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

Teil 11: Awk – Trennzeichen & Muster

Röbbe Wünschiers

Awk-Skripte bestehen grundsätzlich aus Mustern und Befehlen. Durch die Muster wird bestimmt, auf welche Zeilen der Eingabedatei die Befehle angewendet werden sollen. Dies ist extrem hilfreich: Synthesereaktoren in der chemischen Industrie, um ein beliebiges Beispiel zu nennen, werden rund um die Uhr überwacht. Denken wir uns ein Messgerät, das im Minutentakt Uhrzeit, Temperatur, pH-Wert und Druck aufzeichnet und täglich in einer Ausgabedatei speichert – das sind 86.400 Datensätze pro Tag. Versuchen Sie gar nicht erst die Datei in MS-Excel mit seinem 65.536 Zeilenlimit zu öffnen. Mit Awk können Sie ein Muster bestimmen, das Ihnen die Zeiten und Betriebsdaten ausgibt, als die Temperatur über und der pH-Wert unter einem Grenzwert lagen. Dazu müssen Sie allerdings wissen, durch welche Trennzeichen die Datensätze voneinander separiert sind.

Von unserem Umgang mit Texten sind wir gewohnt, dass Zeilen durch Umbrüche und Wörter durch Leerzeichen voneinander getrennt sind. Dies ist daher auch die Standardeinstellung von Awk. Allerdings trifft man häufig auf Dateien, die nicht diesem Muster entsprechen. Nicht selten werden z.B. Doppelpunkte als Trennzeichen für einzelne Felder (Wörter) in Datensätzen verwendet. Im folgenden Abschnitt werden wir uns daher zunächst damit beschäftigen, wie wir Awk auf neue Trennzeichen umstellen können.

Trennzeichen

Wie in der letzten Ausgabe bereits besprochen bearbeitet Awk Dateien zeilenweise und trennt jede Zeile wiederum in Felder (Wörter) auf. Doch wie sind Zeilen und Felder definiert? Standardmäßig werden Zeilen durch Umbrüche (Zeichenfolge: `\n`) und Wörter durch Leerzeichen oder Tabulatoren (Zeichenfolge: `\t`) getrennt. Das Beispiel in dem nachfolgenden Terminal soll Ihnen ein Gefühl für die Anwendung von Trennzeichen vermitteln.

Tabelle 1: Sonderzeichen – Diese Sonderzeichen werden von den meisten Linuxprogrammen verstanden.

Zeichenfolge	Bedeutung
\a	Alarmton (*alarm*)
\b	Rückschritt (*backstep*)
\n	Zeilenumbruch (*new* line)
\r	Wagenrücklauf (*return*)
\t	Tabulator (*tabulator*)
\\	Backslash
\"	Anführungszeichen

Terminal 1

```
01    $ echo -e "Wort1\nWort2"
02    Wort1
03    Wort2
04    $ echo -e "Wort1\tWort2"
05    Wort1    Wort2
06    $ echo -e "Wort1\t\tWort2"
07    Wort1            Wort2
08    $ echo -e "Wort1\aWort2"
09    Wort1Wort2
10    $ echo "Wort1\tWort2"
11    Wort1\tWort2
12    $
```

Sie haben es wahrscheinlich gemerkt: `\a` in Zeile 8 ist kein Trennzeichen, sondern gibt einen Alarmton aus. Den Befehl `echo` versorgen wir mit der Option `-e` (*extended*, erweitert). Nur mit dieser Option werden die Sonderzeichen als solche erkannt, wie in den Zeilen 10 und 11 zu sehen ist. In Tabelle 1 sind weitere erlaubte Sonderzeichen dargestellt; probieren Sie sie mal aus.

Zurück zu Awk. Was können Sie tun, wenn Ihr Text andere Trennzeichen verwendet; wenn z.B. die Zeilen durch Pluszeichen und die einzelnen Felder nicht durch Leerzeichen, sondern durch Minuszeichen, getrennt sind. In Terminal 2 ist beispielhaft ein Auszug einer solchen Datei gezeigt. Jeder durch ein Pluszeichen getrennter Datensatz besteht aus vier, von Minuszeichen getrennten, Feldern: Zeit, pH-Wert, Temperatur und Druck. Der erste Datensatz dient lediglich als Legende.

Terminal 2

```
01    $ cat data.txt
02    time-pH-temp-pres+23:01-13
      .4-73.4-12+23:02-13.5-73.3
      -12+23:03-13.4-73.1-12+23:
      04-13.6-72.6-12+23:05-13.7
      -71.1-12+23:06-14.0-70.4-1
      1+23:07-14.1-71.4-12+23:08
      -13.8-72.3-12+23:09-13.7-7
      3.1-12+23:10-13.3-73.5-12+
      23:11-13.6-73.3-12
03    $
```

Diese, sehr unübersichtliche, Datei können Sie unter *http://www.clb.de/computare.htm* herunterladen. Um Awk die verwendeten Trennzeichen in der Eingabedatei mitzuteilen gibt es zwei Variablen (Tabelle 2): *RS* (*record separator*, Zeilentrennzeichen) und *FS* (*field separator*, Feldtrennzeichen). Diese Variablen müssen vor der Ausführung von Befehlen definiert werden. Dies geschieht am besten in einem `BEGIN`-Block. Wie in der letzten Ausgabe besprochen (CLB 08/2004, Abb. 3), ist der `BEGIN`-Block optional und wird vor der Bearbeitung der zu editierenden Eingabedatei ausgeführt.

Terminal 3

```
01    $ awk 'BEGIN{RS="+"; FS="-"}
      {print $0}' data.txt
02    time-pH-temp-pres
03    23:01-13.4-73.4-12
04    23:02-13.5-73.3-12
05    23:03-13.4-73.1-12
06    23:04-13.6-72.6-12
07    23:05-13.7-71.1-12
08    23:06-14.0-70.4-11
09    23:07-14.1-71.4-12
10    23:08-13.8-72.3-12
11    23:09-13.7-73.1-12
12    23:10-13.3-73.5-12
13    23:11-13.6-73.3-12
14    $
```

Terminal 3 zeigt ein einfaches Awk-Skript, das neue Trennzeichen zuordnet und jeweils die gesamte Zeile ausgibt. Im `BEGIN`-Block werden die Trennzeichen den Variablen *RS* und *FS* zugewiesen. Diese beiden Befehle werden durch ein Semikolon voneinander getrennt. Der Hauptteil des Skriptes besteht aus dem Befehl `print $0`. Sie erinnern sich? Die aktuell bearbeitete Zeile ist in der Variablen *$0*, und die einzelnen Felder in den Variablen *$1*, *$2*, usw. gespeichert (siehe auch CLB 08/2004, Abb. 2). Sollen nur die Temperaturwerte ausgegeben werden, so lautet der Befehl `print $3`. Dies ist in Terminal 4 gezeigt.

Terminal 4

```
01    $ awk 'BEGIN{RS="+"; FS="-"}
      {print $3}' data.txt
02    temp
03    73.4
04    73.3
05    73.1
06    72.6
07    71.1
08    70.4
09    71.4
10    72.3
11    73.1
12    73.5
13    73.3
14    $
```

Variable	Bedeutung	Standardeinstellung
RS	Zeilen-Trennzeichen bei der Eingabedatei (*record separator*)	Zeilenumbruch (\n)
FS	Feld-Trennzeichen bei der Eingabedatei (*field separator*)	Leerzeichen o.Tab.(\t)
ORS	Zeilen-Trennzeichen bei der Datenausgabe (*output record separator*)	Zeilenumbruch (\n)
OFS	Feld-Trennzeichen bei der Datenausgabe (*output field separator*)	Leerzeichen

Tabelle 2: Trennzeichen – Vier Variablen bestimmen, welche Zeichen Awk als Feld- und Zeilentrennzeichen bei der Dateneingabe bzw. -ausgabe verwendet.

Erst in Terminal 4 macht sich die Veränderung des Wertes von *FS* bemerkbar und die Temperaturwerte werden als drittes Feld erkannt. Auf die gezeigte Art und Weise lassen sich beliebige Trennzeichen für die Eingabedatei definieren. Wie sieht es aber mit der Datenausgabe aus? Die Beispiele in den Terminals 3 und 4 zeigen, das der `print` Befehl standardmäßig einen Zeilenumbruch an den auszugebenden Text hängt. Ebenso werden auszugebende Texte, die durch Kommata getrennt sind, automatisch von Leerzeichen getrennt. Ein Beispiel ist im folgenden Terminal gezeigt.

Terminal 5

```
01    $ awk 'BEGIN{print "Chemie",
       "Biologie"; print "Natur"}'
02    Chemie Biologie
03    Natur
04    $
```

Unser Awk-Skript in Zeile 1 verarbeitet keine Eingabedatei. Daher müssen wir das Skript in einen `BEGIN`-Block schreiben und können auf die Angabe einer Eingabedatei verzichten. Das Skript verwendet den `print` Befehl zweimal. Die Ausgabe in der Zeile 2 gehört zum ersten `print` Befehl, die Ausgabe in der Zeile 3 zum zweiten `print` Befehl. Die Werte der Variablen *ORS* (*output record separator*) und *OFS* (*output field separator*) bestimmen die Verwendung von Ausgabe-Zeilen- und Ausgabe-Feldtrennzeichen (Tabelle 2). Wie in Terminal 6 gezeigt entspricht ihre Verwendung der von *RS* und *FS*.

Terminal 6

```
01    $ awk 'BEGIN{ORS=" : ";
       OFS="|"; print "Chemie",
       "Biologie"; print "Natur"}'
02    Chemie|Biologie : Natur : $
03    $ awk 'BEGIN{ORS=" : ";
       OFS="|"; print "Chemie",
       "Biologie"; ORS="\n";
       print "Natur"}'
04    Chemie|Biologie : Natur
05    $
```

In Zeile 1 verwenden wir die Kombination Leerzeichen-Doppelpunkt-Leerzeichen als Ausgabe-Zeilentrennzeichen und den vertikalen Strich als Ausgabe-Feldtrennzeichen. Die Ausgabe ist in Zeile 2

zu sehen. Da wir keinen Zeilenumbruch mehr verwenden, erscheint das Eingabeprompt (`$`) direkt hinter der ausgegebenen Zeile. Durch drücken der ENTER Taste gelangen wir wieder in eine neue Zeile. Wie lässt sich das ändern? Eine mögliche Lösung ist in der Zeile 3 gezeigt. Vor der letzten Anwendung des `print` Befehls setzten wir die Variable *ORS* wieder auf ihren Standardwert, `\n`.

Ich hoffe die vorangegangenen Beispiele haben den Umgang mit Trennzeichen beim Einlesen und der Ausgabe von Daten verdeutlicht. Im nächsten Abschnitt möchte ich Ihnen vorstellen, wie Zeilen mit bestimmten Information extrahiert werden können.

Muster

Die Beschreibung von Mustern (*pattern*) um Zeilen auszuwählen, auf die Awk-Befehle angewendet werden sollen, entspricht im weitesten Sinne einer Suchaktion. Allerdings ist diese Suche nicht auf Zeilen beschränkt die einen bestimmten Text enthalten. Vielmehr können reguläre Ausdrücke ebenso angewendet werden wie logische Analysen von der Form: ist der Wert in Feld 3 größer als 250? Egal welches Muster verwendet wird, es befindet sich immer unmittelbar von dem Hauptteil eines Awk-Skriptes (siehe CLB 08/2004, Abb. 1 und 3). Außerdem können mehrere Muster mit den Boolschen Ausdrücken *und* (`&&`), *oder* (`||`) und *nicht* (`!`) zusammengesetzt werden.

Bevor wir uns einige Muster genauer ansehen, sollten wir die recht unübersichtliche Datendatei *data.txt* umformatieren und in einer Datei namens *data2.txt* speichern. Die Zeilen sollen durch Umbrüche und die Felder durch Leerzeichen getrennt werden. Wir erinnern uns: dies sind die Standardeinstellungen des `print` Kommandos. Das macht die Sache einfach.

Terminal 7

```
01    $ awk 'BEGIN{RS="+"; FS="-"}
      {print $1,$2,$3,$4}'
       data.txt > data2.txt
02    $ cat data2.txt
03    time pH temp pres
04    23:01 13.4 73.4 12
05    23:02 13.5 73.3 12
06    23:03 13.4 73.1 12
07    23:04 13.6 72.6 12
08    23:05 13.7 71.1 12
09    23:06 14.0 70.4 11
10    23:07 14.1 71.4 12
11    23:08 13.8 72.3 12
12    23:09 13.7 73.1 12
13    23:10 13.3 73.5 12
14    23:11 13.6 73.3 12
15    $
```

In Zeile 1 verwenden wir ein kleines Awk-Skript für die Umformatierung und speichern das Ergebnis mit Hilfe einer *redirection* (`>`) in die Datei *data2.txt*. Diese geben wir in Zeile 2 mit Hilfe des `cat` Befehls aus, um uns die Struktur der Daten ansehen zu können. Mit dieser Datei werden wir nun die Funktion verschiedener Muster testen.

Reguläre Ausdrücke

Regulären Ausdrücken werden wir immer wieder begegnen. Es lohnt sich auf alle Fälle, sich eingehend mit ihnen vertraut zu machen (siehe auch CLB 04/2004). Reguläre Ausdrücke bilden sehr vielseitige Muster. Sie werden immer in Schrägstrichen (`/`) eingeschlossen.

Terminal 8

```
01    $ awk '/4\../{print $0}'
       data2.txt
02    23:06 14.0 70.4 11
03    23:07 14.1 71.4 12
04    $ awk '/2\../||/4\../
      {print $0}' data2.txt
05    23:04 13.6 72.6 12
06    23:06 14.0 70.4 11
07    23:07 14.1 71.4 12
08    23:08 13.8 72.3 12
09    $
```

Mit dem Skript in Zeile 1 suchen wir alle Zeilen, in denen eine "Vier-Punkt-Irgendwas" vorkommt. Der Punkt stellt ein beliebiges Zeichen dar. Wenn wir nach einem Punkt suchen, dann müssen wir ihm das Escapezeichen, den Backslash, voranstellen. Durch den Befehl `print $0` wird jede Zeile der Datei *data2.txt* ausgegeben, auf die dieses Muster zutrifft. In Zeile 4 kombinieren wir zwei Muster mit einem logischen *oder*. Eine Zeile wird ausgegeben, wenn sie entweder mit dem regulären Ausdruck `2\..` oder dem regulären Ausdruck `4\..` übereinstimmt.

Terminal 9 zeigt, wie Sie testen können ob ein einzelnes Feld, anstatt der gesamten Zeile, mit einem regulären Ausdruck übereinstimmt (`~`) oder nicht übereinstimmt (`!~`).

Terminal 9

```
01    $ awk '$3~/\.4/{print $0}'
       data2.txt
02    23:01 13.4 73.4 12
03    23:06 14.0 70.4 11
04    23:07 14.1 71.4 12
05    $ awk '$3!~/\.4/{print $0}'
       data2.txt
06    time pH temp pres
07    23:02 13.5 73.3 12
08    23:03 13.4 73.1 12
09    23:04 13.6 72.6 12
10    23:05 13.7 71.1 12
11    23:08 13.8 72.3 12
12    23:09 13.7 73.1 12
13    23:10 13.3 73.5 12
14    23:11 13.6 73.3 12
15    $
```

Mit regulären Ausdrücken haben Sie bereits ein kraftvollen Werkzeug zur Beschreibung von Zeilen die Sie bearbeiten möchten zur Hand. Insbesondere für Textmuster sind sie bestens geeignet. Die Beschreibung von Zahlenmustern geht jedoch noch einfacher.

Relationales Zahlenmuster

Sie bekommen die Aufgabe, aus der Datendatei alle Zeiten auszugeben, an denen der Betriebsdruck unter 12 Bar lag und der pH-Wert 14 oder mehr Einheiten betrug. Nichts einfacher als das.

Terminal 10

```
01    $ awk '$4<12&&$2>=14
      {print $1}' data2.txt
02    23:06
03    $
```

Sie können natürlich alle möglichen Relationen wie größer (>), größer gleich (>=), ist gleich (==), ist ungleich (!=) usw. anwenden. Beachten Sie, dass die „ist gleich“ Relation durch zwei unmittelbar aufeinander folgende Gleichheitszeichen dargestellt wird.

Natürlich gibt es noch mehr Möglichkeiten der Benennung von Mustern. Die hier vorgestellten Beispiele sollen Ihnen einen Eindruck von der Anwendung von Mustern vermitteln. Wie immer finden Sie noch mehr Details in den Manualseiten von Awk, die Sie mit dem Befehl `man awk` oder `man gawk` aufrufen können. Noch ein Wort zu guter Letzt (hoffentlich nicht zu spät): Awk ist ziemlich gutmütig was die Anwendung von Leerzeichen angeht. Eines zu viel oder zu wenig wird selten zu Problemen führen. Seien Sie daher ruhig großzügiger als ich es, aus Platzgründen, bin. Machen Sie auch regen Gebrauch von Kommentaren. Jede Zeile die mit einem #-Zeichen beginnt, wird von dem Awk-Interpreter ignoriert.