Zeichen und können jetzt nach Text suchen – dabei können Sie reguläre Ausdrücke einsetzten. Wenn Sie nun ENTER drücken, werden alle mit dem Suchmuster übereinstimmende Textstellen markiert. Um eine neue Suche zu starten geben sie wieder / ENTER gefolgt von dem Suchmuster ein. Auf diese Weise lassen sich reguläre Ausdrücke prima üben.

Reguläre Ausdrücke in der Genomforschung

Im wissenschaftlichen Bereich finden reguläre Ausdrücke z.B. bei der Genomanalyse breite Anwendung. Das Genom eines Organismus ist im einfachsten Sinne ein langer Text der aus den vier Buchstaben A, C, T und G besteht [3]. An bestimmte Sequenzabschnitte können so genannte Transkriptionsfaktoren binden. Sie regulieren die Aktivität eines Gens, bestimmen also ob ein Gen abgelesen und das entsprechende Protein kodiert wird oder nicht. Hat man einmal experimentell die DNA-Bindungsstelle eines Transkriptionsfaktors bestimmt, so kann man, das Vorhandensein der gesamten Genomsequenz vorausgesetzt, nach allen potentiellen Bindungsstellen in diesem Genom suchen. Gene, die auf die so detektierten Bindungsstellen folgen, stehen also potentiell unter der Kontrolle

des entsprechenden Transkriptionsfaktors. Nehmen wir das folgende Beispiel. Das Protein NtcA (*nitrogen control factor A*) reguliert den Stickstoffhaushalt bei vielen Bakterien. Für ein bestimmtes Bakterium, dessen Genom bereits sequenziert wurde, hat man folgende Bindungsstellen empirisch nachgewiesen: $TGTN_9ACA$ und $TGTN_{10}ACA$, wobei das N für ein beliebiges Nukleotid steht und der Index die Anzahl der Nukleotide angibt. Nach diesem Muster kann man mit dem Kommando `egrep 'TGT.{9,10}ACA' genome.file` suchen. Da die Sequenzsuche zum täglichen Geschäft in der Molekularbiologie zählt, stehen bereits eine Reihe hilfreicher kleiner kostenloser Unix-Programme zum Download bereit. Mit `agrep` (*approximate grep*) steht z.B. ein Werkzeug zur Verfügung, das Fehler (Mutationen) zulässt [4]. `tacg` ist ein Programm, welches speziell für Molekularbiologen entwickelt wurde und auf die Suche von Mustern in DNA-Sequenzen spezialisiert ist [5].

Neue Befehle in dieser Ausgabe

`egrep` `grep` mit regulären Ausdrücken
`sed` Streameditor

Literatur

[1] McCulloch & Pitts (1943) Bulletin of Math. Biophysics 5
[2] Thompson (1968) Communications of the ACM, Vol. 11, No. 6
[3] Wünschiers (2003) CLB 54: 8-13
[4] Wu & Manber (1992) Communications of the ACM, Vol. 35, No. 10
[5] Mangalam (2002) BMC Bioinformatics 3: 8

Tabelle 1: Metazeichen. Die Bausteine regulärer Ausdrücke

Einzelzeichen Metazeichen (*single-letter meta character*)	
`.`	ein beliebiges Zeichen
`[aA]`	eines der angegebenen Zeichen (hier a oder A)
`[^abc]`	ein beliebiges außer der angegebenen Zeichen
`[0-9a-z]`	beliebiges Zeichen zwischen 0 bis 9 und a bis z
Quantifizierer (*quantifier*)	
`?`	kein oder ein Mal
`+`	mindestens ein Mal
`*`	beliebig oft
`{num}`	genau *num* Mal
`{min,}`	mindestens *min* Mal
`{min,max}`	mindestens *min* und höchstens *max* Mal
Anker (*anchor*)	
`^`	Beginn einer Zeile
`$`	Ende einer Zeile
`\<`	Beginn eines Wortes
`\>`	Ende eines Wortes
Escape Zeichen (*escape character*)	
`\`	wird Metazeichen vorangestellt, wenn sie als Literal erkannt werden sollen
Auswahl (alternation)	
`(RA1|RA2)`	Einer der regulären Ausdrücke *RA1* oder *RA2* muß zutreffen
Speicher (*back references*)	
`(RA1)`	Der zum regulären Ausdruck *RA1* passende Text wird gespeichert und kann mit *\1* wieder aufgerufen werden. Es können mehrere Klammern eingesetzt werden.
`\n`	Aufruf des gespeicherten Suchmusters; *n* ist eine ganze Zahl
Zeichenklassen (*character classes*)	
`[:alnum:]`	beliebiges alphanumerisches Zeichen (0-9, A-Z, a-z)
`[:digit:]`	beliebige Zahl
`[:alpha:]`	beliebiger Buchstabe (A-Z, a-z)
`[:upper:]`	beliebiger Großbuchstabe (A-Z)
`[:lower:]`	beliebiger Kleinbuchstabe (a-z)
`[:blank:]`	Leer- oder Tabulatorzeichen
`[:space:]`	Leerzeichen
`[:punct:]`	Punktierung (. , " ' ? ! ; :)
Sonderzeichen (*special characters*)	
`\n`	neue Zeile
`\t`	Tabulator
`\.`	Punkt
`\*`	Stern