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

Teil 10: Awk Grundlagen

Röbbe Wünschiers

Ein anwenderfreundliches aber vielseitiges Werkzeug für die Datenprozessierung ist die Skriptsprache Awk – benannt nach den Familiennamen ihrer Autoren Aho, Weinberger und Kernighan. Obwohl bereits in den 70iger Jahren entwickelt, ist Awk nach wie vor sehr beliebt und weit verbreitet. Viele der alltäglich anfallenden Datenformatierungen und -analysen lassen sich leicht und effektiv umsetzten. Wie bei den meisten Aufgaben aus der Datenprozessierung ist es die elegante Anwendung von regulären Ausdrücken (CLB 04/04), die, verpackt in ein kleines Skript, zum Ziel führt. In der letzten Ausgabe habe ich Sie mit dem Awk-Skript *pdb2xyz.awk* zur Umwandlung eines Molekülstrukturdatei Formats ins kalte Wasser geworfen. Heute möchte ich die grundlegende Arbeitsweise von Awk vorstellen.

Der Aufruf eines Awk-Skripts besteht aus mehreren Bestandteilen (Abbildung 1). Unablässig ist natürlich der Awk-Interpreter selbst, der mit dem Befehl `awk` oder `gawk` (GNU Awk) aufgerufen wird. Um festzustellen ob Awk auf Ihrem System installiert ist, geben Sie in der Kommandozeile (Terminal, Konsole, Shell) einfach den Befehl `awk --version` oder `gawk --version` ein. Ist Awk installiert, dann erhalten Sie auf diese Weise Informationen über die installierte Version. Auf den Aufruf des Awk-Interpreters folgt das Awk-Skript selbst, gefolgt von der zu bearbeitenden Datei. Dieser Aufruf kann auf dreierlei Weise erfolgen. Handelt es sich um ein sehr kurzes Skript, einen so genannten Einzeiler, dann wird das Skript meist direkt, eingeschlossen in einfachen Anführungszeichen, zusammen mit der zu bearbeitenden Datei (*datei.txt*) eingegeben: `awk 'skript' datei.txt`. Liegt ein längeres Skript vor, oder ein Einzeiler den man sehr häufig anwendet, dann wird das Skript in einer Datei abgelegt und nur der Dateiname des Skripts

Abbildung 1: **Aufruf eines Awk-Skripts**. Ein Einzeiler wird, eingeschlossen von einfachen Anführungszeichen, direkt eingegeben (oben). Eine Skriptdatei wird mit der Option `-f` (*file*) aufgerufen (unten). Wesentlich ist der Aufruf des Awk-Interpreters (1), des Skripts (2) und meistens einer zu bearbeitenden Eingabedatei.

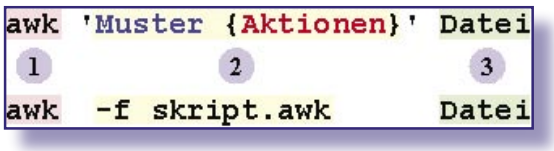

Abbildung 2: **Zeilen & Wörter**. Awk zerlegt jede Zeile (*record*) in Wörter (*fields*). Die aktuelle Zeile ist in der Variablen *$0*, die Wörter der aktuellen Zeile in den Variablen *$1*, *$2*, ... gespeichert.

(*skript.awk*) zusammen mit der zu bearbeitenden Datei (*datei.txt*) angegeben: `awk -f skript.awk datei.txt`. Schließlich kann das Skript so gestaltet werden, dass es wie ein normales Linux-Kommando ausgeführt wird: `skript.awk datei.txt`. Alle drei Möglichkeiten werden wir uns im Folgenden näher ansehen. Die Ausgabe erfolgt immer auf den Bildschirm, d.h. die original Datei wird nicht verändert. Zur Speicherung des Ergebnisses wird die Umleitung (*redirection*) verwendet (siehe CLB 01/04).

Zeilen und Wörter

Awk behandelt die zu bearbeitende Datei zeilenweise (Abbildung 2). Zeilen sind standardmäßig dadurch charakterisiert, dass sie mit einem Zeilenumbruch enden. Der Zeilenumbruch wird in Linux durch die beiden Zeichen `\n` erzeugt. Die aktuelle Zeile ist in der Variabel *$0* gespeichert. Jede Zeile wird von Awk wiederum in Wörter zerlegt. Das erste Wort der aktuellen Zeile ist in der Variablen *$1*, das zweite Wort in der Variablen *$2* usw. abgelegt. Wörter sind für Awk standardmäßig zusammenhängende Zeichenfolgen die durch Leerzeichen oder Tabulatoren voneinander getrennt sind. Ich schreibe *standardmäßig*, da Awk dazu angewiesen werden kann andere Trennzeichen für Zeilen und Wörter zu verwenden – doch davon mehr in einer späteren Ausgabe. In der Awk-Sprache ist eine Zeile ein *record* und ein Wort ein *field*.

Skript Syntax

Ein Awk-Skript besteht grundsätzlich aus zwei Elementen, einem Muster (*pattern*) und den Befehlen (*actions*). Das Muster beschreibt die Zeilen der Eingabedatei die bearbeitet werden sollen. Wird das Muster weggelassen, dann werden alle Zeilen bearbeitet. Die Befehle geben an, wie die Zeilen oder Wörter bearbeitet werden sollen. Befehle können in einem `BEGIN`-Block, einem Hauptteil und einem `END`-Block stehen

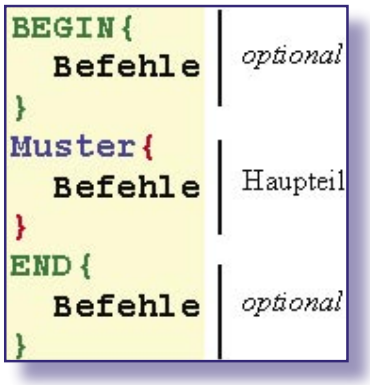

Abbildung 3: **Skript Aufbau**. Den Kern eines Awk-Skripts bilden die Befehle des Hauptteils. Sie werden auf jede Zeile der Eingabedatei angewandt. Optional kann ein Muster (*pattern*) angegeben werden, das die zu bearbeitenden Zeilen der Eingabedatei einschränkt. Die optionalen BEGIN- und END-Blöcke werden ausschließlich vor und nach der Bearbeitung des Hauptteils abgearbeitet. Alle Befehle des Hauptteils sowie der BEGIN- und END-Block werden jeweils von geschwungenen Klammern eingeschlossen. Es ist wichtig, dass das Muster in einer Zeile mit der öffnenden Klammer des Hauptteils liegt.

(Abbildung 3). Die Befehle der optionalen **BEGIN**- bzw. **END**-Blöcke werden einmalig vor bzw. nach der Bearbeitung der Eingabedatei ausgeführt. Die Kommandos des Hauptteils dagegen werden für jede Zeile auf die das Muster zutrifft ausgeführt.

Betrachten wir nun ein einfaches Beispiel. Zunächst erzeugen wir eine Beispieldatei namens *enzyme.txt* mit den Michaelis-Menten Konstanten einiger Enzyme.

Terminal 1

```
01    $ cat > enzyme.txt
02    Enzym Km
03    Protease    2.5
04    Hydrolase   0.4
05    ATPase      1.2
06    $
```

In Zeile 1 lenken wir mit dem Befehl `cat > enzyme.txt` die Eingabe der Tastatur auf den Bildschirm und in die Datei *enzyme.txt*. Mit der Tastenkombination CTRL + D beenden wir die Eingabe nach Zeile 5 und gelangen zurück in die Eingabezeile. Trennen Sie die Enzymnamen von den zugehörigen Km-Werten durch eine oder mehrere Leerzeichen oder Tabulatoren. Natürlich können Sie die Datei *enzyme.txt* auch mit dem Texteditor Vim erstellen (siehe CLB 02/04).

Jetzt wenden wir Awk's einfachsten Befehl an; die Ausgabe aller Zeilen durch den `print` Befehl. Mit dem Muster `/2/` beschränken wir den `print` Befehl auf alle Zeilen, welche die Zahl 2 enthalten.

Terminal 2

```
01    $ awk '/2/{print $0}' enzyme.
      txt
02    Protease    2.5
03    ATPase      1.2
04    $
```

Terminal 2 zeigt alle wichtigen Eigenschaften eines Awk-Skripts. Das Skript ist in einfachen Anführungszeichen eingeschlossen. In diesem Beispiel gibt es keinen **BEGIN**- oder **END**-Block, sondern nur einen Hauptteil der von geschwungenen Klammer umgeben ist. Er besteht nur aus dem Befehl `print $0`. Wie weiter oben angesprochen entspricht der Wert der Variable *$0* der gesamten aktuell bearbeiteten Zeile. Vor dem Hauptteil steht das Muster (*pattern*), der reguläre Ausdruck `/2/` (reguläre Ausdrücke werden bei Awk immer von Schrägstrichen (*slashes*) umgeben). Daher werden nur Zeilen ausgedruckt, welche die Zahl 2 enthalten. Würden wir das Muster weglassen, dann würden alle Zeilen ausgegeben. Dies ist im folgenden Beispiel gezeigt.

Terminal 3

```
01    $ awk '{print $1}' enzyme.txt
02    Enzym
03    Protease
04    Hydrolase
05    ATPase
06    $
```

Allerdings geben wir in Terminal 3 nicht die gesamte Zeile aus, sondern nur das erste Wort einer jeden Zeile. Das erste Wort ist in der Variablen *$1* gespeichert (Abbildung 2).

Lassen Sie uns nun ein Awk-Skript in eine Textdatei schreiben. Am besten verwenden Sie hierzu den Texteditor Vim (siehe CLB 02/04). Speichern Sie das nachfolgend wiedergegebene Skript in einer Datei namens *zwei.awk ab*.

Skript *zwei.awk*

```
01    #!/usr/bin/awk -f
02    BEGIN{print "Zeilen mit
      2..."}
03    /2/{print $0}
04    END{print "Fertig"}
```

Das Skript besteht aus vier Zeilen. Würden wir nur die Zeile 3 verwenden, dann hätten wir ein Skript mit der in Terminal 2 dargestellten Funktion. Die Zeilen 2 und 4 erweitern das Skript um einen **BEGIN**- und einen **END**-Block. D.h. vor und nach der Bearbeitung der Eingabedatei wird in diesem Fall ein Text ausgegeben. Die erste Zeile sieht etwas sonderbar aus. Sie ist ausschließlich dann notwendig, wenn das Skript wie ein normaler Linux-Befehl, also ohne Aufruf des Kommandointerpreters **awk**, ausgeführt werden soll. Mehr dazu in Terminal 5. Werfen wir zunächst einen Blick auf Termial 4.

Terminal 4

```
01    $ awk -f zwei.awk enzyme.txt
02    Zeilen mit 2...
03    Protease        2.5
04    ATPase  1.2
05    Fertig
06    $
```

Zeile 1 von Terminal 4 zeigt den Aufruf eines Awk-Skripts aus einer Datei. Im Vergleich zu Zeile 1 in den Terminals 3 und 4 entfallen die einfachen Anführungszeichen. Dafür benötigen wir die Option `-f` (*file*) gefolgt von dem Namen der Skriptdatei. Alles andere bleibt gleich. Die wahrscheinlich komfortabelste Art und Weise ein Awk-Skript aufzurufen ist die, bei der das Skript wie ein Linux-Kommando aufgerufen werden kann. Dazu muss das Skript, wie oben dargestellt, mit der Zeile `#!/usr/bin/awk -f` beginnen. Die Kombination „`#!`“ wird auch als *shebang* bezeichnet. Sie bewirkt den direkten Aufruf des nachfolgenden Kommandos unter Umgehung der Shell. Es wird also, wie in Terminal 4, `awk` mit der Option `-f` aufgerufen. Allerdings muss der Pfad zu der Programmdatei *awk* angegeben werden; hier */usr/bin/* (CLB 12/03). Auf Ihrem System kann Awk sich woanders befinden. Sie können den tatsächlichen Pfad mit dem Kommando `whereis awk` bestimmen. Weiterhin muss das Awk-Skript ausführbar sein. Dazu ist es notwendig die Zugriffsrechte der Datei zu ändern. Wie in Teil 2 (CLB 12/03) bereits besprochen, können die aktuellen Zugriffsrechte mit dem Befehl `ls -l` angezeigt werden. Mit dem Befehl `chmod u+x zwei.awk` (`chmod` = *change file modes*) werden dem Besitzer (`u` = *user*) die Rechte zum Ausführen (`+x` = *plus execute*) der Datei *zwei.awk* zugewiesen.

Terminal 5

```
01    $ ls -l zwei.awk
02    -rw-r--r--  1 rw  staff  83 Jul
      26 14:27 zwei.awk
03    $ chmod u+x zwei.awk
04    $ ls -l zwei.awk
05    -rwxr--r--  1 rw  staff  83 Jul
      26 14:27 zwei.awk
06    $ ./zwei.awk enzyme.txt
07    Zeilen mit 2...
08    Protease        2.5
09    ATPase  1.2
10    Fertig
11    $
```

Die Umstellung der Zugriffsrechte erfolgt in Zeile 3 von Terminal 5. In Zeile 6 rufen wir das Skript wie einen Linux-Befehl auf. Die Zeichen `./` müssen vorangestellt werden, da sich das Benutzerverzeichnis normalerweise nicht in dem Systempfad befindet, in welchem Linux nach Befehlen sucht. Sie können sich aber z.B., wie in CLB 03/04 beschrieben, ein Alias erzeugen. Eine andere Möglichkeit ist die Erstellung eines Verzeichnisses in welches Sie alle selbst geschriebenen Skripte kopieren. Dieses Verzeichnis können Sie dann wie in Zeile 3 von Terminal 6 gezeigt zum Systempfad hinzufügen.

Terminal 6

```
01    $ mkdir ~/Skripte
02    $ cp zwei.awk ~/Skripte
03    $ PATH=$PATH:~/Skripte
04    $
```

Ab sofort können Sie von jedem Verzeichnis aus das Skript *zwei.awk* und alle anderen Skripte die Sie in das Verzeichnis *~/Skripte* kopieren, direkt und ohne die Verwendung von „`./`“ausführen. Zur Erinnerung: ~ bezeichnet den Pfad zu Ihrem Home-Verzeichnis (siehe CLB 12/03). Um die Veränderungen permanent zu speichern fügen Sie die Zeile 3 in die Datei *.bashrc* ein (siehe auch CLB 03/04).

In der kommenden Ausgabe werden Sie u.a. mehrere Möglichkeiten kennen lernen das Muster (*pattern*) zu definieren.

Neue Befehle in dieser Ausgabe

`awk`	Aufruf des Kommandointerpreters von Awk
`chmod`	Änderung der Zugriffsrechte einer Datei
`whereis`	Bestimmung des Pfades eines Kommandos