Seminararbeit

Awk

Michael Gasser

16. Februar 2011

Betreuer: Martin Avanzini

Zusammenfassung (Englisch)

This article is about the program language Awk . Awk is not the modernest language, but it is a still used language. Awk is designed for specific tasks, such for simple text manipulation. It is also present quite on every system. Awk is easy to understand and everybody can it fast learn. This article describes how a Awk -program is built, how a Awk -program works and what the features of Awk are. The article shows also the differences between the program language Awk and its precursor the UNIX-tool sed.

Inhaltsverzeichnis

1 Einleitung

Programmiersprachen sind fast so vielfältig wie natürlichen Sprachen und können gleich wie diese anhand von Gemeinsamkeiten in Gruppen eingeteilt werden. Gleich wie natürliche Sprachen sind manche Programmsprachen bekannter und werden von mehreren Menschen gesprochen als andere. Einige Programmiersprache haben jedoch ihren eigenen Nischenplatz gefunden und entwickeln sich dort weiter, während andere Sprachen verschwinden. Awk ist eine Skriptsprache, die sich auf das Arbeiten mit Daten in spaltenförmig organisierten Textdateien spezialisiert hat. Die tägliche Arbeiten mit Textdateien sollen mit Awk einfach und elegant erledigbar sein.

1.1 Geschichte

In den 70er Jahren wurden die meisten Arbeiten an Textdateien mit einfachen nicht-interaktiven Stream-Editoren wie Sed erledingt. Diese haben eine eher unkomfortable Handhabung und eine begrenzte Funktionalität. Aus diesen Grund und um zu zeigen, dass ein gleichzeitiges Verarbeiten von Zahlen und Strings möglich ist, entwickelten 1977 A.V.**A**ho, P.J.**W**einberger und B.W.**K**ernighan eine neue Skriptsprache. Die Sprache sollte ein einfaches und mächtiges Programmierwerkzeug mit den sich die täglich anfallenden Analysen und Manipulationen von Daten in Textformat leicht und elegant durchführen lassen. Um den Namen ihrer neuen Programmiersprache hatten sich die Entwickler keine großen Gedanken gemacht. Den Namen setzten sie einfach aus den ersten Buchstaben ihrer Nachnamen zusammen – Awk . Awk kam besser an als erwartet. Die Community von Awk vergrößerte sich stetig und so wurde der Interpreter von Awk eines der ersten Standarttools in Unix Version 3. Da Awk immer öfters auch für größere Aufgaben benutzt wurde, wurde 1985 eine verbesserte Version eingeführt. Diese Version führte benutzerdefinierte Funktionen und eine Reihe weiterer Verbesserungen ein. Mittlerweile gibt es noch weitere Varianten von Awk , die sich teils im Funktionsumfang unterscheiden.

1.1.1 Versionen

Awk ist in verschieden Varianten vorhanden, die teils unterschiedlich mächtig sind. Dadurch ist Awk nicht immer ohne Änderungen portierbar. Eine Version von Awk ist heutzutage standardmäßig auf jeden Unixsystem zu finden und auch für Windows gibt es portierte Pakete. Für Linuxsysteme ist standardmäßig Gawk installiert. In diesem Artikel werden vor allen die Versionen Nawk und Gawk beschrieben.

1.2 Charakeristika

Awk ist eine Skriptsprache zur eleganten und einfachen Manipulieren und Auswerten von ein oder mehrerer Textdateien. Die Daten sollten dabei spaltenförmig organisierten sein, um alle Vorteile von Awk auszunutzen. Awk durchläuft die Dateien zeilenweise und untersucht die Daten nach bestimmten Textpattern. Trifft ein Pattern zu, werden je nach Pattern eine gewisse Aktionen ausgeführt.

Variante	Beschreibung
Awk	Ursprüngliche Version (1977). Auch Oawk genannt.
Awka	Freier Awk -Compiler
Mawk	Freier Awk von Michael Brennan Auf Performance optimierte Version, geringere Funktionalität
Nawk	New Awk . Erweitere Version (1985)
Gawk	Freie GNU-Version von Nawk mit Erweiterungen. Deutlich schneller, kaum Beschrenkungen
Tawk	Nawk-Version mit sehr vielen Erweiterungen und Compiler

Tabelle 1: Übersicht über die verschieden Awk -Versionen [2]

Dies kann eine einfache Ausgabe oder eine Zwischenspeicherung eines Werts für eine spätere Auswertung sein. Awk -Programme sind üblicherweise sehr kurz und da ein Awk -Interpreter auf jeden Unix-System vorhanden ist, lassen sich Awk -Programme leicht portieren. Auch komplexere Aufgaben lassen sich in Awk lösen. Übersteigt der Umfang der zu verarbeitenden Datenmenge jedoch ein gewisses Maß, sinkt die Ausführungsperformance stark ab. Awk erlaubt eine schnelle und interaktive Entwicklung von Programmen. Ein großer Vorteil von Awk ist die sehr schnelle und einfache Erlernbarkeit. Da mit Kernighan einer der Entwickler von C mitwirkte, sind die Syntax und einige Konzepte von Awk stark an C angelehnt. Für Programmierer die bereits C können, ist Awk innerhalb kürzester Zeit erlernbar.

1.3 Einsatzgebiete

Typische Einsatzgebiete für Awk sind [1]:

- Extrahierung von Dateiinhalten
- Dateikonvertierung
- Reportgenerierung
- Filtern von Datenflüsse innerhalb einer Pipe
- Prototyping

Awk -Skripte werden oft auch in Shell-Skripte integriert.

2 Überblick

2.1 Aufruf

Ein Awk -Programm kann über verschieden Arten aufgerufen werden. Entwerder per Programmdatei:

awk [*Optionen*] **-f** *Programmdatei* [*Eingabedateien*]

oder es kann auch der Programmcode direkt als Kommandoargument übergeben werden:

awk [*Optionen*] '*Programmcode*' [*Eingabedateien*]

Die '...' werden benötig, damit Sonderzeichen nicht von der Shell vorinterpetiert werden. Werden weitere Argumente übergeben, werden diese wie in C in einen Array ARGV abgelegt. Die Variable ARGC liefert die Zahl der übergebenen Kommandozeilen-Argumente. Mit der Option -F kann das Trennzeichen für Wörter (FS) festgelegt werden und mit der Option -v können weitere Systemvariablen gesetzt werden.

2.2 Grundprinzip

Ein Awk -Programm besteht aus einer Folge von Regeln. Jede Regel wird dabei aus einen Pattern-Aktion-Paar gebildet. Als Eingabe werden den Programm eine oder mehrere Dateien übergeben. Während der Programmausführung werden die Dateien in einer Leseschleife zeilenweise durchlaufen und auf alle Pattern hin überprüft. Trifft eines der Pattern zu, wird die dazugehörige Aktion ausgeführt. Weiters gibt es noch zwei spezielle Pattern, die jeweils nur einmal ausgeführt werden. Das *BEGIN*- und *END*-Pattern. Die *BEGIN*-Regel wird als allererste Regel vor den Beginn der Datenverarbeitung ausgeführt. Sie dient vor allem für Initialisierungen von Variablen. Die *END*-Regel wird als letzte Regel nach der Abarbeitung der letzten Zeile gestartet und wird meist für Ausgaben und Reportgenerierung verwendet.

2.3 Eingabedateien

Die als Parameter übergeben Textdateien werden in Awk automatisch eingelesen und bereitgestellt. Werden keine Dateien angegeben, wird von der Standardeingabe (stdin) gelesen. Die Daten werden dann anhand von Trennzeichen, die von speziellen Systemvariablen definiert werden, aufgespalten. Während des Einlesen der Dateien in der Leseschleife werden die Daten in Datensätze aufgeteilt. Ein Datensatz entspricht in den meisten Fällen einer Zeile. Diese Datensätze werden dann weiters mittels der Systemvariable FS in einzelne Wörter aufgeteilt. FS ist standardmäßig mit " " (Leerzeichen) voreingestellt. Die beim Datendurchlauf jeweils aktuellen Daten werden in Feldvariablen gespeichert. Der aktuelle Datensatz wird in $0 gespeichert. Die einzelnen Wörter werden dann nach ihren Indexnummer in $1, $2,... gespeichert.

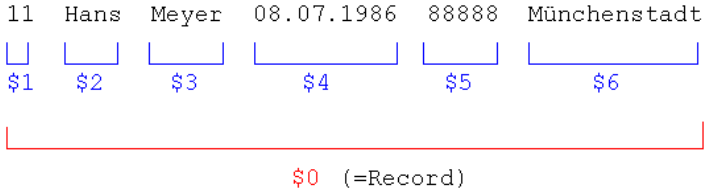

Abbildung 1: Beispiel einer Aufteilung eines Datensatzes [1]

einfaches Beispiel

Das folgenden Beispiel zeigt eine Berechnung der Punkteanzahl einer Proseminargruppe. Als Eingabe dient eine Textdatei mit den Schülernamen, der Anzahl der anwesenden Stunden und den Ergebnissen aus 3 Tests.

Susanne	8	100	89	87
Walter	10	23	58	68
Richard	6	75	67	57
Lisa	8	80	90	52

Abbildung 2: Proseminar.txt

Das folgende Skript berechnet die Gesamtpunkteanzahl d er Tests pro Student.

```
{ total = $3+$4+$5
  print $1, total}
```

Dabei wird für jede Zeile die Summe der Werte der 3., 4. und 5. Spalte in der Variable 'total' gespeichert. Das Ergebnis wird dann mit den Namen des Studneten ausgegeben.
Ausgabe:

```
Susanne 276
Walter 149
Richard 199
Lisa 222
```

2.4 Schematischer Programmablauf

Anweisungen und Eingabedateien eines Awk -Programms werden in einer bestimmten Reihenfolge abgearbeitet. Dabei werden bestimmte build-in Variablen benutzt.

1. Initialisierung interner Variablen
   Systemvariablenwerte die beim Programmaufruf per -F oder -v-Option spezifiziert wurden, werden gesetzt. Alle anderen Systemvariablen werden mit Standardwerten belegt.

2. Ausführung aller BEGIN-Regeln (falls vorhanden) in der Reihenfolge ihres Auftreten
   Enthält eine BEGIN-Aktion einen exit-Befehl, wird das Einlesen der Daten übersprungen und direkt die END-Regel ausgeführt.

3. Leseschleife
   Führt die Schleife so lange aus, bis alle Daten von allen übergebenen Dateien oder von der Standard-Eingabe abgearbeitet wurden.

a) Für jeden Datensatz in der aktuellen Datei:
   Eingabedaten werden zeilenweise in $0 eingelesen.
   Der aktuellen Datensatz wird (anhand von FS) in Wörter zerlegt und in $1 bis $NF gespeichert.

   i. Für jede Regel in der Reihenfolge ihrer Definition:
      Der aktuelle Datensatz wird auf das Pattern geprüft. Ist Pattern erfüllt, wird die dazugehörige Aktion ausgeführt.

4. Ausführung aller END-Regeln (falls vorhanden) in der Reihenfolge ihres Auftreten

[2]

3 Programmstruktur

Ein Awk -Programm besteht aus einer Folge von Regeln und Funktionsdefinitionen, die durch neue Zeile getrennt werden. Die Reihenfolge der Regeln ist zu beachten, da die Regeln in der Reihenfolge der Definition auf die Eingabedaten angewendet werden. Standardmäßig wird folgende Abfolge von Regeln und Funktionsdefinitionen vorgeschlagen [2] [3]:

BEGIN-Regel
Regel_1 ... Regel_n
END-Regel
Funktionsdefintion ...

3.1 Regeln

Eine Regel hat folgende Form:

Patter **{** Aktion **}**

Gegenfalls kann entweder das Pattern oder die Aktion fehlen. Das Pattern dient zur Filterung der Daten. Es muss erfüllt sein, damit die dazugehörige Aktion ausgeführt werden kann. Ist kein Pattern angeben, wird die Aktion für jede Zeile ausgeführt. Eine Aktion beschreibt, was mit den Datensatz zu tun ist. Ist keine Aktion angegeben wird einfach der jeweilige Datensatz ausgegeben. Kommentare können mit # gekennzeichnet werden.

3.1.1 Pattern

Ein Pattern kann sein:

Logischer Ausdrück

Ein logischer Ausdruck ist eine Verknüpfung von Operanden mit Operatoren wie Negation !, Vergleiche =,<,> oder Beinhaltungsabfrage ∼. Weiters können komplexere logische Ausdrücke aus Verknüfungen verschiedener Patterntypen mittels && (logisches UND), || (logisches ODER) und ! (logische Negation) zusammengesetzen werden. Die dazugehörige Aktion wird für jede Zeile ausgeführt, für die der logische Ausdruck wahr ist. Dabei wird 0 und "" (leere Zeichenkette) als falsch interpretiert und alle anderen Zahlen und Strings als wahr.
Beispiel:

```
$3+$4+$5>150 && $2>=8  {print $1,"bestanden"}
```

Ausgabe bei Eingabedatei 2:

```
Susanne bestanden
Lisa bestanden
```

Regulärer Ausdruck

Ein regulärer Ausdruck wird in Awk mit Schrägstrichen /.../ geklammert und setzt sich aus einfachen Zeichen, Escape-Sequenzen und Metazeichen zusammen. Einige Metazeichen für einfache reguläre Ausdrücke sind:

\	Metazeichen quoten
^	Anfang eines Strings
$	Ende eines Strings
.	Beliebiges einzelnes Zeichen
[ ]	Zeichenklasse
[^]	Invertierte Zeichenklasse

Reguläre Ausdrücke können weiters mit | (Alternation), Konkatenation, **( )** (Gruppierung), ***** ('beliebig oft'), **+** ('mindestens einmal') oder **?** (Option) zusammengefasst werden. Reguläre Ausdrücken können auch dynamische durch Strings oder durch Variablen erzeugt werden. Dynamische reguläre Ausdrücke unterliegen allerdings Performance-Einbußen. Die dazugehörige Aktion wird für jede Eingabezeile ausgeführt, die mit den regulären Ausdruck übereinstimmt.
Beispiel:

```
/(Wal|Sus).*er/
```

Ausgabe bei Eingabedatei 2:

```
Walter 10 23 58 68
```

Bereichsdefinition

Eine Bereichsdefinition setzt sich aus zwei Pattern zusammen, die durch einen Beistrich getrennt sind. Damit wird ein Bereich innerhalb einer Datei definiert. Dieser Bereich reicht vom Datensatz auf den das erste Pattern passt, bis einschließlich des Datensatzes auf den zweite Pattern passt. Dies kann beliebig oft

für ein Eingabedatei zutreffen. Treffen beide Pattern auf eine Datensatz zu, besteht der Bereich nur aus diesen einen Datensatz. Die dazugehörige Aktion wird für jede Eingabezeile ausgeführt, die innerhalb des defnierten Bereichs liegt. Beispiel:

```
$3==100,$4==67   {print $0}
```

Ausgabe bei Eingabedatei 2:

```
Susanne 8 100 89 87
Walter 10 23 58 68
Richard 6 75 67 57
```

BEGIN oder END

Die beiden Pattern BEGIN und END werden im Gegensatz zu den andern Pattern nur einmal ausgeführt. BEGIN ist vor der dem Lesen der ersten Datei wahr und wird somit vor der eigentlichen Datenabarbeitung ausgeführt. END ist nach dem Lesen der letzten Datei wahr und ist somit die letzte ausgeführte Regel. Die Pattern BEGIN und END können nicht mit anderen Pattern verknüpft werden.

3.1.2 Aktion

Eine Aktion ist eine Folge von Anweisungen, die mit Strickpunkt oder neuer Zeile getrennt werden. Die Syntax der meisten Anweisungen in Awk entspricht dabei der Syntax der analogen C-Anweisung. Wird eine Anweisung zu lang, kann sie mit "/" in der nächsten Zeile fortgesetzt werden.

Kontrollstrukturen

Awk hat eine Reihe von Kontrollstrukturen mit denen der Ablaufen des Programms gesteuert werde kann.

- Mit Verzweigungoperation wie if-then-else wird nach der Überprüfung einer Bedingung, entweder der then- oder der else-Zweig ausgeführt.

  if (*Bedingung*) *Anweisung* [**else** *Anweisung*]

  Für Zuweisungen kann auch folgende Anweisung benutzt werden:

  Variable = (*Bedingung*)**?** *WertTrue* **:** *WertFalse*

  Je nach dem ob die Bedingung wahr oder falsch ist, wird der Variablen der Wert für wahr oder Wert für falsch zugewiesen.

- Mit Schleifen können Anweisungen solange wiederholt werden, bis eine Bedingung nicht mehr gilt. Bei der abweisenden Schleifen (while-Schleife) findet die Überprüfung der Bedingung jeweils vor den Ausführen der Anweisungen statt.

while (*Ausdruck*) *Anweisung*

Bei nicht abweisenden Schleifen (repeat-until-Schleife) findet die Überprüfung jeweils nach den Ausführen der Anweisungen statt.

do *Anweisung* **while** (*Ausdruck*)

Bei einer Zählscheife (for-Schleife) wird pro Durchlauf eine Variable rauf bzw. runter gezählt, bis eine gewisse Bedingung nicht mehr erfüllt ist.

for (*Ausdruck1*; *Ausdruck2*; *Ausdruck3*) *Anweisung*

Mit der for-in-Schleife kann über Arrays iteriert werden.

for (*Variable* **in** *Array*) *Anweisung*

Die Variable durchläuft alle Indizes eines Arrays. Die Durchlaufreihenfolge ist implementationsabhängig. Weiters sollten keine neuen Arrayelemente während des Schleifendurchlaufs hinzugefügt werden, da ansonsten das Ergebnis nicht vorhersehbar ist.

- Mit kontrollierten Sprüngen können Programmabschnitte übersprungen werden. Mit **continue** wird sofort mit der nächsten Iteration der umschließenden Schleife fortgefahren und mit **break** wird die Schleife vorzeitig beendet. **exit** [*Ausdruck*]und **return** [*Ausdruck*]beenden das Programm bzw. die jeweilige Funktion und geben einen Ausdruck oder 0 zurück. Mit **next** wird zum nächsten Datensatz gesprungen und die Überprüfung der Pattern beginnt wieder bei der ersten Regel. Die Anweisung **nextfile** (nur in Gawk vorhanden) beendet das Einlesen der aktuellen Eingabedatei und fährt mit den Einlesen der nächsten Eingabedatei fort. Die Verarbeitung beginnt dabei wieder bei der ersten Regel.

Ausgabeoperationen

Für die Ausgabe stehen in Awk zwei Funktionen zu Verfügung:

print [*Ausdruck1 Ausdruck2, ...*]

und

printf(*Format, Ausdruck1, Ausdruck2,...***)**

Die Anweisung print gibt die Ausdrücke, in der angegeben Reihnfolge und durch ein Zeichen (OFS) getrennt, auf der Standardausgabe (stdout) aus. Anschließend wird noch ORS ausgegeben. Ist kein Ausdruck angegeben wird $0 ausgegeben. Die Anweisung printf hingegen, gibt die Ausgabe in einer formatierten Weise aus. printf wird wie in C benutzt, ein Formatierungsstring beschreibt wie die Ausdrücke ausgegeben werden sollen.
Mit >*Dateiname*, >>*Dateiname* und | *Kommando* am Ende der Ausgabe-Funktion, kann die Ausgabe in Dateien oder Kommandos umgeleitet werden. Geöffnete Dateien oder Pipes sollten per **close(***Dateiname***)** oder **close(***Kommando***)** wieder geschlossen werden.

Eingabeoperationen

Mit der Funktion **getline** [*var*] kann explizit der nächste Datensatz der aktuellen Eingabedatei gelesen werden. Der Datensatz wird dabei entweder die Variable var oder $0 gespeichert. Die Funktion kann drei Rückgabewerte zurück liefern. 1 falls die der Datensatz gelesen werden konnte, 0 falls das Dateiende erreicht wurde oder -1 falls ein Fehler auftrat. Anschließend wird mit der Abarbeitung des aktuellen Datensatztes fortgefahren. Mit **getline** < *Dateiname* oder *Kommando* | **getline** können Datensätze auch aus Dateien oder von der Standardausgabe eines Kommandos lesen werden.
Mit der Funktion **system(**ausdruck**)** kann ein als Ausdruck übergebenes Kommando explizit ausgeführt werden. Der Rückgabewert ist der exit-Status des ausgeführten Kommando.

Operatoren

Operatoren in Awk mit abnehmender Priorität [1]:

Operator	Bedeutung	Assoziation
()	Gruppierungs-Operator	links-assoziativ
$	Feldopator	links-assoziativ
++ –	Postfix-Operator	links-assoziativ
++ –	Präfix-Operator	links-assoziativ
^ **	Potenzierungs-Operator	rechts-assoziativ
!	logische Negation	links-assoziativ
+ -	Vorzeichen-Operation	links-assoziativ
* / %	Multiplikation, Division , Modulo	links-assoziativ
+ -	Addition, Subtraktion	links-assoziativ
	Stringkonkatentation	links-assoziativ
<<= == != >= >	relationale Operatoren	links-assoziativ
~ !~	Beinhaltungs-Opartoren	links-assoziativ
in	Array-Mitglieds-Operation	links-assoziativ
&&	logisches UND	links-assoziativ
||	logisches ODER	links-assoziativ
? :	Bedingungs-Operation	rechts-assoziativ
= += -= *= /= %= ^= **=	Zuweisungs (inkl. Operation)	rechts-assoziativ

3.2 Datentypen

Awk kennt nur zwei Basisdatentypen

- Strings
- Zahlen

Strings werden mit "..." geklammert und können beliebig lang sein. Der Speicherplatz von Strings wird automatisch verwaltet und muss nicht explizit reserviert oder freigeben werden. Zahlen können sowohl Ganze Zahlen als auch Fliesskommazahlen sein. Intern werden Zahlen jedoch nur als 16-Bit Fliesskommazahlen verwaltet. Dadurch haben verschieden Darstellungen einer Zahl den selben Wert.
Eine Variable kann einen der beiden Basisdatentypen haben oder auch eine

Mischung von beiden sein. Konstanten hingegen können nur einen der beiden Basisdatentypen haben. Der jeweilige Typ von Variablen und Konstanten wird aus den jeweiligen Kontext ermittelt und kann sich bei Variablen dynamisch ändern, je nach dem welche Operationen ausgeführt werden.

Typkonvertierung

Bei arithmetischen Operationen ist das Ergebnissen immer ein nummerischer Wert. Dabei wird alles was wie als Zahl aussieht als Zahl behandelt, auch wenn es in Wirklichkeit ein String ist. Der nummerische Wert eines String ist dabei der Wert des nummerischen Präfixes, wobei Leerzeichen werden ignoriert werden.
Beispiel : " 123ab" + 1 → 124
Eine Stringoperation liefern als Ergebnis immer einen String. Ebenso werden auch die einzelnen Operanden in Strings umgewandelt.

Arrays

In Awk sind ein- oder mehrdimensionale Arrays erlaubt. Arrays müssen nicht explizit deklariert oder ihre Größe im voraus angegeben werden. Array oder Arrayelement können aber explizit mit *delete* gelöscht werden. Einen Array können sowohl Strings als auch Zahlen zuwiesen werden, wodurch Arrays meisten den Mischdatentyp aus Zahl und String haben. Auf Arrayelemente wird per Arraynamen und Index in eckigen Klammern zugegriffen (z.B. a[1]). Ist bei einen Zugriff, der Index des Elements noch nicht vorhanden, wird ein neues Element einfügt und mit 0 bzw. Leerstring vorinitialisiert. Arrays können in Awk , anders als in anderen Sprachen, sowohl mit Zahlen als auch mit Strings indiziert werden. Intern werden Arrays jedoch nur über Strings verwaltet. Daher wird bei einen Zugriff auf einen Index über eine Zahl und der Zahl als String, immer auf das selbe Element zugegriffen.
Beispiel:

```
BEGIN { a[5]=2; a[''5'']=8; print a[5] }
```

Ausgabe: 8
Indizes mehrdimensionaler Arrays werden mit Beistriche getrennt. Wird ein mehrdimensionaler Index in Operationen verwendet, muss er durch runde Klammern eingeschlossen. In Awk werden mehrdimensionale Array nur mit mehreren eindimensionalen Array simuliert. Awk konkateniert die einzelne Indizes mit einen Trennzeichen aus der Systemvariablen SUPSEP dazwischen und fasst den erhalten String dann als eindimensional Index auf. Weiters sind Arrays in Awk unsortiert. Insgesamt entsprechen Arrays in Awk eher einen Hash.

3.3 Variablen

In Awk gibt es 3 Typen von Variablen

Benutzerdefinierte Variablen

Variablen in Awk brauchen keine Deklaration, sondern werden mit den ersten Benutzung automatisch angelegt und sind mit 0 bzw. Leerstring vorinitialisiert. Der Typ einer Variable kann sich je nach Benutzung dynamische ändern. Alle Variablen sind global und von überall im Programm zugreifbar. Awk hat einen Garbage Collection, dadurch muss man keinen Speicherplatz explizit reservieren oder freigeben.

Feld-Variablen

Feldvariablen erlauben einen komfortablen Zugriff auf die aktuellen Daten der Leseschleife. Mit $0 kann auf den gesamten Datensatz zugegriffen werden und mit $ und der Indexnummer des Wortes auf die einzelnen Wörter: $1, $2, ..., $NF. Auf Feldvariablen kann gleich wie auf andere Variablem zugegriffen werden und ihr Inhalt verändert werden. Einen Änderung einer Feldvariablen hat Auswirkung auf andere Feldvariablen, aber keine auf die Datei selbst. Bei einer Änderung eines Wortes in $i wird auch $0 geändert. Wird $0 geänderten werden alle Inhalte von $1,$2,... und NF neu gesetzt.

Systemvariablen (Build-in Variablen)

Awk besitzt eine Reihe von Systemvariablen. Diese werden für die interne Bearbeitung und die Unterteilung der Eingabedaten benutzt.

Variablenname	Bedeutung	Voreinstellung
ARGC	Anzahl der Argumente in der Kommandozeile	-
ARGV	Array mit Kommandozeilenargumente	-
ENVIRON	Array mit Umgebungsvar. und ihren Werten	-
FILENAME	Name der aktuellen Eingabedatei	-
FNR	Datensatznummer der aktuellen Eingabedatei	-
FS	Wötertrenner	" "
NF	Anazhl Wörter in aktuellen Datensatz	-
NR	Anzahl bisher eingelesener Datensätze	-
OFMT	Ausgabeformat für Zahlen in print	"%.6g"
OFS	Ausgabe-Worttrenner in print	" "
ORS	Ausgabe-Datensatztrenner in print	"\n"
RLENGTH	Länge des ermittelten Strings bei *match*	-
RSTART	Beginn des ermittleten Strings bei *match*	-
RS	Datensatztrenner/delimiter	"\n"
SUPSEP	Trennzeichen für mehrdimensionalen Arrayindex	"\034"

Tabelle 2: Liste der wichtigsten Systemvariablen in Awk [4]

3.4 Funktionen

Benutzerdefinierte Funktionen

Ab der 2. Version von Awk (nawk) können eigene Funktionen definiert werden. Funktionen in Awk benötigen keine vorherige Deklaration und können sich rekursiv aufrufen. Eine Funktionsdefinition sind überall dort möglich, wo ein Pattern-Aktions-Paar erlaubt ist. Funktionen werden aber in der gängigen Praxis am Programmende geschrieben. Eine Funktionsdefinition hat folgende Form:

function *Funktionsname* **(***Parameterliste***) {***Anweisungen***}**

Der Funktionsname muss eindeutig gewählt werden und darf keinen Name einer build-in Funktion haben. Die einzelnen Parameter der Parameterliste werden durch Beistriche getrennt. Werden zu wenige Parameter beim Funktionsaufruf übergeben, werden die restlichen Parameter mit 0 bzw. " " initialisiert. Bei einfachen Variablen als Parameter wird nur ein Kopie übergeben (call-by-value). Dies ist die einzige Möglichkeit in Awk eine lokale Variablen zu simulieren.
Wird ein Array übergeben, wird keine Kopie sondern eine Referenz übergeben ("call-by-reference"). Wird das Array innerhalb der Funktion geändert, wird auch das Original geändert. Mit einer *return*-Anweisung wird die Funktion beendet und ein Wert zurückgeliefert. Ist keine *return*-Anweisung vorhanden, wird die Funktion nach der letzten Anweisung verlassen.

Funktionsbibliotheken

Um auch später wieder auf Funktionen zugreifen zu können, können Funktionen in Awk -Dateien gesammelt werden. Will man eine dieser Funktion benutzen, muss die jeweilige Datei beim Awk -Aufruf dazu gelinkt werden:

awk -f *Speicherort_der_Datei* **-f** *Awk -Programmdatei*

Build-in Funktionen

Awk besitzt bereits eine Reihe fix eingebauter String- und Arithmetik-Funktionen.

Funktion	Beschreibung
atan2(y,x)	Arcustangens von x/y
cos(x)	Cosinus von x (x im Bogenmaß)
exp(x)	Exponentialfunktion
int(x)	ganzzahliger Anteil von x
log(x)	natürlicher Logaithmus von x
rand()	Zufallszahl im Intervall [0,1]
sin(x)	Sinus von x (x im Bogenmaß)
sqrt(x)	Quadratwurzel von x
srand(x)	initialisiert Zufallsgenerator

Tabelle 3: Arithmetik-Funktionen in awk [2]

Funktion	Beschreibung
gsub(r,s,t)	Ersetzt in String t alle Strings r durch s
index(s,t)	Sucht das erstes Vorkommen von t im String s
length(s)	Bestimmt die Länge des Strings s
match(s,r)	Überprüft ob String r ein Teilstring in s abdeckt
split(s,a)	Zerlegt s entsprechend FS und speichert Wörter im Array v
sprintf(f,a1,a2,...)	Formatiert Argumente wie printf, allerdings ohne Ausgabe
sub(r,s,t)	Ersetze in t den linkesten Teilstring r durch s
substr(s,p)	Liefert ab Position p den Rest von s
substr(s,p,n)	Liefert ab Position p den Teilstring aus s mit n Zeichen

Tabelle 4: String-Funktionen in awk [2]

4 Vergleich und Zusammenfassung

4.1 Vergleich mit Sed

Sed kann als der Vorläufer von Awk gesehen werden. Während Sed nur ein einfacher Stream-Editor ist, ist Awk eine ausgewachsene Programmiersprache. Awk hat viele Konzept aus Sed übernommen. Sed liesst auch Daten automatisch zeilenweise aus Dateien ein und führt anhand von Sed -Editieranwiesungen gewisse Funktionen aus. Eine Sed -Editieranweisung ähneln stark den Regel in Awk und haben die Form:

[adresse1 [, adresse2]] funktion [argumente]

Die beiden Adressen würden den Pattern in Awk entsprechen. Sie definieren die Zeilen, auf die die Funktion ausgeführt werden soll. Eine Adresse kann eine gewisse Zeile per Zeilennummer oder einen regulären Ausdruck filtern. Wird nur eine Adresse angeben, wird die Funktion nur auf jene Zeilen angewandt, auf die die Adresse passt. Werden zwei Adressen angeben, wird ein Bereich gleich wie in Awk definiert. Wird keine Adresse angegeben, wird die Funktion für alle Zeilen ausgeführt. Funktionen in Sed bestehen nur aus einen Buchstaben, was den Code sehr kryptisch und schwer lesbar macht. Sed besitzt kaum Kontrollanweisungen. Die einzigen Kontrollanweisung sind unbedingte Sprünge mit 'b' und bedingte Sprünge mit 't'. Ebenso bietet Sed keinerlei Möglichkeiten eigene Funktionen oder Variablen zu erstellt und verwendet. Sed hat nur einen Puffer, in dem Werte zwischengespeichert werden können. Auch arithmetische Operationen wie in Awk sind in Sed nicht möglich. Aufgrund der geringen Funktionalität ist Sed also nur begrenzt einsetzbar. Awk ist übersichtlicher, einfacher zu verstehen und mächtiger. Der einzige wirkliche Vorteil von Sed gegenüber Awk ist die höhere Verarbeitungsgeschwindigkeit. Sed kann daher nur begrenzt als eine Alternative von Awk angesehen werden. Eine wirkliche Alternative zu Awk sind moderne Skriptsprachen wie Perl oder Python. Diese können auch in bei komplexeren Problemen eingesetzt werden. [4] [2]

4.2 Zusammenfassung

Awk ist ein Turing-vollständige Sprache, die sich auf das Arbeiten mit spaltenorientierte Textdateien spezialisiert hat. Awk hat viele Vorteile. Awk ist

übersichtlich, hat eine C-ähnliche Syntax und Eigenschaften und ist schnell erlernbar. Weiters Awk erlaubt eine schnelle, interaktive Entwicklung.Dennoch hat auch Awk eine Reihe von Nachteilen. Vor allem die langsame Verarbeiten bei größeren Datenmengen ist zu bemängeln. Auch die Fehlersuche in awk ist wegen des fehlenden Debugger und der teils sehr kryptischen Fehlermeldung ziemlich schwierig. Awk ist trotz ständiger Weiterentwicklung nicht mehr die jüngste Sprache und Sprachen wie Perl oder Python nehmen immer mehr seinen Platz ein. Dennoch haltet sich Awk auf Grund seiner Einfachheit und leichten Erlernbarkeit und wird nicht so schnell in das Sprachenjenseits hinüber treten.

Literatur

[1] awk. Website, Juni 2010. URL `http://de.wikibooks.org/wiki/Awk`.

[2] T. Birnthaler. *Die Programmiersprache Awk*, 4 2009. URL `http://www.ostc.de/awk.pdf`.

[3] Dale Dougherty and Arnold Robbins. *sed & awk*. O'Reilly & Associates, 2 edition, 1997. ISBN 1-56592-225-5.

[4] Helmut Herold. *awk und sed*. UNIX und seine Werkzeuge. Addison-Wesley, 2 edition, 1994. ISBN 3-89319-685-4.