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

Teil 13: Awk – Daten Eingabe, Ausgabe und Formatierung

Röbbe Wünschiers

In allen bislang vorgestellten Awk-Skripten wurde eine einzelne Eingabedatei bearbeitet. Die Ausgabe der editierten Datei erfolgte auf den Bildschirm. In dieser Ausgabe möchte ich Ihnen zeigen, wie Sie mehrere Eingabedateien bearbeiten und direkt in Ausgabedateien schreiben können. Des weiteren werden Sie Möglichkeiten kennen lernen, die Ausgabe zu formatieren.

In Teil 11 dieser Serie (CLB 09/04) haben wir mit der Datei *data.txt* gearbeitet, die Betriebsdaten (Uhrzeit, pH-Wert, Temperatur, Druck) eines Reaktors enthält. Nehmen wir an, dass Sie von zwei weiteren Messstationen je eine Datei erhalten, *dataCO2.txt* bzw. *dataO2.txt*, die weitere Betriebsdaten (Uhrzeit und CO_2-Konzentration bzw. Uhrzeit und O_2-Konzentration) enthalten. Es gilt nun die Inhalte dieser drei Dateien zu einer Datei zusammenzufügen. Außerdem sollen aus der Datei *data.txt* der pH-Wert und die Temperatur in eigene Dateien, *dataPH.txt* bzw. *dataTEMP.txt*, geschrieben werden. Um diese Aufgabe zu erfüllen, müssen Sie aus mehreren Dateien Inhalte lesen und in mehrere Dateien Inhalte schreiben können. In Terminal 1 werden die Inhalte der Ausgangsdateien angezeigt, die Sie, wie alle Skripte, aus dem Internet von der Adresse *http://www.clb.de/computare.htm* herunterladen können.

Terminal 1

```
01    $ cat 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    $ cat dataCO2.txt
15    time-CO2
16    23:00-1.3234
17    23:04-1.637
18    23:08-1.6
19    23:12-1.583
20    23:14-1.4485
21    23:16-1.41
22    $ cat dataO2.txt
23    time-O2
24    23:00-12.44
25    23:02-12.2876
26    23:04-12.24
27    23:06-11.965
28    23:08-12.1
29    23:10-12.373
30    $
```

Sie erinnern sich? Der Befehl `cat` zeigt den Inhalt einer Datei an.

Ausgabe in Dateien

Lassen Sie uns damit beginnen, den pH-Wert und die Temperatur der Datei *data.txt* zusammen mit der Uhrzeit in die Dateien *dataPH.txt* bzw. *dataTEMP.txt* zu extrahieren. In Teil 4 (CLB 02/04) haben Sie bereits die *redirection* kennen gelernt. Seitdem haben wir diese "Umleitung" häufiger verwendet um die Ausgabe auf den Bildschirm in eine Datei umzuleiten. Dabei ersetzt > den Inhalt einer vorhandenen Datei, während >> die Ausgabe an eine vorhandene Datei anfügt. In beiden Fällen wird die Datei erstellt, falls sie nicht existiert. Dieselbe Syntax können wir auch in Kombination mit dem `print` Befehl in Awk-Skripten verwenden. Der Befehl `print "Hello World" > "datei.txt"` schreibt den Text "Hello World" in die Datei `datei.txt`. Dies ist in Terminal 2 gezeigt.

Terminal 2

```
01    $ awk 'BEGIN{print "CLB" >
      "datei.txt"}'
02    $ cat datei.txt
03    CLB
04    $ awk 'BEGIN{print "ist
      toll" >> "datei.txt"}'
05    $ cat datei.txt
06    CLB
07    ist toll
08    $
```

```
01    #!/bin/awk -f
02
03    BEGIN{FS="-"}
04    {
05    print $1" - "$2 > "dataPH.txt"
06    print $1" - "$3 > "dataTEMP.txt"
07    }
```

Skript *extract.awk*

Mit Awk ist es also sehr einfach in Dateien zu schreiben. Wie lösen wir nun die uns gestellte Aufgabe? Nun, wir wissen, dass das Feldtrennzeichen der Datei *data.txt* das Minuszeichen ist. Das erste Feld, gespeichert in der Variablen *$1*, enthält dann die Zeit, das zweite Feld (*$2*) den pH-Wert und das dritte Feld (*$3*) die Temperatur. Das Skript *extract.awk* zeigt die Lösung.

Im `BEGIN`-Block legen wir das Feldtrennzeichen (Wert der Variablen *FS*) fest (siehe Teil 11 in CLB 09/04). In Zeile 5 geben wir das erste und zweite Feld der Eingabedatei in die Datei *dataPH.txt* aus. In Zeile 6 geben wir das erste und dritte Feld der Eingabedatei in die Datei *dataTEMP.txt* aus. Fertig! Aber warum müssen wir nicht das >> Kommando anstatt von > verwenden? Innerhalb eines Skriptes hängt Awk automatisch die neuen Zeilen an die Datei an. Terminal 3 zeigt die Anwendung und das Resultat des Skriptes.

Terminal 3

```
01    $ awk -f extract.awk data.txt
02    $ cat dataPH.txt
03    time - pH
04    23:01 - 13.4
05    23:02 - 13.5
06    23:03 - 13.4
07    23:04 - 13.6
08    23:05 - 13.7
09    23:06 - 14.0
10    23:07 - 14.1
11    23:08 - 13.8
12    23:09 - 13.7
13    23:10 - 13.3
14    23:11 - 13.6
15    $ cat dataTEMP.txt
16    time - temp
17    23:01 - 73.4
18    23:02 - 73.3
19    23:03 - 73.1
20    23:04 - 72.6
21    23:05 - 71.1
22    23:06 - 70.4
23    23:07 - 71.4
24    23:08 - 72.3
25    23:09 - 73.1
26    23:10 - 73.5
27    23:11 - 73.3
28    $
```

Dateiinhalte einlesen

Wenden wir uns der schwierigeren Aufgabe zu, die Inhalte der drei Dateien zu einer Datei namens *dataALL.txt* zu fusionieren. Diese Aufgabe soll ein Awk-Skript namens *fuse.awk* erfüllen. Betrachten wir zunächst die Struktur der drei Dateien. Alle haben die Uhrzeit als gemeinsames Merkmal. Allerdings gibt es nicht für alle Uhrzeiten alle Daten. Wir werden sie daher heranziehen um die Daten zu "synchronisieren". Dazu können wir z.B. einen Hash (assoziativer Array) verwenden (siehe Teil 12 in CLB 10/04), dessen Index (Schlüssel, *key*) die Uhrzeit und dessen Werte die Messdaten sind. Schauen Sie sich das Skript *fuse.awk* und seine Ausgabe in Terminal 4 an.

Das Kommando in Zeile 1 des Skriptes *fuse.awk* ermöglicht seinen Aufruf ohne explizit den Awk-Kommandointerpreter aufzurufen (siehe Teil 10 in CLB 08/04). D.h., anstatt `awk -f fuse.awk` können Sie das Skript mit dem Befehl `./fuse.awk` ausführen. Voraussetzung ist allerdings, dass Sie die Skriptdatei mit dem Kommando `chmod u+x fuse.awk` ausführbar machen (siehe Teil 10 in CLB 08/04).

In Zeile 4 definieren wir das Minuszeichen als Feldtrennzeichen. Darauf folgen drei `while`-Blöcke (engl. *while* bedeutet "solange"). Sie arbeiten nach dem Schema: *while (ist wahr){führe aus}*. In den runden Klammern wird eine Kondition überprüft. Solange diese Kondition zutrifft (wahr ist), werden die Befehle in den geschwungenen Klammern ausgeführt. Welche Kondition wird in unserem Fall überprüft? Es ist die Frage, ob z.B. (`getline < "data.txt"`) größer als Null ist. Der Befehl `getline` liest in diesem Fall eine Zeile von der Datei *data.txt*. Diese Zeile wird in der Variablen *$0* gespeichert, gemäß dem Feldtrennzeichen aufgeteilt und in den Variablen *$1*,

Skript *fuse.awk*

```
01    #!/bin/awk -f
02
03    BEGIN{
04    FS="-"
05
06    while ((getline < "data.txt") > 0){
07      datenPH[$1]=$2
08      datenTEMP[$1]=$3
09      datenPRES[$1]=$4
10      zeiten[$1]=""
11      }
12    while ((getline < "dataCO2.txt") > 0){
13      datenCO2[$1]=$2
14      zeiten[$1]=""
15      }
16    while ((getline < "dataO2.txt") > 0){
17      datenO2[$1]=$2
18      zeiten[$1]=""
19      }
20
21    for (zeit in zeiten){
22      print zeit" - "datenPH[zeit]" - "datenTEMP[zeit]" - "
        datenPRES[zeit]" - "datenCO2[zeit]" - "datenO2[zeit]
23      }
24    }
```

$2, usw. gespeichert. Wenn eine Zeile erfolgreich gelesen wurde, liefert der Befehl außerdem als Ergebnis den Wert 1. Solange also Zeilen aus der Datei *data.txt* gelesen werden können, liefert `getline` den Wert 1. Die Kondition *1 > 0* ist erfüllt. D.h. die Befehle in den Zeilen 07-10 werden ausgeführt und die nächste Zeile aus der Datei *data.txt* wird gelesen. Wurde die letzte Zeile der Datei *data.txt* gelesen, kann keine weitere Zeile mehr gelesen werden und `getline` liefert als Ergebnis 0. In diesem Fall ist die Bedingung *0 > 0* nicht erfüllt und das Skript fährt in Zeile 12 fort.

Auf die dargestellte Art und Weise werden alle drei Dateien zeilenweise eingelesen. Wie aber werden die Daten gespeichert und weiter verarbeitet? Dazu verwenden wir für jeden Messwert eine eigene Hash-Variable (siehe Teil 12 in CLB 10/04). Als Index verwenden wir jeweils die Uhrzeit, zu der der Wert gemessen wurde. Außerdem verwenden wir in den Zeilen 10, 14 und 18 den Hash *zeiten*, um alle Uhrzeiten zu denen je gemessen wurde zu erfassen. Dies ist notwendig, da nicht jede Datei die gleichen Zeitpunkte enthält. Der Hash *zeiten* besteht nur aus Indizes - wir weisen jeder Uhrzeit einen leeren Textstring zu (`""`). Indizes die bereits vorhanden sind, werden einfach überschrieben. Auf diese Weise enthält *zeiten* jede Uhrzeit zu der jemals gemessen wurde exakt einmal.

In den Zeilen 21 bis 23 erfolgt die Ausgabe der Daten. Hierfür machen wir uns das `for`-Konstrukt zunutze (engl. *for* bedeutet "für"; siehe auch Skript 1 in CLB 10/04). Es arbeitet nach dem Schema: *for (jeden Index des Hashes zeiten){führe aus}*. Der Hash *zeiten* besteht in unserem Fall aus 16 Elementen, nämlich den 16 einmaligen Uhrzeiten, die als Index des Hashes dienen. Das Kommando `for (zeit in zeiten)` nimmt einen Index (also eine Uhrzeit) des Hashes *zeiten* und speichert ihn in der Variablen *zeit* ab. Dann werden die Befehle in der nachfolgenden geschwungenen Klammer ausgeführt, also die Zeile 22. Der aktuelle Index steht in der Variablen *zeit* zur Verfügung und wird genutzt, um alle Messdaten zum gegebenen Zeitpunkt mittels einer `print` Anweisung auszugeben. Die einzelnen Messdaten werden dabei von dem Textstring Leerzeichen-Minus-Leerzeichen (" - ") getrennt. Die Zeile 22 wird somit 16-mal ausgeführt. Die Ausgabe ist in Terminal 4 gezeigt.

Wir hätten auch weiter oben im Skript *fuse.awk*, etwa nach der Zeile 4, den Ausgabe-Feldseperator (OFS) als " - " definieren können (siehe Teil 11 in CLB 09/04). In diesem Fall hätte sich die Zeile 22 als `print zeit datenPH[zeit] datenTEMP[zeit] datenPRES[zeit] datenCO2[zeit] datenO2[zeit]` schreiben lassen. Generell führen immer mehrere Wege zum Ziel. Mit der Ausgabe, wie sie in Terminal 4 zu sehen ist, können wir schon ziemlich zufrieden sein - aber nicht ganz. Die Ausgabe ist in keiner Weise sortiert.

Terminal 4

```
$ ./fuse.awk
23:08 - 13.8 - 72.3 - 12 - 1.6 - 12.1
23:09 - 13.7 - 73.1 - 12 -  -
23:00 -  -  -  - 1.3234 - 12.44
23:10 - 13.3 - 73.5 - 12 -  - 12.373
23:01 - 13.4 - 73.4 - 12 -  -
23:11 - 13.6 - 73.3 - 12 -  -
23:02 - 13.5 - 73.3 - 12 -  - 12.2876
23:12 -  -  -  - 1.583 -
23:03 - 13.4 - 73.1 - 12 -  -
23:04 - 13.6 - 72.6 - 12 - 1.637 - 12.24
23:14 -  -  -  - 1.4485 -
23:05 - 13.7 - 71.1 - 12 -  -
23:06 - 14.0 - 70.4 - 11 -  - 11.965
time - pH - temp - pres - CO2 - O2
23:16 -  -  -  - 1.41 -
23:07 - 14.1 - 71.4 - 12 -  -
$
```

Dies liegt daran, dass das `for`-Konstrukt in der Zeile 21 die Daten in einer Weise aus dem Arbeitsspeicher liest, die wir nicht beeinflussen können. Wir können die Ausgabe aber einfach an den `sort` Befehl weiterleiten (siehe Teil 4 in CLB 02/04). Dies ist in Terminal 5 gezeigt.

Wir verwenden `sort` mit der Option `-n`, um die Spaltenbezeichnung als erste Zeile zu erhalten. Probieren Sie zum Vergleich `sort` ohne jede Option aus. Auf diese Weise nutzen wir geschickt die Leistung vorhandener Programme aus und müssen uns um das Sortieren in dem Awk-Skript selbst keine Gedanken machen.

Terminal 5

```
$ ./fuse.awk | sort -n
time - pH - temp - pres - CO2 - O2
23:00 -  -  -  - 1.3234 - 12.44
23:01 - 13.4 - 73.4 - 12 -  -
23:02 - 13.5 - 73.3 - 12 -  - 12.2876
23:03 - 13.4 - 73.1 - 12 -  -
23:04 - 13.6 - 72.6 - 12 - 1.637 - 12.24
23:05 - 13.7 - 71.1 - 12 -  -
23:06 - 14.0 - 70.4 - 11 -  - 11.965
23:07 - 14.1 - 71.4 - 12 -  -
23:08 - 13.8 - 72.3 - 12 - 1.6 - 12.1
23:09 - 13.7 - 73.1 - 12 -  -
23:10 - 13.3 - 73.5 - 12 -  - 12.373
23:11 - 13.6 - 73.3 - 12 -  -
23:12 -  -  -  - 1.583 -
23:14 -  -  -  - 1.4485 -
23:16 -  -  -  - 1.41 -
$
```

```
01      for (zeit in zeiten){
02        if (zeit == "time"){
03          printf " %-6s  %-6s %-6s %-6s  %-6s %-6s \n", zeit, datenPH[zeit],
            datenTEMP[zeit], datenPRES[zeit], datenCO2[zeit], datenO2[zeit]
04        }
05        else{
06          printf "%-6s %6.2f %6.2f %6.2f %6.2f %6.2f \n", zeit,
            datenPH[zeit], datenTEMP[zeit], datenPRES[zeit], datenCO2[zeit],
            datenO2[zeit]
07        }
08      }
```

Skript Erweiterung für fuse.awk

Ausgabe formatieren

Einen weiteren Schönheitsfehler hat die Ausgabe: sie ist unübersichtlich formatiert. Daher möchte ich Ihnen schließlich zeigen, wie die Ausgabe formatiert werden kann. Hierfür bietet Awk den Befehl `printf` (*print formated*, formatiert drucken). Dieser Befehl ist recht kompliziert in der Handhabung. Dennoch möchte ich Ihnen an unserem Beispiel kurz die Arbeitsweise von `printf` vorstellen. Weitere Details finden Sie wie immer in den Manualseiten, die Sie mit dem Kommando `man awk` aufrufen können. Das besondere an `printf` gegenüber print ist, dass Sie einen so genannten *format string* verwenden müssen. Auf diesen folgen, getrennt durch Kommata, die eigentlich auszugebenden Daten. Wir haben zwei Arten von Daten, Texte und Zahlen mit Nachkommastellen. Mit dem Formatierbefehl `%s` geben Sie Zeichenketten und mit `%f` Dezimalzahlen (*floating number*) aus. Durch Einfügen einer Zahl wird die Anzahl der Zeichen festgelegt, die das Datum einnehmen darf. So bedeutet `%6s`, dass ein Textstring 6 Zeichen einnehmen darf. Ist der String kleiner, wird der verbleibende Platz mit Leerzeichen aufgefüllt. Ein vorangestelltes Minuszeichen bedeutet, das der Textstring linksbündig orientiert wird. Bei Dezimalzahlen kann, gefolgt von einem Punkt, zudem die Anzahl der Nachkommastellen festgelegt werden. Mit `%6.2f` wird eine Dezimalzahl mit 2 Nachkommastellen ausgegeben. Die gesamte Zahl darf 6 Zeichen einnehmen und wird rechtsbündig ausgegeben. `printf` gibt keinen Zeilenumbruch aus. Daher müssen wir ihn explizit angeben (`\n`). Es gibt noch eine Vielzahl weitere Möglichkeiten der Formatierung, auf die ich hier aber nicht eingehen möchte.

Für die formatierte Ausgabe müssen wir die Zeilen 21 bis 23 im Skript *fuse.awk* durch die Zeilen in Skript Erweiterung ersetzen.

Die Erweiterung ist etwas umfangreicher, da wir überprüfen müssen, ob die ausgegebene Zeile die Spaltentitel enthält. Diese können wir nicht als Dezimalzahl formatieren. Daher geben wir alle Variablenwerte als Strings aus, wenn (`if`) die Variable *zeit* dem Wert "time" entspricht. Ansonsten (`else`) wird nur der Wert der Variablen *zeit* als Text, alle anderen Variablenwerte als Dezimalzahl mit zwei Nachkommastellen ausgegeben. Die Ausgabe der neuen Version ist in Terminal 6 gezeigt.

Um die Aufgabe endgültig zu lösen, müssen wir das Ergebnis nur noch in die Datei *dataALL.txt* umleiten. Der Befehl hierzu lautet: `./fuse.awk | sort -n > dataALL.txt`. Die Formatierung der Ausgabe in Terminal 6 mag uns zwar gefallen, allerdings wurde allen nicht gemessenen Daten durch `printf` einfach die Zahl Null zugewiesen. In Terminal 5 waren in diesen Fällen keine Werte eingetragen. Haben Sie eine Idee, wie das Problem zu lösen ist?

Terminal 6

```
01      $ awk -f fuse.awk | sort -n
02       time    pH     temp   pres     CO2     O2
03      23:00    0.00   0.00   0.00    1.32   12.44
04      23:01   13.40  73.40  12.00    0.00    0.00
05      23:02   13.50  73.30  12.00    0.00   12.29
06      23:03   13.40  73.10  12.00    0.00    0.00
07      23:04   13.60  72.60  12.00    1.64   12.24
08      23:05   13.70  71.10  12.00    0.00    0.00
09      23:06   14.00  70.40  11.00    0.00   11.96
10      23:07   14.10  71.40  12.00    0.00    0.00
11      23:08   13.80  72.30  12.00    1.60   12.10
12      23:09   13.70  73.10  12.00    0.00    0.00
13      23:10   13.30  73.50  12.00    0.00   12.37
14      23:11   13.60  73.30  12.00    0.00    0.00
15      23:12    0.00   0.00   0.00    1.58    0.00
16      23:14    0.00   0.00   0.00    1.45    0.00
17      23:16    0.00   0.00   0.00    1.41    0.00
18      $
```

Neue Befehle in dieser Ausgabe

`printf "Format-String", Variablen` Formatierte Ausgabe

`if (Kondition ist wahr) {Befehle}` Bedingte Befehlsausführung

`else {Befehle}`

`while (Kondition ist wahr) {Befehle}` Bedingte Befehlsausführung

`getline < "Dateiname"` Lese Zeile von Datei

`print irgendwas > "Dateiname"` Drucke *irgendwas* in Datei