

AWK

In diesem Teil der Veranstaltung wird awk als Werkzeug zur Verarbeitung und Analyse von Daten behandelt.

Inhalt

| | |
|---|--------------------|
| 1. Überblick..... | 3 |
| 2. Programmstruktur und Aufrufsyntax..... | 5 |
| 3. Einfache und formatierte Ausgabe..... | 8 |
| 4. Vergleichemöglichkeiten..... | 11 |
| 5. Anfangs- und Ende-Aktion..... | 12 |
| 6. Variablen, Typen und Ausdrücke..... | 14 |
| 6.1. Vordefinierte Variablen..... | 14 |
| 6.2. Benutzerdefinierte Variablen..... | 15 |
| 6.3. Konkatenation von Strings..... | 16 |
| 6.4. Typen und Ausdrücke..... | 16 |

| | |
|--|--------------------|
| 7. Kontrollstrukturen..... | 19 |
| 8. awk Funktionen..... | 23 |
| 9. Reguläre Ausdrücke..... | 25 |
| 10. Beispiele..... | 26 |

1. Überblick

AWK ist ein Filter, der zur Manipulation von Textdateien verwendet werden kann. Das Programm wurde von

Alfred V. **A**ho

Peter J. **W**einberger

Brian B. **K**ernighan

als Programmiersprache zur Verarbeitung und Analyse von Daten entwickelt.

Folgende **Eigenschaften** charakterisieren awk:

Einfaches aber **mächtiges Programmierwerkzeug**,
spezialisiert auf **ASCII-Daten**,
awk-**Programme** sind meist **kurz** und
schnell geschrieben (**Prototyping**),
C-ähnliche Syntax.

Awk verarbeitet Textdateien, in denen Spalten durch Separatoren (Default=Blank,) getrennt sind. \$i repräsentiert Spalte i, \$0 alle Spalten.

Beispiel (Tafel):

```
$ cat bundesliga
Muenchen - Nuernberg          3 : 2 34000 Zuschauer
Kaiserslautern - Moenchengladbach 2 : 1 28260 Zuschauer
Uerdingen - Homburg           3 : 0 10000 Zuschauer
St.Pauli - Bremen             0 : 0 20600 Zuschauer
Leverkusen - Dortmund        1 : 0 22000 Zuschauer
Stuttgart - Karlsruhe        2 : 0 34000 Zuschauer
Bochum - Koeln                0 : 1 21000 Zuschauer
$
```

Wer war die gewinnende Heimmanschaft?

```
$ awk '$4 > $6 { print $1 }' bundesliga
Muenchen
Kaiserslautern
Uerdingen
Leverkusen
Stuttgart
$
```

Welche Spiele sind unentschieden ausgegangen?

```
$ awk '$4 == $6 { print $1, "-", $3, $4 ":" $6 }' bundesliga
St.Pauli - Bremen 0:0
$
```

2. Programmstruktur und Aufrufsyntax

Die **Programmstruktur** ist einfach:

```
    Muster  { Aktionen }  
    |      |  
awk '$4 > $6 { print $1 }'
```

Muster und Aktion sind nicht beide zwingend.

Muster ohne Aktion (ganze Zeile wird ausgegeben):

```
$ awk '$4 > 2' bundesliga  
Muenchen - Nuernberg           3 : 2 34000 Zuschauer  
Uerdingen - Homburg           3 : 0 10000 Zuschauer  
$
```

Aktion ohne Muster (alle Zeilen werden bearbeitet):

```
$ awk '{ print $1 " schießt zuhause " $4 " Tore" }' bundesliga  
Muenchen schießt zuhause 3 Tore  
Kaiserslautern schießt zuhause 2 Tore  
Uerdingen schießt zuhause 3 Tore  
St.Pauli schießt zuhause 0 Tore  
Leverkusen schießt zuhause 1 Tore  
Stuttgart schießt zuhause 2 Tore  
Bochum schießt zuhause 0 Tore  
$
```

Der awk wird wie folgt aufgerufen.

awk 'awk-Programm' Eingabedatei(en)

```
$ awk '$4 > $6 { print $1 }' bundesliga1 Bundesliega2  
...
```

Ohne Eingabedateien wird die **Standardeingabe** als Datei verwendet:

```
$ awk '$1 > 0 {print $1, " mit MwSt: ", $1*1.16}'  
100  
100 mit MwSt: 116  
$
```

Ein **awk-Programm** kann in einer **Datei** abgelegt werden:

```
$ cat MwSt.awk  
$1 > 0 {print $1, " mit MwSt: ", $1*1.16}  
$  
$ awk -f MwSt.awk  
100  
100 mit MwSt: 116  
$
```

awk-Aufruf als **Filter**:

```
$ echo 100 | awk -f MwSt.awk
```

```
100 mit MwSt: 116
$
```

```
$ echo 100 | awk '$1 > 0 {print $1, " mit MwSt: ", $1*1.16}'
100 mit MwSt: 116
$
```

awk-Aufruf mit **Programmdatei** und **Datendatei**:

```
$ awk -f MwSt.awk betraege
100 mit MwSt: 116
200 mit MwSt: 232
300 mit MwSt: 348
$
```

3.Einfache und formatierte Ausgabe

Jede Zeile ausgeben:

```
$ awk '{print $0}' bundesliga
Muenchen - Nuernberg          3 : 2 34000 Zuschauer
Kaiserslautern - Moenchengladbach 2 : 1 28260 Zuschauer
Uerdingen - Homburg           3 : 0 10000 Zuschauer
St.Pauli - Bremen             0 : 0 20600 Zuschauer
Leverkusen - Dortmund         1 : 0 22000 Zuschauer
Stuttgart - Karlsruhe         2 : 0 34000 Zuschauer
Bochum - Koeln                0 : 1 21000 Zuschauer
$
```

Nur **bestimmte Felder mit Zwischenraum** ausgeben:

```
$ awk '{print $1, $3}' bundesliga
Muenchen Nuernberg
Kaiserslautern Moenchengladbach
Uerdingen Homburg
St.Pauli Bremen
Leverkusen Dortmund
Stuttgart Karlsruhe
Bochum Koeln
$
```

Nur **bestimmte Felder ohne Zwischenraum** ausgeben:

```
$ awk '{print $1 $3}' bundesliga
MuenchenNuernberg
KaiserslauternMoenchengladbach
UerdingenHomburg
St.PauliBremen
LeverkusenDortmund
StuttgartKarlsruhe
BochumKoeln
$
```

Textausgabe:

```
$ awk '$6 > $4 { print $3 " gewinnt in " $1 }' bundesliga
Koeln gewinnt in Bochum
$
```

Durch das awk-Kommando **printf** sind **formatierte Ausgaben** möglich:

```
$ cat bundesliga
Muenchen - Nuernberg          3 : 2 34000 Zuschauer
Kaiserslautern - Moenchengladbach 2 : 1 28260 Zuschauer
Uerdingen - Homburg          3 : 0 10000 Zuschauer
St.Pauli - Bremen            0 : 0 20600 Zuschauer
Leverkusen - Dortmund        1 : 0 22000 Zuschauer
Stuttgart - Karlsruhe        2 : 0 34000 Zuschauer
Bochum - Koeln               0 : 1 21000 Zuschauer
$
$ awk '{printf("%-20s: %6d Zuschauer; %3d Tore gefallen\n", $1, $7, $4+$6)}'
bundesliga
Muenchen          : 34000 Zuschauer; 5 Tore gefallen
Kaiserslautern   : 28260 Zuschauer; 3 Tore gefallen
Uerdingen        : 10000 Zuschauer; 3 Tore gefallen
St.Pauli         : 20600 Zuschauer; 0 Tore gefallen
Leverkusen       : 22000 Zuschauer; 1 Tore gefallen
Stuttgart        : 34000 Zuschauer; 2 Tore gefallen
Bochum           : 21000 Zuschauer; 1 Tore gefallen
$
```

Die printf Umwandlungszeichen entsprechen den der C-Funktion printf.

4. Vergleichemöglichkeiten

Vergleichsmöglichkeiten im Muster:

```
$ awk '$4 > $6 { print $1 }' bundesliga
Muenchen
Kaiserslautern
Uerdingen
Leverkusen
Stuttgart
$
```

Vergleiche mit Berechnungen:

```
$ awk '$4+$6 > 3 { printf("In %s fielen mehr als 3 Tore\n", $1)}' bundesliga
In Muenchen fielen mehr als 3 Tore
$
```

Textvergleiche:

```
$ awk '$1=="Muenchen" {printf("%d Tore im Olympiastadion\n", $4+$6)}' bundesliga
5 Tore im Olympiastadion
$
```

5. Anfangs- und Ende-Aktion

Aktionen, die einmalig am Anfang bzw. am Ende ausgeführt werden, sind wie folgt anzugeben, dazwischen steht der Programmtext, der sich auf die Datendatei bezieht.

```
BEGIN { Aktionen }  
  
    { ... }  
  
END   { Aktionen }
```

```
$ cat tabelle.awk  
BEGIN { printf("%20s - %-20s | %s\n", "Heimmannschaft", "Gastmannschaft",  
            "Ergebnis")  
        }  
  
        { printf("%20s - %-20s | %2d : %2d\n", $1, $3, $4, $6); }  
  
END { printf("-----\n") }  
$
```

```
$ awk -f tabelle.awk bundesliga
Heimmannschaft - Gastmannschaft | Ergebnis
    Muenchen - Nuernberg           | 3 : 2
    Kaiserslautern - Moenchengladbach | 2 : 1
    Uerdingen - Homburg             | 3 : 0
    St.Pauli - Bremen                | 0 : 0
    Leverkusen - Dortmund           | 1 : 0
    Stuttgart - Karlsruhed          | 2 : 0
    Bochum - Koeln                   | 0 : 1
```

\$

6. Variablen, Typen und Ausdrücke

6.1. Vordefinierte Variablen

NF - Feldanzahl der jeweiligen Eingabezeile

```
$ awk '{ print NF, $1, $(NF-1), $NF }' bundesliga
8 Muenchen 34000 Zuschauer
8 Kaiserslautern 28260 Zuschauer
8 Uerdingen 10000 Zuschauer
8 St.Pauli 20600 Zuschauer
8 Leverkusen 22000 Zuschauer
8 Stuttgart 34000 Zuschauer
8 Bochum 21000 Zuschauer
$
```

NR - Die aktuelle Zeilennummer

```
$ awk '{ print "Spiel", NR, $1 $2 $3 }' bundesliga
Spiel 1 Muenchen-Nuernberg
Spiel 2 Kaiserslautern-Moenchengladbach
Spiel 3 Uerdingen-Homburg
Spiel 4 St.Pauli-Bremen
Spiel 5 Leverkusen-Dortmund
Spiel 6 Stuttgart-Karlsruhed
Spiel 7 Bochum-Koeln
```

6.2. Benutzerdefinierte Variablen

Variablen werden automatisch durch die erste Verwendung angelegt und initialisiert.

```
$ cat auswaertsgewinner.awk
$6 > $4 { ausgew = ausgew + 1 }
END { print ausgew, "Auswaertsmannschaft(en) haben/hat gewonnen" }
$
$ awk -f variablen.awk bundesliga
1 Auswaertsmannschaft(en) haben/hat gewonnen
$
```

```
$ cat spieltagAnalyse.awk
    { sum = sum +$7 }
END {
    print NR, "Spiele"
    print "Gesamtzuschauer:", sum
    print "Durchschnitt pro Spiel:", sum/NR
}
$
$ awk -f spieltagAnalyse.awk bundesliga
7 Spiele
Gesamtzuschauer: 169860
Durchschnitt pro Spiel: 24265.7
$
```

Mit **Teilaktionen** ist eine Maximumsberechnung einfach durchzuführen:

```
$ cat meisteZuschauer.awk
$7 > max { max =$7; team =$1 }
END      { print "Die meisten Zuschauer waren in", team, "(" max ")" }
$
$ awk -f meisteZuschauer.awk bundesliga
Die meisten Zuschauer waren in Muenchen (34000)
$
```

6.3.Konkatenation von Strings

Eine Stringvariable „vereine“ schrittweise zusammengesetzt.

```
$ cat auslandsniederlagen.awk
BEGIN    { print "Folgende Mannschaften erlitten eine Auswaertsniederlage:" }
$4 > $6 { vereine = vereine $3 " " }
END      { print vereine }
$
$ awk -f auslandsniederlagen.awk bundesliga
Folgende Mannschaften erlitten eine Auswaertsniederlage:
Nuernberg Moenchengladbach Homburg Dortmund Karlsruhe
$
```

6.4.Typen und Ausdrücke

Awk kennt die Typen Number (stets. float) und Sting. Der Kontext der Verwendung von Variablen bzw. Feldern der Datei bestimmt den Typ.

Sind zwei Variable numerisch, wird bei Vergleichen numerisch verglichen, ansonsten lexikographisch.

`NF > 5` ist true, wenn mehr als 5 Felder vorhanden sind.

`$1 > "s"` ist true, wenn das erste Feld lexikographisch nach s kommt (z.B. "susi", "thil").

Soll eine Variable als **Number** behandelt werden, so ist einfach **0 zu addieren**; soll eine Variable als **String** behandelt werden, wird der **Null-String angehängt**.

```
a = 12      # a hat Wert 12.0
b = a " "   # b ist jetzt "12" nicht "12.0"
```

Zum Bilden von Ausdrücken sind die meisten von C bekannten Operatoren verwendbar (von höchster zu niedrigsten Priorität):

| | |
|--------------------------|---|
| <code>(...)</code> | Grouping |
| <code>\$</code> | Field reference. |
| <code>++ --</code> | Increment and decrement , both prefix and postfix. |
| <code>^</code> | Exponentiation (<code>**</code> may also be used, and <code>**=</code> for the assignment operator). |
| <code>+ - !</code> | Unary plus , unary minus , and logical negation . |
| <code>* / %</code> | Multiplication , division , and modulus . |
| <code>+ -</code> | Addition and subtraction . |
| <code>space</code> | String concatenation . |
| <code>< ></code> | |
| <code><= >=</code> | |

| | |
|-------------|--|
| != == | The regular relational operators. |
| ~ !~ | Regular expression match, negated match. NOTE: Do not use a constant regular expression (/foo/) on the left-hand side of a ~ or !~. Only use one on the right-hand side. The expression /foo/ ~ exp has the same meaning as ((\$0 ~ /foo/) ~ exp). This is usually not what was intended. |
| in | Array membership . |
| && | Logical AND . |
| | Logical OR . |
| ?: | The C conditional expression . This has the form expr1 ? expr2 : expr3. If expr1 is true, the value of the expression is expr2, otherwise it is expr3. Only one of expr2 and expr3 is evaluated. |
| = += -= | |
| *= /= %= ^= | Assignment . Both absolute assignment (var = value) and operator-assignment |

7.Kontrollstrukturen

Folgende Kontrollstrukturen sind verfügbar:

```
if ( expression ) statement else statement
```

```
$ cat totozahlen.awk
BEGIN { print "Die Totozahlen sind:" }
      { if ($4 > $6)
          printf("1 ")
        else if ($4 == $6)
          printf("0 ")
        else
          printf("2 ")
      }
END { printf("\n") }
$
$ awk -f totozahlen.awk bundesliga
Die Totozahlen sind:
1 1 1 0 1 1 2
$
```

for (*expression*; *expression*; *expression*) *statement*

```
$ cat gesamtnote.awk
# Notenberechnung
# Eingabe: Schuelername note1 note2 note3 ....
# Ausgabe: Schuelername: Gesamtnote
{ sum = 0
  for (i=2 ; i<=NF ; i=i+1)
    sum = sum + $i
  printf("%20s: %2.2f\n", $1, sum/(NF-1))
}
$
$ cat Klausurergebnis
Eva      1.1 1.2 3.3
Adam     3.0 2.7 5.0
$
$ awk -f gesamtnote.awk Klausurergebnis
          Eva: 1.87
          Adam: 3.57
$
```

```
while ( expression ) statement  
do statement while ( expression )
```

```
$ cat investments  
Postbank          2002 100000 4 2004  
HypoVereinsbank  2002 100000 6 2005  
$  
$ cat verzinsung.awk  
# berechne zukünftiges Zinsertraege einer Anlage  
# Eingabe: Anlagenname Anlagejahr Kapitalbetrag Zinssatz(%) LetztesJahr  
# Ausgabe: Kapital am Ende der Laufzeit  
{  
    printf("\n%s:\n", $1);  
    i = $2  
    kapital = $3  
    zinssatz = $4/100  
    while (i<=$5) {  
        printf("\t%4d : %10.2f EUR\n", i, kapital);  
        kapital = kapital * (1 + zinssatz)  
        i++  
    }  
}  
$
```

```
$ awk -f verzinsung.awk investments
```

```
Postbank:
```

```
    2002 : 100000.00 EUR  
    2003 : 104000.00 EUR  
    2004 : 108160.00 EUR
```

```
HypoVereinsbank:
```

```
    2002 : 100000.00 EUR  
    2003 : 106000.00 EUR  
    2004 : 112360.00 EUR  
    2005 : 119101.60 EUR
```

```
$$
```

Daneben existieren die aus C bekannten Anweisungen `break` und `continue`.

8.awk Funktionen

awk verfügt über zahlreiche Funktionen für

numerische Berechnungen (sin, ...)

Stringfunktionen (index, split, ...)

Zeit und Datum (strftime, ...)

Bitmanipulationen (shift, ...)

Benutzerdefinierte Funktionen sind möglich:

```
function name(parameter list) { statements }
```

Beispiel (Fakultät)

```
$ cat fakultaet.awk
# fakultaet
function factorial( n ) {
    if ( n <= 1 ) return 1
    else return n * factorial(n - 1)
}
$1 > 0 { print "the factorial of ", $1, "is ", factorial($1) }
$
```

```
$ awk -f fakultaet.awk
3
the factorial of 3 is 6
10
the factorial of 10 is 3628800
100
the factorial of 100 is 9.33262e+157
$
```

9.Reguläre Ausdrücke

Anstelle eines Musters kann auch ein regulärer Ausdruck angegeben werden. Damit hat ein awk-Programm dann die Form:

```
Muster { Aktionen }
```

oder

```
/ regulärer Ausdruck / { Aktionen }
```

Nur Zeilen eines bestimmten Musters anzeigen:

```
$ awk '/^#include/{print}' stdio.h
```

10.Beispiele

Drucke **Benutzer eines Unix Systems**, die die bash als Standard-Shell verwenden:

```
$ cat bachUser.awk
BEGIN {
  FS=":";
  print "Bash-Benutzer:"
  print "-----";
  printf "%-15s %-30s %s\n", "user-id", "Name", "Home";
  print "-----";
}
/\/bin\/bash/ {printf "%-15s %-30s %s\n", $1, $5, $6}
$
$ awk -f bachUser.awk /etc/passwd | more
Bash-Benutzer:
-----
user-id          Name                Home
-----
root             root                /root
bin              bin                 /bin
daemon           daemon              /sbin
as               Alois Schuette      /home/as
...
```

Hörsaalübung

Eine Datei mit folgendem Aufbau soll ausgewertet werden:

| Land | km2 | MioEinwohner | Kontinent |
|------|------|--------------|-----------|
| USSR | 8649 | 275 | Asien |

Feldtrenner soll TAB sein.

Ermitteln werden soll die Summe der Fläche und die Summe der Einwohner

Ein Shellskript dazu soll die Datei mit den Länderinfos vom Benutzer erfragen und dann die Auswertung mittels awk durchführen.