Standard– und Programmiertools für Unix

Stephan Thesing

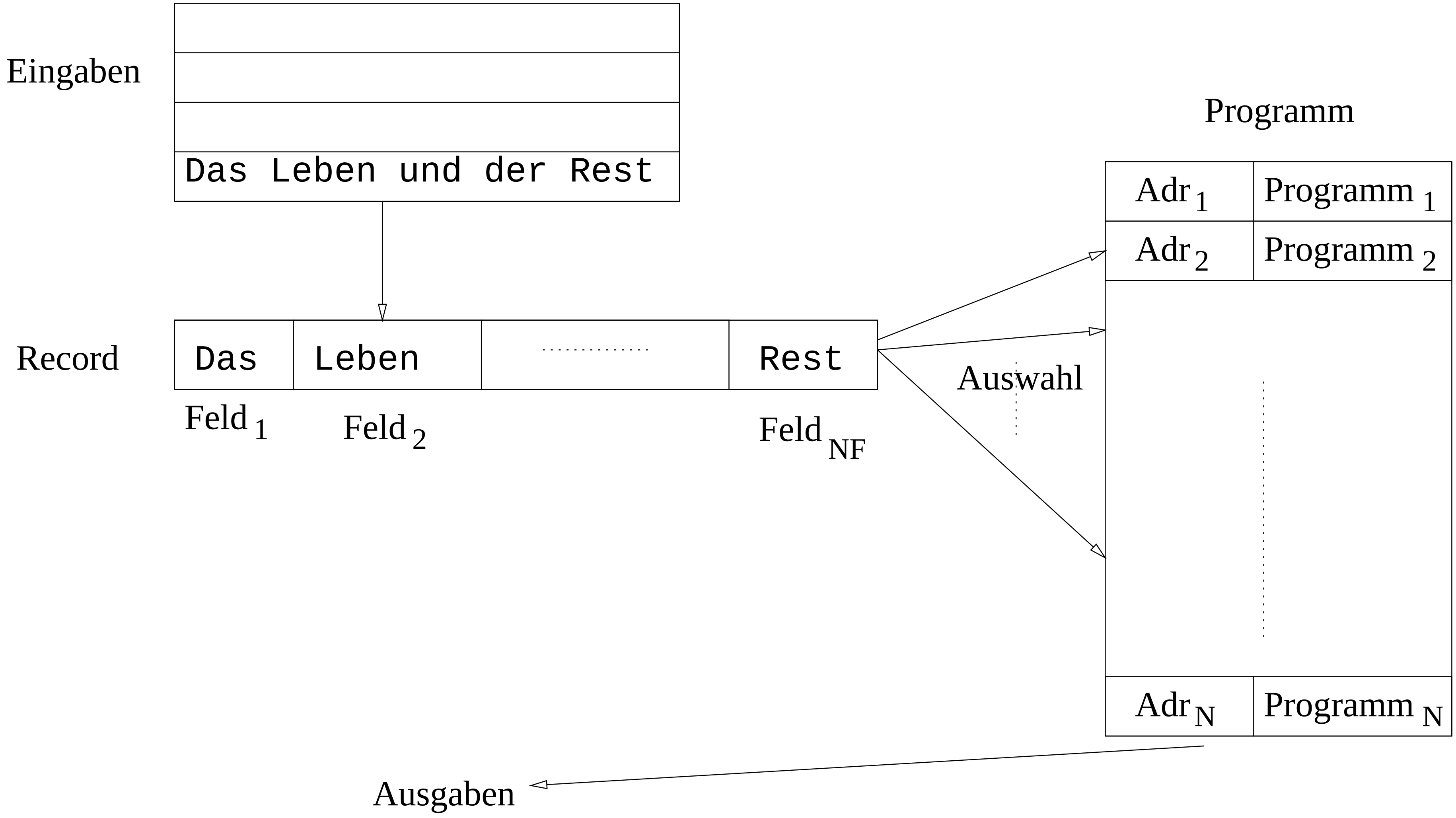
3. `awk`: Ein weiteres Tool zur Textdatenbearbeitung

Mehr Fähigkeiten als `sed`

C-ähnliche Sprache

Prinzip: Records und Felder

Auswahl von Programmstücken zur Bearbeitung bestimmter Records



Anzahl Felder in `NF`, Recordtrenner in `RS`, Feldtrenner in `FS`.

Beispiel:

```
/^[0-9]/{ anzziffern++; }
/^[a-zA-Z]/{ anzbuchst++; }
END{ printf("%d Ziffern- und %d Buchtabenzeilen\n",
      anzziffern,anzbuchst);}
```

(Adressen,Programmstück) Paare.

`BEGIN`, `END`: Vor/Nach eigentlicher Abarbeitung.

Ohne Adresse: Passt auf jeden Record.

Aufruf von `awk`:

`awk` *Programm* [*Files*]

`awk -f` *Programmfile* [*Files*]

3.1. Adressen

Neben Bereichen, Pattern auch durch Prädikate auswählbar, z.B.

```
NR % 2==0  { print "gerade Zeilennummer";}
/^[0-9]/ && NR!=10 { print "Ziffer, nicht 10. Zeile\n";}
/^[0-9]/? NR!=10: NR!=20 { next;}
NR==1,NR==20{ next;}
1,20{ next;}
BEGIN{ FS=",";}
```

4. Pattern

Mehr Pattern als `sed`.

`.` Zeichen, ausser `\n`

Klassen Z.B.: `[0-9]`, `[^a-z]`

`*` Wiederholung: 0 bis x-mal: `[0-9]*`

`+` Wiederholung: 1 bis x-mal: `[0-9]+`

| Alternative: `a|b`

`?` 0 oder 1-mal: `items?`

Klammerung : `(ab)*`

5. Programme

`awk` ist an **C** angelehnt.

Es gibt

- Variablen: Skalar oder Array. Kein Unterschied zwischen Zahlen und Strings!
- Die meisten der **C** Operatoren in Ausdrücken verfügbar
- Pattern-Matching in Ausdrücken
- `for`, `while` Schleifen
- Funktionen
- Arrays sind assoziativ
- Felder+Records als Variablen verfügbar

5.1. Ausdrücke

Variablen:

- Eingebaute Variablen: `NF`, `FS`, ...
- Benutzer-Variablen, z.B. `anzbuchst`
- Record+Felder: `$0`, `$1`, ..., `$NF`

Variable kann skalar oder Array sein.

Skalar: String oder Zahl, z.B. `"Ein String"`, `99.73`

Array: assoziativ, Index ist String: `ds[99]=$2`, `namen["Anfang"]="Ende"`

Strings werden automatisch zu Zahlen konvertiert und umgekehrt.

Also ist `ds[99]=$2` identisch zu `ds["99"]=$2`

Bei nicht möglicher Konvertierung ist das Ergebnis `0`, kein Fehler.

Mehrere Indices bei Arrays möglich: `a[1,2]=99` (intern: einzelner Index!)

Kein 'bool': `0` als 'falsch' angesehen, jede andere Zahl als 'wahr'

Operatoren:

`awk` kennt die meisten der Operatoren von **C**, z.B:

```
a+=2; b=42;
(NR==1)?42:6*9
(NR==1)||(NR==19)
42<6*9
a++; b--;
```

Dazu kennt es noch eigene Operatoren:

```
a=2^4;
name=vorname "," nachname;
name ~ /^Dr. /
file !~ /[/]tmp/
"Anfang" in namen
```

Vordefinierte Variablen:

ARGC, ARGV Kommandozeilenparameter

FS Feldseperator, z.B. `FS=",";` in `BEGIN`

RS Recordseperator, z.B. `RS="::";` in `BEGIN`

NF Anzahl Felder im Record

NR Nummer des aktuellen Records

OFS Feldtrenner für `print`

ORS Zeilentrenner für `print`

Man kann auf der Kommandozeile Variablen vorbelegen:

```
awk -f foo.awk Start=99 Ende=1000
```

Funktionsaufrufe, z.B. `s=substr(t,1,2);`

5.2. Kommandos

```
print "Die Laenge ist", len, "Meter" >"/tmp/xx";
printf("Die Laenge ist %d Meter\n",len) >>"/tmp/data." fnr;
if (NR==10) next;
if (len!=10) exit (1);
return 6*9;
```

Blockbildung:

```
if (NR==2) {s=90; print "Zeile 10";}
```

`awk` kennt vier Schleifen und ein Conditional:

```
if (NR==10) {s=45; flag=1;}
if ($2 ~ /^foo [,.]*$/) text="a"; else text="B";
do {s+=n; n--;} while (n>0);
while (n>0) { st+=n; n--; if (a[n]==1) break;}
for (i=1;i<10;i++) { if (i==2) continue; s+=i;}
for (v in werte) printf("Wert[%s]=%s\n",v,werte[v]);
```

Datenfile mit Zahlen oder Kommentaren in jeder Zeile.

Durchschnitt der Zahlenwerte:

```
/^[0-9]/{summe+=$1; anzahl++; next;}
{print "Kommentar: ", $0;}
END { if (anzahl>0) {
        durch=summe/anzahl;
        printf("%d Werte, Schnitt: %4.2f\n",
               anzahl,durch);
      } else {
        print "Keine Datenwerte";
      }
    }
```

Eigene Funktionen werden mittels `function` definiert:

```
function foo(v) { s=2^v; summe+=s; return s;}
/^[0-9]/ { t=foo($1); print t; next;}
{print $0;}
```

`awk` kennt aber schon eine ganze Menge vordefinierter Funktionen.

```
f=cos(0.45);
delete ARGV[i];
p=index($1,".");
len=length($2);
pos=match($3,/[0-9]+ [a-z]+/);
anz=split($i,res,":");
s=sprintf("%d Werte in %d Files",werte,files);
sanz=sub(/UdS/,"Uni des Saarlandes",$0);
s2=substr(s,1,10);
ecode=system("rm " fname);
s=tolower(s);
u=toupper(u);
```

5.3. Kommandozeilenparameter

-F *Sep* Entspricht `FS="`*Sep*`";` in `BEGIN`

-f *File* Programm in *File*

Var*=*Value Entspricht *`Var=Value`* in `BEGIN`

Man kann aber auch Parameter an `awk` selbst übergeben:

`ARGC` enthält Anzahl Kommandozeilenparameter.

`ARGV` enthält die Parameter (Array).

Z.B:

```
BEGIN{ for (i=1;i<ARGC;i++) {
         if (ARGV[i] ~ /^--ignore/ ) {
            ignore=1;
            delete ARGV[i];
         }
       }
}
/^#/{ if (ignore)
         next;
      print $0;
    }
```

5.4. Beispiele

Filelängen:

```
ls -l `find . -name '*.c'` |
awk '{summe+=$5;}
      END{ printf("%d bytes\n",summe);}'
```

Sortieren:

```
ls -l `find . -type f` |awk '{printf("%08d %s\n",$5,$0);}' |
  sort |
    awk '{ for (i=2; i<=NF;i++)
             printf("%s%s",(i==2)?"" : " ", $i);
           printf("\n");}'
```

5.5. Zusammenfassung

`awk` bietet die Möglichkeit, Textfiles in Records und Felder zu zerlegen

Für auswählbare Records können Programmstücke ausgeführt werden.

Assoziative Arrays

An **C** angelehnte Programmiersprache.

Probleme:

Stark an Records gebunden

Umgang mit eigenen Files schwierig

Schwach getypte Sprache