Stringverarbeitung II
Filterprogrammierung mit AWK

Christian Gollwitzer

Experimentalphysik V, Universität Bayreuth, D-95440 Bayreuth

4. Mai 2006

Geschichte

- Name nach den Initialen der Erfinder Aho, Weinberger & Kernighan (C-Erfinder)
- erste Version AT&T 1977, jetzt oawk=old awk
- nawk=new awk 1985 Bestandteil von UNIX SVR3.1
- Verbesserung von sed und egrep $\Rightarrow$ Programmiersprache (Weiter nach Perl durch Larry Wall,1987)
- Drei freie awks: **GNU awk**, mawk, »The One True Awk«
- Kommerziell: MKS awk, Interix (jetzt Microsoft), TAWK (Tompson Automation)
- Idee: Zwei Zeichen machen ein sinnvolles(!) Programm

Filterkonzept – Software Tools

- Explizit beschrieben in R. Pike & B. Kernighan: Program design in the UNIX environment
- Ein Filter liest zeilenweise von `stdin` und gibt zeilenweise nach `stdout`, Fehler nach `stderr`
- `stdin` und `stdout` können umgelenkt werden ⇒ Teil 3 (Shellprogrammierung)
- `sort` sortiert `stdin` nach `stdout`
- `sort <input >output` sortiert Datei `input` nach `output`
- `awk 'program'` liefert Universalfilter
- `#!/bin/awk -f` macht Skriptdateien ausführbar

Die kürzesten awk-Programme – how to fool a C programmer

- `awk 'NF' datei` $\Rightarrow$ Lösche alle Leerzeilen

Die kürzesten awk-Programme – how to fool a C programmer

- `awk 'NF' datei` $\Rightarrow$ Lösche alle Leerzeilen
- `awk '/regexp/'` $\approx$ `egrep 'regexp'`

Die kürzesten awk-Programme – how to fool a C programmer

- `awk 'NF' datei` $\Rightarrow$ Lösche alle Leerzeilen
- `awk '/regexp/'` $\approx$ `egrep 'regexp'`
- `awk 'END {print NR}'` Zähle Zeilen einer Datei

Die kürzesten awk-Programme – how to fool a C programmer

- `awk 'NF' datei` $\Rightarrow$ Lösche alle Leerzeilen
- `awk '/regexp/'` $\approx$ `egrep 'regexp'`
- `awk 'END {print NR}'` Zähle Zeilen einer Datei
- `awk '{print NR, $0}'` Zeilennummern einfügen

Die kürzesten awk-Programme – how to fool a C programmer

- `awk 'NF' datei` $\Rightarrow$ Lösche alle Leerzeilen
- `awk '/regexp/'` $\approx$ `egrep 'regexp'`
- `awk 'END {print NR}'` Zähle Zeilen einer Datei
- `awk '{print NR, $0}'` Zeilennummern einfügen
- Zauberei??? $\Rightarrow$ Pattern-Action Prinzip

Pattern-Action Modell

- awk-Programm besteht aust Reihe von Mustern und Aktionen:

  ```
  pattern 1 {action 1}
  pattern 2 {action 2}...
  ```

- Sowohl *pattern* (true) als auch *action*(print) können fehlen
- Input wird zeilenweise gescannt. Jedes Pattern, das zutrifft, führt seine Aktion aus.
- *pattern* ist bool'scher oder regulärer Ausdruck (gefunden=wahr).
- Viele interne Variablen: NR, NF, $0, $1..$NF
- Splitting der Felder mit FS, Zeilentrenner RS

Beispiele: One-Liner

- Zeige jede dritte Zeile, d.h. Zeile 1,4,7,...
- Bedingung: Anzahl gelesener Zeilen$/3 \equiv 1 (\mathrm{mod}\ 3)$

Beispiele: One-Liner

- Zeige jede dritte Zeile, d.h. Zeile 1,4,7,...
- Bedingung: Anzahl gelesener Zeilen$/3 \equiv 1 (\mathrm{mod}\ 3)$
- Pattern : `NR%3==1`
- Action: `print`, i.e. weglassen
- `awk 'NR%3== 1'`

Beispiel: Rheologische Daten umsortieren

- Daten der Form

```
# Header
# Strom I[-3] = -2 A
1 456 456
2 234 234
# Header
# Strom I[-3] = -1 A
1 345 345
2 156 156
```

- Möchte Strom als zusätzliche Spalte haben
- Idee: Suche `Strom` $\Rightarrow$ setze `strom`
- Falls Zahlen am Anfang: Drucke `strom`, dann Zeile,
- Sonst immer: Drucke Zeile

Beispiel: Rheologische Daten umsortieren

- Daten der Form

```
# Header
# Strom I[-3] = -2 A
1 456 456
2 234 234
# Header
# Strom I[-3] = -1 A
1 345 345
2 156 156
```

- Möchte Strom als zusätzliche Spalte haben
- Idee: Suche `Strom` $\Rightarrow$ setze `strom`
- Falls Zahlen am Anfang: Drucke `strom`, dann Zeile,
- Sonst immer: Drucke Zeile
-

```
awk '/Strom/ {strom=$5}
/^[0-9]/ { print strom, $0; next }
1' rheologie.dat
```

Beispiel: Schwierigkeitsindex LIX

- Definition: Mittlere Satzlänge (in Wörtern) + Prozentzahl langer Wörter (Björnsson 1968)
- Benötigte Parameter: Anzahl Wörter, Anzahl Sätze, Anzahl langer Wörter ($>$ 6 Buchstaben)
- Trick in awk: Setze in `BEGIN`-Regel `RS` auf Satzzeichen liefert Sätze in `$0` und Wörter in `$1..$NF`
- Benutze `match`, um die Länge eines Wortes zu messen (Interpunktion kan noch vorn und hinten dran sein)

Beispiel: Schwierigkeitsindex LIX


```
#!/bin/gawk -f
#
BEGIN { RS="[.;!?:]"}
{ anzahl_woerter+=NF
    for (i=1; i<=NF; i++) {
        match($i, /[a-zA-ZäöüßÄÖÜ]+/)
        if (RLENGTH>6) {
            anzahl_schwerewoerter++
            print substr($i, RSTART, RLENGTH)
        }
    }
}
END { print anzahl_woerter, anzahl_schwerewoerter, NR
    print anzahl_woerter/NR +
100*anzahl_schwerewoerter/anzahl_woerter }
```

Beispiel: Assoziative Arrays – Wörter nach Häufigkeit

- Arrays in awk werden mit Strings indiziert (Hash-Table):
  `Freundin["Hans"]="Berta"`
- klassisches Beispiel: Zähle die Häufigkeit von Wörter in einem Dokument
  `wortnr["und"]++`
- Einbau in unser Lix-Programm
- Auflisten aller Elemente eines Arrays:

```
for (index in array) {
    print index, array[index] }
```

- Sortierung ist willkürlich

Literatur

- Effective AWK-Programming, Arnold Robbins, O'Reilly
  `http://www.gnu.org/software/gawk/manual/gawk.pdf`
- The One True Awk – das Original
  `http://cm.bell-labs.com/cm/cs/awkbook/index.html`
- Sed&Awk, O'Reilly
  `http://www.oreilly.de/catalog/sed2/`
- Microsoft Windows Services for Unix
  `http://www.microsoft.com/technet/interopmigration/unix/sfu/default.mspx`
  `http://www.heise.de/newsticker/meldung/43648`