

FILTER UND REGULAR EXPRESSIONS (2.TEIL)

Markus Rajai Hammori

Markus.Hammori@wohnheim.uni-ulm.de

Inhaltsverzeichnis

1	Einleitung	1	8	Zusammenfassung Anwendungen	9
2	Erweiterte Reguläre Ausdrücke und Metacharacters	2	9	Üblich Fehler	10
2.1	Erweiterte Reguläre Ausdrücke nach POSIX Standard	2	2 A	Quellverzeichnis	11
2.2	Metacharacters	3	B	Beispieldateien	11
2.3	Zusammenfassung	3			
3	ED, die Basis für GREP, SED und AWK	3			
3.1	Kommando-Syntax (anhand von Beispielen)	3			
3.2	Zusammenfassung	3			
4	GREP	4			
4.1	Syntax, und Varianten von GREP	4			
4.2	Beispiele	4			
4.3	Zusammenfassung	4			
5	SED	4			
5.1	Grundsätzliches	4			
5.2	Allgemeine Syntax	5			
5.3	Editierbefehle	5			
5.3.1	Ersetzen	5			
5.3.2	Löschen	5			
5.3.3	Einfache Textverarbeitung	5			
5.3.4	Weiter Editierbefehle	6			
5.4	Zusammenfassung	6			
6	AWK	6			
6.1	Grundsätzliches	6			
6.1.1	Programmteile	6			
6.1.2	Strukturierung	7			
6.2	Allgemeine Syntax	7			
6.3	Muster und Prozeduren	7			
6.3.1	Muster	7			
6.3.2	Prozeduren	7			
6.4	Vordefinierte Variablen	8			
6.5	Operatoren	8			
6.6	Beispiel	8			
6.7	Zusammenfassung	9			
7	DIFF	9			
7.1	Syntax	9			

ABSTRACT

SED und AWK sind die sogenannten Powertools unter Unix/Linux. In Ihnen sind die Funktionen der meisten anderen (einfachen) Filter vereinigt. SED ist hierbei eher der Filter für den "Alltags-Gebrauch", während AWK, mit allen Merkmalen einer Programmiersprache ausgestattet, sich für aufwendigere Vorhaben empfiehlt. Dieser Artikel versucht einen Überblick über die Funktionsweisen und Anwendungsgebiete dieser Tools zu bieten. Desweiteren wird kurz auf ED, die Basis der Powertools, und GREP, einen der bekanntesten Filter eingegangen. Der letzte Filter der vorgestellt wird, und in vielen Entwicklungstools verwendet wird ist DIFF, der die Unterschiede zwischen (Text-) Dateien findet und ausgibt. Es wird auch kurz auf die in vielen Programmen verwendeten Metacharacters und die erweiterten Regulären Ausdrücke eingegangen.

1. EINLEITUNG

Filter werden in Unix-Systemen praktisch andauernd eingesetzt. Häufig ist man sich dessen gar nicht bewußt, den Programme wie ls, cat oder less werden als System-Bestandteil angesehen, und nur selten als Filter erkannt. Es ist deshalb auch nicht verwunderlich das auch in der Literatur die Definitionen von Filter variieren. So findet sich in [1] folgende Definition:

Programme bzw. Kommandos werden in UNIX auch Filter genannt: Sie lesen meist eine Folge von Zeichen ein und geben diese in irgendeiner modifizierten Form wieder aus.

Diese Definition ist ungewöhnlich, und, wenn man anderen Quellen aus dem Internet und der Literatur Glauben schenken kann, auch falsch. Programm wie copy und rm lesen keine Zeichen, und geben im Fall von rm erst recht keine

aus. Auf der anderen Seite muß die Ausgabe nicht unbedingt die modifizierten Zeichen der Eingabe sein, sondern kann auch nur Informationen über die Eingabe, wie zum Beispiel Zeilenanzahl enthalten. Es bleibt einem letzten Endes nur folgende unbefriedigend allgemeine Definition:

Filter sind Programme die einen Eingabestream in einer wohldefinierten Art in einen Ausgabestream verarbeiten, und dabei keine Veränderungen an den Eingabedaten vornehmen.

Die Beispiele aus dem ersten Teil des Referats sollten einem geholfen haben eine eigene Vorstellung vom Begriff des Filters zu entwickeln, deshalb soll an dieser Stelle auch nicht weiter darauf eingegangen werden.

Im folgenden Artikel wird versucht dem Leser einen, den Möglichkeiten entsprechenden, Einstieg in die fortgeschrittenen Filter zu bieten. Im Fall von GREP, DIFF und SED wird mit Hilfe von Beispielen auch ein Basiswissen erreicht, das es einem ermöglichen sollte mit den Programmen zu arbeiten. Im Falle von AWK kann allerdings aufgrund des beschränkten Umfangs dieser Arbeit, und der Mächtigkeit von AWK, nur ein grober Überblick über die Fähigkeiten dieser Sprache gegeben werden.

Im ersten Teil diese Referats wurden die Regulären Ausdrücke, Ihre Unterschiede zu den Shell Wildcards und die einfacheren Filter erklärt. Diese werden deshalb im Folgenden als bekannt vorausgesetzt.

Darüberhinaus gelten folgende Konventionen:

- Literaturverweise haben im Text die Form [n] wobei eine beliebige Zahl sein kann. Das Quellenverzeichnis befindet sich im Anhang A
- Manche der Beispiele haben Eingabedateien, die sich nicht in das Format dieses Artikels bringen lassen. Diese werden im Text mit < n > referenziert, wobei n für eine beliebige Zahl steht. Die Eingabedateien befinden im Anhang B.
- Als SHELL wird die bash vorausgesetzt, sie wird mit # \$ symbolisiert.
- Shelleingaben werden **bold** dargestellt
- Syntaxdefinitionen werden in *kursiv* dargestellt.
- Die Angesprochenen Filter werden groß geschrieben, z.b.: GREP. Ihr Aufruf erfolgt natürlich in Kleinbuchstaben

2. ERWEITERTE REGULÄRE AUSDRÜCKE UND METACHARACTERS

Bevor wir uns den Filtern zuwenden, müssen noch ein paar Erweiterungen an den Regulären Ausdrücken vorgestellt werden, die im ersten Teil diese Referats nicht besprochen wurden. Es handelt sich dabei um die erweiterten Regulären Ausdrücke nach dem POSIX Standard und die Metacharacters.

2.1. Erweiterte Reguläre Ausdrücke nach POSIX Standard

Nach dem POSIX Standard werden die Regulären Ausdrücke in die grundlegenden RegExp (die im ersten Teil dieses Referats behandelt wurden) und die Erweiterten RegExps aufgeteilt. Die Erweiterungen bestehen im Grunde genommen aus zwei Teilen. Zum einen wurden einige neue Ausdrücke eingeführt, um die Arbeit mit Regulären Ausdrücken zu vereinfachen:

Ausdruck	Bedeutung
+	findet ein oder ein Vorkommnis der vorangehenden RegExp
?	findet 0 oder mehr Vorkommnisse
	entweder der Vorrangehende oder der nachfolgende Ausdruck wird gefunden
()	gruppiert Reguläre Ausdrücke

Zum anderen wurden sogenannte Character Classes eingeführt:

Klasse	Zugehörige Zeichen
[:alnum:]	druckbare Zeichen (inklusive Leerzeichen)
[:alpha:]	alphabetische Zeichen
[:blank:]	Leerzeichen und Tabulatoren
[:cntrl:]	Kontrollzeichen
[:digit:]	Numerische Zeichen
[:graph:]	druckbare und sichtbare Zeichen (keine Leerzeichen)
[:lower:]	Kleinbuchstaben
[:print:]	druckbare Zeichen
[:punct:]	Satzzeichen
[:space:]	Leerzeichen
[:upper:]	Großbuchstaben
[:xdigit:]	Hexadeccimal Ziffern

Der Vorteil dieser Klassen ist, das in ihnen internationale Zeichen akzeptiert werden, also zum Beispiel [[:alpha:]] auch das é enthält, welches ja in [a-z] nicht enthalten ist.

Weitere Neuerungen sind:

- *Bindungssymbole*. Ein Bindungssymbol ist eine Zeichenfolge, die als eine Einheit behandelt werden soll. Es besteht aus den zwischen [. und .] eingeschlossenen Zeichen.
- *Äquivalenzklassen*. Eine Äquivalenzklasse listet Zeichen auf die als gleichbedeutend gelten sollen, wie zum Beispiel e und é. Die Zeichen müssen hierbei zwischen [= und =] eingeschlossen werden.

Es bleibt anzumerken, daß der POSIX Standard sich erst nach und nach durchsetzt. Er ist bisher nur in EGREP und einigen Versionen von AWK implementiert. Die Beispiele in den Folgenden Kapiteln werden deshalb mit den grundlegenden Regulären Ausdrücken gestaltet.

2.2. Metacharacters

Abschliessend bleiben noch die, relativ Programmspezifischen, Metacharacters zu erwähnen. Im folgenden werden sie, zusammen mit Ihren Gültigkeiten in den verschiedenen Programmen aufgelistet:

Symbol	ed	vi/ ex	sed/ grep	Bedeutung
\(\)	•	•	•	Muster für späteren Gebrauch speichern
\n	•	•	•	gibt das n-te gespeicherte Muster aus
\{n,m\}	•		•	findet n bis m Vorkommnisse
\	•	•	•	schützt das nachfolgende Zeichen
~		•		vorheriges Ausgabe Muster benutzen
%	•			vorheriges Ausgabe Muster benutzen

2.3. Zusammenfassung

Die Erweiterten Regulären Ausdrücke erleichtern einem die Suche nach Zeichenklassen, besonders in internationalen Zeichensätzen. Da der POSIX Standard sich noch nicht endgültig durchgesetzt hat empfiehlt es sich vor dem Gebrauch eines Filters immer die Manpage zu lesen, um festzustellen welche RegExps und Metacharacters in der aktuellen Version unterstützt werden.

3. ED, DIE BASIS FÜR GREP, SED UND AWK

Man kann den Ursprung der AWK Programmiersprache auf SED und GREP, und über diese beiden Programme zu ED dem ursprünglichen UNIX Zeileneditor zurückverfolgen.

Das Konzept des Zeileneditors ist heutzutage zwar noch bekannt, jedoch werden inzwischen hauptsächlich Full-Screen Editoren eingesetzt. ED wird hier erwähnt, obwohl er kein Filter ist. Das liegt daran, daß er die Basis für GREP, SED, AWK und viele der heute gängigen Editoren wie vi bildet. Die kurze Einführung in die Syntax ist aber keinesfalls verschwendete Zeit, da man die Kommandos fast identisch in den späteren Kapiteln wiederfindet.

3.1. Kommando-Syntax (anhand von Beispielen)

Ein Zeileneditor arbeitet, wie der Name schon sagt, immer nur auf einer Zeile. Diese Tatsache ist durchaus von Bedeutung, da Sie einem hilft die Zeilenorientierung, und die daraus folgenden Schwächen von GREP, SED und AWK zu verstehen.

Wenn ED gestartet wird, sieht man einen Prompt vor sich ähnlich dem der bash. Man befindet sich dann am Anfang der geöffneten Datei. Um die aktuelle Zeile auszugeben muss man das Kommando p (print) eingeben. Sollte man die Zeile löschen wollen so muss man d (delete) eingeben. Um zum Beispiel die 3. Zeile einer Textdatei zu löschen gibt man 3d ein. Man kann jedoch statt der direkten Angabe der Zeile auch einen regulären Ausdruck verwenden. Um zum Beispiel die nächste Zeile in der Windows 98 oder 95 vorkommt zu löschen würde man folgendes Kommando eingeben:

```
/Windows 9[58]/d
```

Will man alle Zeilen, in denen der Ausdruck vorkommt, löschen, so muss man noch den global Befehl g vorstellen:

```
g/Windows 9[58]/d
```

Eine andere Möglichkeit obigen Befehl zu schreiben wäre Windows 9[58]/Windows 9[58]/d, denn man kann anstatt einer Zeilenangabe, oder des "global" Befehls, auch einen Regulären Ausdruck voranstellen.

3.2. Zusammenfassung

- ED bildet Basis für GREP, SED und AWK
- Befehle stimmen mit den SED Befehlen überein
- ED verarbeitet immer nur die aktuelle Zeile
- Zeilenangaben oder Reguläre Ausdrücke werden zum Erweitern des Arbeitsbereichs benutzt

4. GREP

GREP ist wohl einer der bekanntesten Filter unter Linux. Er durchsucht die Eingabedatei(en) nach dem angegebenen Muster und gibt als Voreinstellung die gefundenen Zeilen aus.

GREP ist dabei eigentlich nur eine Auslagerung des oben schon angesprochenen print Befehls `p` aus dem ED Editor. Er entspricht der Kommandozeile `g/re/p` aus ED, wobei `re` für einen Regulären Ausdruck steht und `g` angibt, daß alle gematchten Zeilen ausgegeben werden sollen. Folglich ist GREP die Abkürzung für "Global Regular Expression Print".

Es bleibt zu erwähnen das GREP verständlicherweise die Zeilenorientierung von ED "geerbt" hat. Daraus resultiert eine der Schwächen des Programms, das nämlich keine Ausdrücke über Zeilengrenzen hinweg gefunden werden können. Dieses Problem ist erst mit dem SED zu lösen, oder durch aneinanderketten mehrerer Filter durch Pipes.

4.1. Syntax, und Varianten von GREP

Die Syntax von GREP ist prinzipiell sehr einfach:

```
grep [option] pattern files
```

Hierbei ist zu beachten, daß es mehrere grundlegende Varianten von GREP gibt, die pattern jeweils komplett anders interpretieren:

- `grep` (`grep -G`): Standard GREP, interpretiert pattern als Standard RegExp
- `egrep` (`grep -E`): Extended GREP, interpretiert pattern als erweiterte RegExp nach POSIX Standard (wie in 2.1 erläutert)
- `fgrep` (`grep -F`): interpretiert pattern als Liste von Ausdrücken die alle gematched werden sollen

Auf die Optionen von GREP soll hier nicht näher eingegangen werden, einige werden in den Beispielen knapp erläutert, hauptsächlich verweise ich aber auf die manpage `grep(1)`.

4.2. Beispiele

Die meisten Anwendungen von `grep` sind, wie das Programm selbst, recht übersichtlich. Daher werden im folgenden nur die Kommandozeilen und eine kurze Erläuterung aufgeführt, auf eine Beispieldatei wird verzichtet.

Die einfachste Form GREP einzusetzen ist indem man ihn die Eingabe nach einem einfachen Muster durchsuchen läßt.

Folgendes Beispiel würde ein Verzeichnis nach mp3s (Format für Audiodateien) durchsuchen:

```
$ ls | grep mp3
```

Genauso gut kann man natürlich auch reguläre Ausdrücke verwenden, wie im nachfolgenden Beispiel, in dem zusätzlich mit `-i` die Unterscheidung zwischen Groß- und Kleinschreibung ausgeschaltet wird:

```
$ grep -i "windows9[58]" datei
```

4.3. Zusammenfassung

GREP eignet sich vor allem um in einer größeren Textdatei nach etwas zu suchen. Er kann auch sehr gut im Zusammenhang mit anderen Filtern eingesetzt werden, da er auf seine Funktion spezialisiert, und deshalb darin auch schneller als SED und AWK ist. Für umfangreicher Aufgaben und Skripte für Problemlösungen ist GREP jedoch nicht geeignet, dafür empfehlen sich die später in diesem Referat beschriebenen Programme.

5. SED

SED ist ein **Stream Editor**, das bedeutet er nimmt grundlegende Textverarbeitung an einem Inputstream vor. Dies geschieht völlig nicht-interaktiv, was bedeutet das alle Befehle vorher in Form eines Skripts an SED übergeben werden müssen, und dann abgearbeitet werden. Das ist auch der Grund warum SED oft die Kehrseite der interaktiven Editoren genannt wird.

Jemanden der noch nicht mit Streameditoren gearbeitet hat mag es riskant erscheinen mehrere Befehle auszuführen ohne zu verfolgen ob alles korrekt verläuft. Das typische Merkmal der Filter, das die Eingabe nicht verändert wird kommt einem hier zur Hilfe, man kann sich nichts ruinieren. Trotzdem sollte man bei der Erstellung von SED Skripten sehr systematisch vorgehen, da sonst viel Zeit in die Fehlerbehebung fließt.

5.1. Grundsätzliches

Um von Anfang an korrekt Skripte zu schreiben muss man die folgenden Prinzipien nach denen SED arbeitet verstanden haben:

- Alle Editierbefehle werden nacheinander auf jede Zeile der Eingabe angewandt. (Also zum Beispiel bei 2 Befehlen und 2 Zeilen werden erst beide Befehle auf die erste und danach beide Befehle auf die zweite Zeile angewandt)

- Befehle werden auf alle Zeilen angewandt, außer Zeilenadressierung schränkt die betroffenen Zeilen ein.

Somit ist schon der eigentlich Unterschied zwischen ED- und SED- Befehlen formuliert. In ED wird die Zeilenadressierung benützt um die Anzahl der bearbeiteten Zelen zu erhöhen, in SED wird sie benützt um die Zeilen einzuschränken.

5.2. Allgemeine Syntax

Der Aufruf von SED erfolgt mit mit:

```
sed [-n] [-e] Befehl Datei(en) bzw.
sed [-n] -f Skriptdatei Datei(en)
```

Hierbei haben die optionalen Parameter folgende Bedeutung:

- n Nur Zeilen mit einem p Kommando oder dem p Flag des s Kommandos (nähere Erläuterung folgt) spezifizierten Zeilen werden ausgegeben.
- e Nächster Parameter wird als Kommando interpretiert (schützt in langen Skripten die Befehle)

Die Befehle haben folgendes Format:

```
[Adresse[,Adresse]][!]Befehl [argumente]
```

SED kopiert am Anfang die Einabe in den für ihn reservierten Arbeitsspeicher. Die SED Befehle bestehen wie oben angegeben aus einem Adressteil und Editierungsbefehlen. Wenn die Adresse des Befehls auf die aktuelle Zeile passt(die Adresse kann auch ein Regulärer Ausdruck sein) dann wird der Editierungsbefehl auf die Zeile angewandt. Wenn der Adressteil leer ist wird der Befehl auf die gesamte Eingabe angewandt. Wird eine Zeile von einem Befehl verändert, so wird der nächste Befehl auf die veränderte Zeile angewandt.

5.3. Editierungsbefehle

Die Editierungsbefehle bestehen aus einem Buchstaben, und werden im diesem Abschnitt mit je einem kurzen Beispiel für jede Gruppe erläutert.

5.3.1. Ersetzen

Syntax:

```
[Adresse][!]s/Muster/Ersatz/[n,g,p,w Datei]
```

Bedeutung der Optionen:

- n n-tes Vorkommen des Musters Ersetzen
- g alle Vorkommen ersetzen
- p Zeile ausgeben
- w Ausgabe in Datei schreiben

Beispiel:

In der Debian Distribution von Linux ist das automatische Update Tool apt-get enthalten, das einem automatisch die gewünschten packages (inklusive dependencies) vom Server downloaded. Mit der Option `-print-uris` kann man sich die URLs für die Dateien Ausgeben lassen und erhält dann eine Ausgabe der Form `<1>`. Mit dem folgenden Skript ist es möglich daraus die reinen URLs zu erhalten, die man dann zum Beispiel an wget weitergeben könnte.

```
sed -n "s/'\(.*\)'.*/\1/p"
```

Erläuterung:

Wenn man sich die Beispieldatei ansieht stellt man fest, daß die eigentliche URL durch ' eingeklammert wird. SED wird mit der -n Option gestartet, es werden also nur vom Kommando betroffene Zeilen ausgegeben (da das p flag gesetzt ist). Damit sind schon die ersten Zeilen der Eingabedatei ausgenommen, die keine relevanten Daten enthalten. Nun wird mit `\(.*)'` Die URL zwischen den ' gefunden und in den ersten Speicherplatz geschrieben. Anschliesend wird die Zeile durch die URL ersetzt. Das Ergebnis sieht man in `<2>`.

Ein weiteres gutes Beispiel ist folgendes Skript mit dem man E-Mail Adressen aus html Seiten herausfiltern kann:

```
sed -n 's/.*[^\.-a-z_]\([.-a-z_]*@[.-a-z_]*\).*/\1/p'
```

Ich empfehle es einfach mal zu testen, und dann zu versuchen es nachzuvollziehen

5.3.2. Löschen

Syntax:

```
/Adresse/[!]d
```

Beispiel: Man könnte obiges Beispiel auch aufteilen, und zuerst alle Zeilen die kein ' enthalten löschen :

```
sed "/^'.*/d"
```

Erläuterung: In diesem Fall kann man die Aufgabe fast wörtlich umsetzen. Die Adresse verlangt eine Zeile in der ein ' am Anfang steht (durch das ^ bedingt) gefolgt von beliebigen Zeilen. Durch das ! werden alle Zeilen gelöscht die nicht der Adresse entsprechen.

5.3.3. Einfache Textverarbeitung

In diesem Abschnitt sind die Befehle append, insert und change zusammengefasst, da diese exakt dieselbe Syntax haben.

Append fügt Text nach der mit Adresse spezifizierten Zeile ein:

```
[Adresse][!]a\
Text
```

Insert fügt Text vor der mit Adresse spezifizierten Zeile ein:

```
[Adresse][!]i\  
Text
```

Change ersetzt die mit Adresse spezifizierte Zeile durch Text:

```
[Adresse][!]c\  
Text
```

Vorsicht, nach dem Befehl kommt ein backslash, kein slash. Auch hier ist es wieder wichtig sich klarzumachen das die Kommandos die gesamte Zeile betreffen.

Beispiel:

Mit folgendem Skript würde man zum Beispiel jede Zeile in der "Microsoft" vorkommt aus gegebenem Anlaß durch "LINUX RULES" ersetzen.

```
sed -n 'c/*Microsoft.*\  
LINUX RULES'
```

5.3.4. Weiter Editierbefehle

Es gibt noch einige weitere Editierbefehle, die sich aber in Syntax und Anwendung nicht von den bisher vorgestellten unterscheiden. Für weiter Informationen empfehle ich die man page sed (1) und für eine ausführliche Beschreibung [2]. Eine sehr gute Kurzreferenz ist mit [3] bei O'Reilly erschienen.

5.4. Zusammenfassung

SED eignet sich speziell um einfache Textverarbeitung auf mehreren Dateien auszuführen, oder repetitive Aufgaben zu automatisieren. Häufig wird SED auch eingesetzt um Konverter zwischen Datenformaten zu schreiben. Im Gegensatz zum ED werden die Zeilenangaben zum Einschränken des Arbeitsbereichs verwendet.

Da die Syntax relativ einfach ist, besteht die Kunst bei der SED Programmierung hauptsächlich darin, Strukturen in den Eingabedateien zu erkennen, und dann passende Reguläre Ausdrücke zu entwickeln. Für die Schwäche von GREP, keine Ausdrücke über Zeilengrenzen hinweg zu erkennen gibt es in SED einen workaround. Man löscht einfach mit einem ersten Befehl die newlines, und sucht dann nach dem Muster. Eine Implementierung dieser Methode findet sich in [2] unter dem Namen "phrase".

6. AWK

Wie in der Einleitung bereits angesprochen ist AWK nicht ein typischer Filter, sondern eine komplette Programmiersprache. Sie wurde 1977 in den AT&T Bell-Laboratorien von den C- und UNIX-Pionieren Alfred V. Aho, Peter J. Weinberger und Brian W. Kernighan entwickelt (aus den Anfangsbuchstaben der Nachnamen setzt sich auch der Name zusammen). Ursprünglich war AWK als Programm gedacht, "[...] das die UNIX-Tools sed und grep generalisierte und gleichermaßen Text als auch Zahlen bearbeiten konnte". AWK hielt sich von Anfang an sehr stark an die C-Syntax, ließ jedoch ein paar für höhere Programmiersprachen wünschenswerte Eigenschaften fehlen, was auch nicht weiter verwunderlich ist, da es ja nicht als solche gedacht war. Nachdem AWK aber von vielen Usern sehr allgemein eingesetzt wurde, und damit sogar Interpreter und Compiler gebaut wurden entschlossen sich Aho, Weinberger und Kernighan 1985 den verbesserten NAWK (new AWK) herauszubringen. Seit dieser Version ist es beispielsweise möglich, im AWK benutzerdefinierte Funktionen zu entwickeln, dynamische reguläre Ausdrücke zu verwenden, und mit getline den Daten-Input aus mehr als einer Quelle zu erhalten.

Da dieses Referat einen eher begrenzten Rahmen hat, wird eher auf das grundlegende Programmiermodell des AWK als Filter eingegangen, als auf seine Eigenschaften als hohe Programmiersprache. Daher würde für die folgenden Beispiele auch der Umfang des ersten AWK ausreichen, es wird jedoch immer vom POSIX AWK ausgegangen, nicht von einer bestimmten Distribution. In den folgenden Kapiteln werden die wichtigsten Eigenschaften AWKs angesprochen. Ich habe es mir vorbehalten in gewissen Abschnitten nur die wichtigsten Merkmale aufzugreifen. So werden bei den Variablen nur die wichtigsten besprochen, und auf eine Beschreibung der Syntax der einzelnen Kontrollstrukturen verzichte ich komplett, wegen der Ähnlichkeit zu C. Als weiterführende Lektüre empfiehlt sich auch hier wieder [2], oder [4] das mehr auf die "Programmiersprache AWK" eingeht.

6.1. Grundsätzliches

Es ist wichtig das grundsätzliche Modell zu verstehen, daß AWK dem Programmierer bietet.

6.1.1. Programmteile

Ein AWK-Programm besteht grundsätzlich aus drei Teilen:

- **Die Haupteingabeschleife:** Dieser Teil ist der wichtigste eines AWK-Programms. Er wird zwischen { und } programmiert. Die Befehle in ihm werden nacheinander auf jeden Record (Der Record Separator RS

(siehe auch 6.4) ist per Default newline, deshalb wird im Folgenden auch von Zeilen statt Records gesprochen) ausgeführt. Die Anzahl der Zeilen der Eingabe bestimmt also die Wiederholungen der Schleife. Diese Methode erspart einem die Prozedur einen Input-Stream zu öffnen, und die Zeilen nacheinander zu lesen, wie das in den meisten höheren Programmiersprachen der Fall ist. Die Haupteingabeschleife ist von AWK vorgegeben, sie muss nicht vom Programmierer implementiert werden!

- **BEGIN**: die in der BEGIN Prozedur angegeben Befehle werden vor der Haupteingabeschleife ausgeführt. Hier werden normalerweise die Variablen initialisiert, oder Ausgaben vorgenommen die dem Benutzer das Programm erklären.
- **END**: die in der END Prozedur angegebenen Befehle werden am Ende des Programms ausgeführt.

Die END und BEGIN Funktionen sind optional, es hat sich jedoch zu einer Konvention entwickelt seine Variablen in der BEGIN Funktion zu definieren.

Folgendes Skript:

```
#!/usr/bin/awk 'BEGIN{print "Hallo" } {print "Welt"}
END{print "Tschüss"}'
```

hat also bei einer Dreizeiligen Eingabe folgende Ausgabe:

```
HalloWeltWeltWeltTschüss
```

6.1.2. Strukturierung

Eine weitere Besonderheit an AWK, die ihn zur Auswertung von Logfiles predestiniert ist seine strukturierte Dateisicht. Für AWK besteht eine Eingabedatei aus mehreren Zeilen, und jede Zeile ist in mehrere Felder aufgeteilt. Jedes Feld kann dabei mit \$Feldnummer angesprochen werden, mit \$0 wird die gesamte Zeile angesprochen. Der Feldseparator ist als Voreinstellung ein Leerzeichen(bzw. TAB), kann aber im Skript mit "FS=" gesetzt werden (laut Konvention, das erste mal in der BEGIN Funktion). Die genaue Anwendung wird in den Beispielen erkenntlich.

6.2. Allgemeine Syntax

Der Aufruf von AWK erfolgt mit:

```
awk [Optionen] 'Programm' Datei(en)
awk [Optionen] -f Skriptdatei Datei(en)
```

Die wichtigsten Optionen sind:

- F fs Feldseparator wird auf fs gesetzt
- v var=value der Variablen var wird der Wert value zugewiesen
- W Option eine der erweiterten Optionen wird gesetzt

Die erweiterten Optionen sind in der manpage von awk(1) nachzulesen.

6.3. Muster und Prozeduren

AWK Programme bestehen aus Mustern und Prozeduren:

```
Muster { Prozedur }
```

Sowohl Muster als auch Prozedur sind optional. Fehlt das Muster wird Prozedur auf die gesamte Eingabe angewendet, fehlt eine Prozedur wird die gefundene Zeile ausgegeben.

6.3.1. Muster

Es gibt folgende Möglichkeiten für ein Muster:

```
/Regulärer Ausdruck/
relationaler Ausdruck
BEGIN
END
```

- Die Regulären Ausdrücke benützen die Metacharacters, und, je nach Version die Ausdrücke nach dem POSIX Standard.
- Die relationalen Ausdrücke benützen die relationalen Operatoren, die später im Kapitel Operatoren vorgestellt werden.

6.3.2. Prozeduren

Prozeduren bestehen aus einem oder mehreren Befehlen, Funktionen oder Variablenzuweisungen, die durch Strichpunkte oder Newlines getrennt sein können, und von { und } eingeschlossen sein müssen. Die Befehle kann man in fünf Kategorien einordnen:

- Variablen- oder Arrayzuweisungen z.B.: FS = ";"
- Ausgabekommandos z.B: print "Hello World" (für fortgeschrittene Formatierung empfiehlt sich fprint)
- Eingebaute Funktionen z.B: exp(x)
- Kontrollstrukturen z.B: IF (FS==";"){print "Strichpunkt"}

Es bleibt anzumerken, daß Variablen nicht deklariert werden müssen, sie werden mit ihrer ersten Zuweisung automatisch deklariert. Es trägt jedoch zur Übersichtlichkeit bei, wenn die wichtigsten Variablen in der BEGIN Prozedur mit "" initialisiert werden. Die Arrays in AWK sind assoziative Arrays, können also auch mit Strings indiziert werden.

6.4. Vordefinierte Variablen

Variable	Beschreibung
FILENAME	Momentaner Dateiname
FS	Feldseparator (Default = Leerzeichen)
RS	Datensatzseparator(Default = Newline)
NF	Anzahl der Felder im aktuellen Datensatz
NR	Nummer des aktuellen Datensatzes
OFMT	Ausgabeformat für Zahlen
OFS	Ausgabe Feldseparator (Default = Leerzeichen)
ORS	Ausgabe Datensatzseparator (Default = Newline)
\$0	gesamter Eingabedatensatz
\$n	das n-te Feld des aktuellen Datensatzes, Felder werden durch FS getrennt
ARGC*	Anzahl der Kommandozeilenparameter
ARGV*	Array, der die Kommandozeilenparameter enthält

* ist nur in NAWK enthalten

6.5. Operatoren

Symbol	Bedeutung
= += -= *= /= %= ^= **=	Zuweisung
	Logisches ODER
&&	Logisches UND
< <= > >= != ==	Relationale Operatoren
(Leerzeichen)	Konkatenation
+ -	Addition , Subtraktion
* / %	Multiplikation, Division, Modulo
^ **	Exponentialrechnung
++ -	Inkrementierung und Dekrementierung (Post- und Prefix)

6.6. Beispiel

In diesem Beispiel wird neben den AWK Eigenschaften auch die Zusammenarbeit mehrerer Filter mit Hilfe von Pipes (im ersten teil des Referats erklärt). Es wird die Log-Datei des Apache Webservers verarbeitet, ein Beispiel für eine solche Datei ist < 3 > (normalerweise sollte die Datei unter /var/log/apache/access.log zu finden sein). Bevor wir zu

dem Skript kommen sei noch kurz angemerkt das alle diese Aufgaben auch in AWK hätten bewältigt werden können, ich es aber nützlich fand die Arbeitsweise mit Pipes einmal zu demonstrieren.

```
#!/bin/bash
awk '{ printf "%s\t%s\t%s\n", $1, $7, $9 }' $1
| sort | sed 's/\?.*/' |
awk '
BEGIN {
    IP =
    SITE =
    CODE =
}
{
    if ($1!=IP) {
        IP = $1
        SITE = $2
        CODE = $3
        printf "\n%s\n", IP
        printf "\t\t%s\t%s\n", $3, $2
    }
    else if ($2!=SITE) {
        printf "\t\t%s\t%s\n", $3, $2
        SITE = $2
        CODE = $3
    }
}
,
```

Das Ziel des Skriptes ist es, die für den Administrator eines Servers relevanten Daten in leserlicher Form darzustellen. Die gewünschte (und auch tatsächliche) Ausgabe des Skriptes sieht man in < 4 > . Es wird die zugreifende IP, und dann abgesetzt der http-Code (z.B: 404: file not found) und die abgefragte Datei ausgegeben.

Zu diesem Zweck werden mit dem ersten AWK Skript die relevanten Felder 1, 7 und 9 ausgewählt.

Daraufhin wird mit SORT nach der IP sortiert. Die Tatsache das es sich um eine alphabetische Sortierung handelt ist zu vernachlässigen, da es im folgenden nur um eine feste Reihenfolge der Einträge geht (man könnte SORT auch zu einer numerischen Ausgabe zwingen).

Im nächsten Schritt werden mit Hilfe eines SED Skriptes die ? die (zur Parameterübergabe) hinter URLs auftreten können entfernt.

Im abschliessenden AWK Programm werden zunächst einmal in der BEGIN Prozedur die Variablen initialisiert. Es existiert eine Variable für die IP, eine für die abgerufene Seite SITE und eine für den http-Code CODE. Das Programm arbeitet folgendermaßen:

- Variable IP mit dem IP-Feld der aktuellen Zeile vergleichen
- wenn nicht identisch:
 - IP, CODE und SITE werden mit den entsprechenden Feldern der Eingabe Belegt.
 - Es wird eine Newline, die IP und eine weitere Newline ausgegeben
 - CODE und SITE werden ausgegeben
- ansonsten falls SITE ungleich dem SITE-Feld der aktuellen Zeile ist:
 - CODE und SITE ausgeben
 - Code und SITE für den nächsten Vergleich mit den Feldern der aktuellen Zeile belegen

Hier zeigt sich wie leicht man mit Hilfe von AWK Systemverwaltung erleichtern kann. Die Ausgabe des Skriptes läßt sich nicht nur um einiges leichter lesen als die Originaldatei, sondern kann auch besser für Administrationstools verwendet werden.

6.7. Zusammenfassung

AWK eignet sich besonders für stark strukturierte Dateien wie Logfiles. Auf Ihnen läßt sich die Datenbankähnliche Ansicht die AWK einem von der Eingabe bietet am besten nützen. Durch seine Eigenschaften als hohe Programmiersprache läßt sich AWK auch für umfangreiche Projekte nützen, und kann sich in diesem Bereich auch durchaus mit PERL messen. Es besteht in NAWK die Möglichkeit Shell Kommandos mit in die AWK-Programme einzubinden, und ihre Rückgabe zu verarbeiten. Trotz des Umfangs der AWK-Sprache ist es, gerade für C-Programmierer, sehr einfach sich das Grundwissen anzueignen, das für den Einsatz AWKs als Filter benötigt wird.

7. DIFF

DIFF unterscheidet sich stark von den bisher in diesem Referat vorgestellten Filtern. Diese ändern (zumindest im Standardaufruf) in irgendeiner Form den Eingabestream, und werden im Normalfall auf nur eine Datei angewendet. DIFF hingegen gibt die Unterschiede zwischen den an ihn übergebenen Dateien aus. Mit Hilfe der Ausgabe und des Programms PATCH kann man dann Dateien auf den Stand des Musters bringen. Dies lohnt sich vor allem bei sehr umfangreichen Dateien, die synchronisiert werden sollen. Anstatt die komplette neue Datei zu versenden, reicht es aus die Ausgabe, DIFFs zu verschicken. Diese Eigenschaft macht sich zum Beispiel das Versionsverwaltungssystem CVS zunutze.

7.1. Syntax

Der Aufruf von DIFF erfolgt mit:

diff [Optionen] Ursprungsdatei Zieldatei

Die wichtigsten Optionen sind:

-b	ignoriert Leerzeichen
-B	ignoriert Leerzeilen
-ed	liefert ein ED-Skript als Ausgabe (nützlich wenn patch nicht vorhanden)
-i	Groß- und Kleinschreibung wird ignoriert
-I RegExp	Änderungen in Zeilen die der RegExp entsprechen werden ignoriert

Im einfachsten Fall, wenn sowohl Ursprungsdatei als auch Zieldatei wirklich Dateien sind, werden sie einfach miteinander verglichen. Ist Ursprungsdatei ein Verzeichnis, so wird die Datei im Verzeichnis verglichen die denselben Dateiname wie Zieldatei hat, und vice versa. Wenn sowohl Ursprungs- als auch Zieldatei Verzeichnisse sind, dann werden die gleichnamigen Dateien der Verzeichnisse verglichen.

Auf ein Beispiel wird in diesem Fall verzichtet, es bietet sich an, sich kurz zwei kleine Textdateien, mit übersichtlichen Unterschieden zu schreiben, um sich an das Ausgabeformat zu gewöhnen.

8. ZUSAMMENFASSUNG ANWENDUNGEN

Im folgenden wird noch einmal kurz zusammengefasst welchen Filter man für welche Aufgaben verwenden kann. Dabei spielt natürlich zum einen Erfahrung, zum anderen persönliche Vorliebe eine Rolle. Im Prinzip können GREP und SED Programme immer in AWK implementiert werden. Man sollte jedoch beachten, das die weniger umfangreichen Programme für gewisse Aufgaben erstens eine einfacher Syntax, und zweitens eine höhere Geschwindigkeit bieten.

Programm	Anwendungen
GREP	<ul style="list-style-type: none"> • Interessante Inhalte aus großen Dokumenten filtern • Parameter in Config-Dateien suchen • zählen relevanter Zeilen
DIFF	<ul style="list-style-type: none"> • Änderungen in einem Quelltext finden und mitloggen (CVS) • C wird speziell unterstützt • Änderungen in Dokumentationen finden • Ausgabe kann mit Hilfe von PATCH auf Zielformateien angewendet werden
SED	<ul style="list-style-type: none"> • Repetitive einfach Textverarbeitung auf einem oder mehreren Files • Filtern einfacher Textdateien nach bestimmten Daten • Konvertierungsprogramme zwischen Datenformaten
AWK	<ul style="list-style-type: none"> • funktioniert am besten in strukturierten Dateien (z.B.: Log-Files) • Erstellung von übersichtlichen Berichten • Umfangreiche Programme die die Zusammenarbeit mit der Shell erfordern • Komplizierte Formatierung und Filterung • Interpreter und Compilerbau... ;-)

9. ÜBLICH FEHLER

Man sollte bei der Arbeit mit Filtern folgende Punkte beachten:

- die bearbeitete Datei bleibt unverändert, um das Resultat zu speichern muss die Ausgabe in eine Datei gepiped werden
- NICHT in die Eingabedatei pipen, da sonst alle Daten verloren gehen
- Reguläre Ausdrücke nicht mit den SHELL Wildcards verwechseln

Abschliessend bleibt zu bemerken, dass bei der Arbeit mit Filtern, wie bei allen Programmiersprachen, Übung den Meister macht...also Viel Spass.

A. QUELLVERZEICHNIS

- [1] **Duden Informatik:** ein Sachlexikon für Studium und Praxis/ 2.vollständig überarbeitete und erweiterte Auflage
ISBN 3-411-05232-5
- [2] Dale Dougherty & Arnold Robbins: **SED & AWK**, O'Reilley & Associates 1997
ISBN 1-56592-225-5 3-411-05232-5
- [3] Arnold Robbins: **SED & AWK Pocket Reference** , O'Reilley & Associates 2000
ISBN 1-56592-729-X 3-411-05232-5
- [4] Reinhold Kalteis: **Awk:** Die Programmiersprache für UNIX und DOS, Kalteis. -München: Franzis 1991
ISBN 3-7723-4231-0

B. BEISPIELDATEIEN

< 1 >: Ausgabe von apt-get upgrade -print-uris:

Reading Package Lists...

Building Dependency Tree...

The following packages will be REMOVED:

mysql-base netstd wmaker wmaker-plain wsoundprefs wsoundserver xproc

The following NEW packages will be installed:

bootpc cfingerd dpsclient esound esound-common finger fping ftp gdk-implib1

gettext-base gnome-bin gnome-libs-data icmpinfo itcl3.0 libart2 libbz2

[...]

The following packages have been kept back

eterm kbd xmcputate

238 packages upgraded, 54 newly installed, 7 to remove and 3 not upgraded.

Need to get 132MB of archives. After unpacking 79.6MB will be used.

'ftp://ftp.uni-magdeburg.de/pub/mirror/linux/debian/dists/frozen/main/binary-i386/base/base-files_2.2.deb' base-files_2.2_i386.deb
28018 c099b8082e11a9c5576f3bc32d44571d

'ftp://ftp.uni-magdeburg.de/pub/mirror/linux/debian/dists/frozen/main/binary-i386/base/bash_2.03-6.deb' bash_2.03-6_i386.deb
338232 ecdf8c4cb3bfaa59a72efc1d3872b15

'ftp://ftp.uni-magdeburg.de/pub/mirror/linux/debian/dists/frozen/main/binary-i386/base/bsdutils_2.10f-5.deb' bsdutils_2.10f-5_i386.deb
30040 28b511ad2c414e677842f7f8efaf8686

'ftp://ftp.uni-magdeburg.de/pub/mirror/linux/debian/dists/frozen/main/binary-i386/base/diff_2.7-20.deb' diff_2.7-20_i386.deb
129444 ff0f68c693517117df323089d1086cd9

[...]

< 2 >: Ausgabe des ersten Beispielskripts:

ftp://ftp.uni-magdeburg.de/pub/mirror/linux/debian/dists/frozen/main/binary-i386/base/base-files_2.2.deb

ftp://ftp.uni-magdeburg.de/pub/mirror/linux/debian/dists/frozen/main/binary-i386/base/bash_2.03-6.deb

ftp://ftp.uni-magdeburg.de/pub/mirror/linux/debian/dists/frozen/main/binary-i386/base/bsdutils_2.10f-5.deb

ftp://ftp.uni-magdeburg.de/pub/mirror/linux/debian/dists/frozen/main/binary-i386/base/debianutils_1.13.3.deb

ftp://ftp.uni-magdeburg.de/pub/mirror/linux/debian/dists/frozen/main/binary-i386/base/diff_2.7-20.deb

ftp://ftp.uni-magdeburg.de/pub/mirror/linux/debian/dists/frozen/main/binary-i386/base/dpkg_1.6.12.99.deb

ftp://ftp.uni-magdeburg.de/pub/mirror/linux/debian/dists/frozen/main/binary-i386/base/e2fsprogs_1.18-3.deb

[...]

< 3 >: Access.log Datei des Apache-Webservers(Auszug):

134.60.220.112 - - [31/Oct/1999:14:43:15 +0100] "GET /icons/back.gif HTTP/1.0"304 -

134.60.220.112 - - [31/Oct/1999:14:43:16 +0100] "GET /icons/folder.gif HTTP/1.0"304 -

134.60.220.112 - - [31/Oct/1999:14:43:18 +0100] "GET /icons/blank.gif HTTP/1.0"304 -

134.60.220.112 - - [31/Oct/1999:14:43:36 +0100] "GET /doc/ HTTP/1.0" 200 56847
134.60.220.1 - - [11/Nov/1999:22:16:37 +0100] "GET / HTTP/1.0" 304 -
134.60.220.1 - - [11/Nov/1999:22:16:47 +0100] "GET /center.phtml?lang=d&enh=n&site=start HTTP/1.0" 200 5459
134.60.220.1 - - [11/Nov/1999:22:16:50 +0100] "GET /pics/oben.gif HTTP/1.0" 304 -
134.60.220.1 - - [11/Nov/1999:22:16:50 +0100] "GET /pics/start.gif HTTP/1.0" 304 -
134.60.220.1 - - [11/Nov/1999:22:16:52 +0100] "GET /pics/links.gif HTTP/1.0" 304 -

< 4 >: Ausgabe des AWK-Programms zur Auswertung der Access.log Datei (Auszug):

134.60.220.1
200 /
200 /center.phtml
200 /icons/unknown.gif
404 /klettern/index.html
404 /php
404 /php3

216.35.116.90
200 /
404 /robots.txt