Sysadmin-Aufgaben mit AWK bewältigen

Reguläre Zaubereien

Tabellarisch aufgebaute Textfiles sind mit AWK bequem zu durchsuchen und zu verarbeiten. Gerade Admins schätzen an dem Sprachen-Klassiker, dass er auf jedem Unix-ähnlichen System zum Standard gehört. Dieser Artikel gibt eine Einführung. Nico Hochgeschwender

Admins müssen bei ihrer täglichen Arbeit häufig Textdateien automatisch verarbeiten, sei es um Logfiles auszuwerten, Konfigurationen zu erzeugen oder zu ändern oder um neue Accounts anzulegen. Entsprechend vielfältig sind die Werkzeuge, die Unix und Linux hierfür anbieten: Von vergleichsweise einfachen Tools wie »sed« und »ed« bis zu ausgewachsenen Programmiersprachen wie Perl, Tcl oder gar C bleibt dem Admin eine breite Auswahl.

Zu den Klassikern gehört die handliche Programmiersprache AWK. Ihre Syntax ähnelt der von C, der Einstieg wird daher jedem C-Programmierer leicht fallen. Der Name AWK leitet sich von den drei Autoren Aho, Weinberger und Kernighan ab. Die frei (im Sinne der GPL) erhältliche AWK-Version »gawk« [1] gehört zum Standardumfang jeder Linux-Distribution.

Da auch die herkömmlichen Unix-Systeme AWK enthalten, eignet es sich besonders für plattformunabhängige Skripte. Wer Solaris, HP/UX und AIX-Server neben Windows- und Linux-Rechnern zu betreuen hat, wird kaum auf AWK verzichten wollen. Als Beispiel dient im Folgenden die Rechnerliste in Listing 1. Ihre Spalten enthalten Rechnername, IP-Adresse, Betriebssystem, Software und RAM-Größe.

Arbeitsprinzip

Ein AWK-Skript verarbeitet seine Eingabedatei zeilenweise und untersucht sie nach Mustern. Findet es eine passende Stelle, führt AWK eine definierte Aktion aus. Hat der Programmierer kein Muster angegeben, führt AWK die Aktion für jede Zeile aus. Bei einfachen Einzeilern bietet sich folgende Aufrufkonvention an:

```
gawk 'AWK-Programm' Eingabedateien
```

Größere AWK-Skripte sind in einer eigenen Datei besser aufgehoben. Der Aufruf lautet dann »gawk -f *Skriptdatei Eingabefiles*«. Als Erstes soll AWK alle Rechnernamen aus der Beispieldatei (Listing 1) listen:

```
gawk '{print $1}' liste
```

Das Feld »$1« entspricht der ersten Spalte. Wer lieber die IP-Adressen wissen will, ersetzt »$1« durch »$2«. Die komplette Zeile entspricht »$0«: Der Aufruf »gawk '{print $0}' liste« gibt die komplette Datei auf dem Bildschirm aus.

Musterspiele

Alle Information zum Rechner mit Namen Goofy1 lassen sich mit Hilfe eines Musters ermitteln:

```
gawk '$1=="Goofy1" {print $0}' liste
```

AWK prüft in jeder Zeile, ob in der Spalte »$1« genau das Muster »Goofy1« auftritt. Wenn ja, gibt »{print $0}« die ganze Zeile aus. Statt des Gleichheitsoperators »= =« würde der Negationsoperator »!=« dafür sorgen, dass AWK das Kommando nur ausführt, wenn das Muster nicht passt (siehe Tabelle 1). Das

Tabelle 1: AWK-Operatoren

Operator	Erklärung	Operator	Erklärung
$	Feldoperator	>=	größer oder gleich
++ –	Postfix-Inkrement und -Dekrement	>	größer
++ –	Präfix-Inkrement und -Dekrement	~ !~	Vergleich mit regulärem Ausdruck
^	Potenzierung		
!	logische Negation	&&	logisches UND
+ -	Vorzeichen-Operationen	||	logisches ODER
* / %	Multiplikation, Division, Modulo-Operation	=	Zuweisung
		+=	Addition und Zuweisung
+ -	Addition, Subtraktion	-=	Subtraktion und Zuweisung
<	kleiner als	*=	Multiplikation und Zuweisung
<=	kleiner oder gleich	/=	Division und Zuweisung
==	gleich	%=	Modulo-Operation und Zuweisung
!=	ungleich	^=	Potenzierung und Zuweisung

Kommando »gawk '$1 != "Goofy1" {print $0}' liste« dagegen schreibt den Inhalt jener Zeilen auf die Standardausgabe, bei denen der String »Goofy1« nicht in der ersten Spalte auftritt.

Nicht nur einzelne Muster lassen sich mit AWK ermitteln, sondern auch Bereiche. Der folgende Aufruf benutzt zwei reguläre Ausdrücke (siehe Tabelle 2), die jeweils in Schrägstrichen eingeschlossen sind und von AWK mit der ganzen Zeile verglichen werden:

```
gawk '/Goofy1/,/Asterix/ ➥
  {print $0}' liste
```

Als Ausgabe erscheint der komplette Bereich vom Muster »Goofy1« bis einschließlich »Asterix1«.

Verknüpft und verbunden

Muster lassen sich mit Hilfe von booleschen Operatoren (Tabelle 1), zum Beispiel dem UND-Operator, erweitern und verknüpfen:

```
gawk '($3=="OSX") && ($4==➥
"Photoshop") {print $1}' liste
```

Die logischen Operanden sollten immer in runden Klammern stehen. Gerade bei längeren Verknüpfungen ist das Fehlen der Klammern eine potenzielle Fehlerquelle, da der Code unübersichtlicher wirkt.

Neben einer logischen UND-Verknüpfung steht auch eine logische ODER-Verknüpfung »||« zur Verfügung.

Die Ausgabe

Bisher erschien jede Ausgabe auf dem Bildschirm. Ähnlich wie eine Shell beherrscht AWK auch das Umlenken von Datenströmen in eine Datei:

```
gawk '$1=="Obelix" {print $0 > ➥
  "/home/linux/test"}' liste
```

Der Datenstrom endet in der Datei »test«. Wenn sie noch nicht existiert, legt AWK sie automatisch an. Auch das Anhängen der Ausgabe mit »> >« ist möglich. Für dieses einfache Beispiel wäre auch die Ausgabeumleitung der Shell ausreichend. Die AWK-Variante erlaubt es aber, mehrere Ausgaben in unterschiedliche Files zu schreiben und gleichzeitig Ausgaben auf den Bildschirm zu bringen.

AWK versteht auch die von C und der Shell bekannte Funktion »printf()«. Mit ihr kann der Admin die Ausgaben besser formatieren. Wie ihr C-Vorbild gibt sie keinen Zeilenumbruch aus, das muss der Programmierer ausdrücklich per »\n« fordern:

```
gawk '{printf("%x\n",$5)}' liste
```

Die Printf-Funktion soll hier einen Integerwert hexadezi-

Tabelle 2: Reguläre Ausdrücke

Ausdruck	Erklärung
.	Ersetzt irgendein Zeichen
^	Findet den folgenden regulären Ausdruck nur am Anfang einer Zeile
$	Findet den vorangehenden regulären Ausdruck nur am Ende einer Zeile
[ ]	Findet ein beliebiges der in Klammern eingeschlossenen Zeichen
[a-d1-7]	Zeichenklassen mit Bereichen: alle Buchstaben von a bis d und Ziffern von 1 bis 7
X?	Deckt entweder kein oder genau ein X ab
X*	Deckt kein oder mehrere X ab
X|Z	Deckt X oder Z ab
XZ	Deckt X unmittelbar gefolgt von Z ab

mal ausgeben (»%x«) und danach einen Zeilenumbruch durchführen (»\n«). Als Argument erhält sie den Inhalt der fünften Spalte. Weitere Infos sind unter [2] zu finden.

Ein neuer Anfang

Zum Ausgeben von Überschriften oder Meldungen eignen sich die Sprachkonstrukte »BEGIN« und »END«. AWK führt alle »BEGIN«-Kommandos vor dem Lesen der Eingabedatei und alle »END« nach der letzten Zeile aus.

```
gawk 'BEGIN {print "Suche MickeyMouse"}
$1=="MickeyMouse" {print $0}
END {print "-------"}' liste
```

Der Beispielcode ist absichtlich in mehrere Zeilen aufgeteilt: Durch die einfachen Anführungszeihen sorgt die Shell dafür, dass Gawk nur einen Befehlsstring erhält.
Neben dem Verarbeiten von Strings beherrscht AWK auch numerische Operationen. Die letzte Spalte der Datei in Listing 1 enthält Zahlen, die AWK numerisch oder nicht-numerisch bearbeiten kann. Möchte der Admin zum Beispiel wissen, wie viel Arbeitsspeicher insgesamt sich in seinem Labor befindet, gibt er folgendes Kommando ein:

```
gawk '{sum+=$5; print $5, sum}' liste
```

Das kleine Programm summiert das fünfte Feld jeder Zeile und speichert den Zwischenwert in der Variablen »sum«. Daraufhin gibt es den Feldwert und die Summe aus (siehe Abbildung 1). Alle Rechnernamen, deren Arbeitsspeicher

Listing 1: Rechnerliste

DagobertDuck	10.1.1.3	Debian	Kylix	256
Goofy1	10.1.1.4	Solaris	Mathematica	512
MickeyMouse	10.1.1.5	Debian	Apache	512
LuckyLuke1	10.1.1.6	Debian	Samba	256
LuckyLuke2	10.1.1.7	Debian	Eclipse	256
LuckyLuke3	10.1.1.8	Suse	Mupad	256
LuckyLuke4	10.1.1.9	Debian	Mupad	128
LuckyLuke43	10.1.1.10	Debian	Mupad	128
LuckyMickeyMouse	10.1.1.11	Debian	Mupad	128
Asterix1	10.1.1.12	RedHat	NetBeans	128
Asterix2	10.1.1.13	Debian	NFS	256
Obelix	10.1.1.14	RedHat	ICC	256
Apfel1	10.1.1.15	OSX	Photoshop	1024
Apfel2	10.1.1.16	OS6	Photoshop	128
Apfel3	10.1.1.17	OSX	Photoshop	512

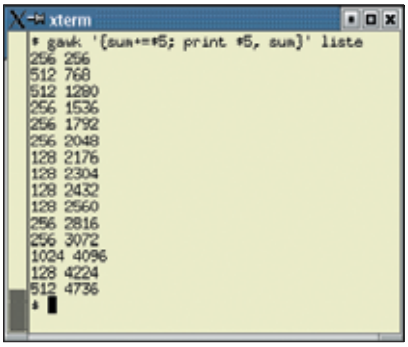

Abbildung 1: Dieser AWK-Aufruf liest die fünfte Spalte von Listing 1 und gibt für jede Zeile den Wert zusammen mit der Zwischensumme aus.

kleiner als 512 MByte ist, ermittelt folgendes Programm:

```
gawk '$5 < 512 {print $1}' liste
```

Zum Manipulieren oder Durchsuchen von Texten haben sich reguläre Ausdrücke bewährt. Durch Metazeichen lassen sich schnell komplexe Suchmuster erstellen. AWK unterstützt diese regulären Ausdrücke und hat sich damit bei Administratoren und Programmieren einen guten Ruf verdient.

```
gawk '$1 ~ /[0-9]/ {print $0}' liste
```

Das Skript sucht in der ersten Spalte »$1« nach einem Muster, in dem eine Ziffer zwischen 0 bis 9 vorkommt. Damit AWK nur »$1« durchsucht, muss man es mit der Tilde explizit dem Muster zuweisen. Die Umkehrung erreicht der Negationsoperator: »!~« sucht alle Zeilen, auf die der reguläre Ausdruck nicht zutrifft.

Reguläre Ausdrücke

Um in der Beispieldatei alle Rechner zu finden, deren Name auf »Duck« endet, genügt wieder ein kleines Kommando:

```
gawk '$1 ~ /Duck$/ {print $0}' liste
```

Der Dollar-Operator in »/Duck$/« verankert den regulären Ausdruck am Ende des Feldes »$1«. Auf »/^Lucky/« passen alle Einträge, die mit »Lucky« beginnen, etwa »LuckyLuke« oder »LuckyMickeyMouse«. Praktisch ist auch das Verknüpfen mit booleschen Operationen:

```
gawk '$1 ~/(y|M)/ {print $0}' liste
```

Dieser Aufruf durchsucht die erste Spalte, ob der String »y« oder »M« vorkommt. Eine Übersicht über die Metazeichen gibt Tabelle 2.
AWK ist reich an Funktionen, die Strings ersetzten oder vertauschen. In der Beispieldatei gibt es nur einen Rechner mit dem Betriebssystem Suse. Wenn er diesen auf Debian umstellt, sollte der Administrator auch seine Rechnerliste aktualisieren. Statt den Editor zu bemühen, bietet es sich an, dies elegant mit AWK und einer Stringfunktion zu lösen.

```
gawk '{sub(/Suse/, "Debian", $3);
  print >> "/home/linux/test"}' liste
```

Die Funktion »sub()« (substitute) erhält das Suchmuster »/Suse/«, den Ersatztext »"Debian"« und die Spalte »$3«. Trifft das Muster zu, ersetzt sie es durch den neuen Text.
Die Beschreibung der weiteren Stringfunktionen ist unter anderem in [3] und in der Manpage zu finden. Wer aufwändige AWK-Skripte schreiben will, kann auch eigene Funktionen definieren, Schleifen verwenden und mehrdimensionale Arrays nutzen. Die GNU-Variante beherrscht sogar TCP/IP-Kommunikation, sodass sich Gawk für Netzwerk-Aufgaben eignet [4].

Fazit

Zwar ist AWK nicht so mächtig wie Perl und andere Sprachen, dafür sind AWK-Skripte kurz und gut lesbar. Vorteilhaft ist auch, dass AWK-Programme noch nach Jahren laufen, selbst auf einem fremden Linux/Unix-Derivat. *(fjl)*

Infos

[1] GNU-AWK: [http://www.gnu.org/software/gawk/]
[2] Printf-Beispiele im Gawk-Manual: [http://www.gnu.org/software/gawk/manual/html_node/Printf-Examples.html]
[3] Helmut Herold, „AWK und SED“: Addison Wesley, 1991
[4] TCP/IP-Kommunikation mit Gawk: [http://www.gnu.org/software/gawk/manual/html_node/gawkinet/]

Der Autor

Nico Hochgeschwender studiert Informatik und beschäftigt sich unter anderem mit mobiler Robotik. In der Freizeit geht er am liebsten zum Wandern in die Berge oder zum Rennradfahren.