

Stefan Krist

Was ist sed/awk

Funktionsweise

Regular Expressions

BRE**ERE**

sed - stream editor

Allgemeines**Basiskommandos****gsed Erweiterungen**Aho, Weinberger,
Kernighan**Allgemeines****Ausdrücke und****Anweisungen****Funktionen****Spezielle Variablen****gawk Erweiterungen**

stream editoren

Stefan Krist

H F U

5. Juni 2006

Was ist sed/awk

Regular Expressions

sed - stream editor

Aho, Weinberger, Kernighan

Was ist sed/awk

Funktionsweise

Regular Expressions

BRE

ERE

sed - stream editor

Allgemeines

Basiskommandos

gsed Erweiterungen

Aho, Weinberger,
Kernighan

Allgemeines

Ausdrücke und

Anweisungen

Funktionen

Spezielle Variablen

gawk Erweiterungen

Was ist sed/awk

Regular Expressions

sed - stream editor

Aho, Weinberger, Kernighan

Was ist sed/awk

Funktionsweise

Regular Expressions

BRE

ERE

sed - stream editor

Allgemeines

Basiskommandos

gsed Erweiterungen

Aho, Weinberger,
Kernighan

Allgemeines

Ausdrücke und

Anweisungen

Funktionen

Spezielle Variablen

gawk Erweiterungen

- ▶ Stream Editoren
- ▶ Eingabe von stdin oder file
- ▶ Ausgabe auf stdout

Definition

- Stream Editoren
- Eingabe von stdin oder file
- Ausgabe auf stdout

sed und awk sind beides Stream-Editoren, die nach dem gleichen Prinzip arbeiten. Sie bekommen die Eingabe von Standard-Input oder auch aus einer Datei und geben die manipulierte Version anschließend wieder aus.

- ▶ sed → 1973/74 von Lee E. McMahon, Bell Labs
- ▶ awk → 1977 von Aho, Weinberger und Kernighan

Schleife {

- ▶ Einlesen
- ▶ Manipulieren / Skript durcharbeiten
- ▶ Ausgeben

}

2006-06-05

stream editoren
└─ Was ist sed/awk
 └─ Funktionsweise
 └─ Funktionsweise

```
Funktionsweise  
  
Schritt 1  
└─ Editor / Skript des Editors  
└─ Ausgabe  
}
```

Beide haben eine große Schleife, die im Prinzip nur die nächste Zeile der Eingabe einliest und anhand von einem Skript manipuliert.

- ▶ Standardisiert im SUSv3 bzw. POSIX
- ▶ Jedes Unix-Derivat besitzt seinen eigenen sed

Folgende Versionen, um nur einen Überblick zu geben.

- ▶ gsed
- ▶ sed
- ▶ BSD sed
- ▶ MacOS sed
- ▶ ssed

stream editoren

└─ Was ist sed/awk

└─ Funktionsweise

└─ Verionitis - sed

Verionitis - sed

- Standardisiert im SUSv2 bzw. POSIX
- Jedes Unix-Debian besitzt seinen eigenen sed

Folgende Versionen, um zu einer Übersicht zu gelangen.

- gsed
- sed
- BSD sed
- MacOS sed
- sedit

Es gibt Zahlreiche Versionen von sed. sed ist im POSIX bzw. der aktuellen Single Unix Specification V3 standartisiert. Die meisten Linux-Derivate benutzen den gsed. Auf diesen wird auch im folgenden eingegangen.

- ▶ Standardisiert in verschiedenen Standards
- ▶ Drastische Unterschiede im Sprachumfang

Folgende Versionen, um nur einen Überblick zu geben.

- ▶ awk
- ▶ nawk - the one true awk
- ▶ gawk

2006-06-05

stream editoren

Was ist sed/awk

Funktionsweise

Verionitis - awk

Verionitis - awk

- Standarden in verschiedenen Standards
- Drastische Unterschiede im Sprachumfang

Folgende Versionen, um einen Überblick zu geben.

- awk
- naik - the one true awk
- gawk

Aber es geht auch noch schlimmer. awk hat mehr Versionen als sed, die sich in ihrem Sprachumfang auch noch mehr unterscheiden als dieser. Zudem ist awk auch in mehreren Standards definiert.

Was ist sed/awk

Regular Expressions

sed - stream editor

Aho, Weinberger, Kernighan

- ▶ BRE
- ▶ Muster
- ▶ Pattern
- ▶ Reguläre Ausdrücke

matchen auf den Textteil, den sie beschreiben.

- ▶ BRE
- ▶ Metasymbole
- ▶ Patterns
- ▶ Regulare Ausdrücke
matchen auf den Texten, den sie beschreiben.

Regular Expressions auch Pattern oder Muster genannt (im folgenden BRE) teilen sich in die im SUSv3 standardisierten Basic Regular Expressions und die Extended Regular Expressions.

- ▶ Ein Zeichen steht für sich selbst
- ▶ BRE a passt auf "a"
- ▶ Ausgenommen sind Sonderzeichen wie
., [, ^, \$, \, *

stream editoren
└─ Regular Expressions
 └─ BRE
 └─ Basismuster

Basismuster

- Die Zeichen selbst für sich selbst
- BRE passt auf "x"
- Angenommene sind Sonderzeichen
.. [, ^ , \$,
\ , *

Die wohl einfachste BRE wird das Zeichen selbst sein. Aber hier muss man Aufpassen, folgende Zeichen werden anders interpretiert, ., [, ^, \$, \, *. Die Bedeutung der Zeichen wird nachher noch erklärt.

Ein Backslash „schützt“ andere Zeichen.
So steht “\.” für Punkt.

Punkt matcht ein beliebiges Zeichen.
Ausgenommen dem newline-Zeichen.

[um Sequenzen zu beschreiben.

[abc] passt auf a, b, oder c.

- ▶ Sonderzeichen verlieren ihre Bedeutung. ^ steht, wenn es direkt nach [kommt, für die Negierung des Ausdrucks.
- ▶ Der “\” hat keine Schutzfunktion mehr.

- ▶ “-“-Zeichen dient als Trennzeichen von Bereichen.

[b-f] steht für die ASCII Reihenfolge. Das “-“-Zeichen am Anfang oder am Ende eines [Ausdrucks] steht für sich selbst.

- ▶] beendet den Klammersausdruck. Nach [steht] für sich selbst.

In Sequenzen können Zeichenklassen definiert werden.

- ▶ `:alnum`: Buchstaben oder Ziffern
- ▶ `:alpha`: Buchstaben
- ▶ `:blank`: Space, Tab, return
- ▶ `:cntrl`: Kontrollsequenzen
- ▶ `:digit`: 0-9
- ▶ `:graph`: alle druckbaren Zeichen ausser space
- ▶ `:lower`: alle Kleinbuchstaben
- ▶ `:print`: alle druckbaren Zeichen
- ▶ `:space`: Zeichen die Leerraum erzeugen, space, tab, formfeed, vertical tab, newline, carriage return
- ▶ `:upper`: Großbuchstaben
- ▶ `:xdigit`: 0-9 a-f A-F

Was ist sed/awk

Funktionsweise

Regular Expressions

BRE

ERE

sed - stream editor

Allgemeines

Basiskommandos

gsed Erweiterungen

Aho, Weinberger,
Kernighan

Allgemeines

Ausdrücke und

Anweisungen

Funktionen

Spezielle Variablen

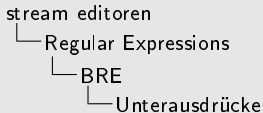
gawk Erweiterungen

Bei BREs darf nach belieben angehängt werden.

```
[ab][cd][[:digit:]]
```

passt zum Beispiel auf bc1.

- ▶ Ermöglichen einen gematchten Teil zu referenzieren.
- ▶ Sie werden mit `\(\)` eingeleitet.
- ▶ `a\(b[cd]\)e`
matcht "abce" und "abde".
- ▶ Auf `bc` oder `bd` kann referenziert werden



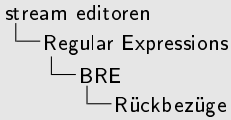
Unterausdrücke

- ▶ Ermöglichen einen gematchten Teil zu referenzieren
- ▶ Sie werden mit `|` abgegrenzt
- ▶ `match: "abc" and "def";`
 - ▶ Auf `abc` oder `def` referenziert werden

Über Unterausdrücke ist es möglich, Teile der gematchten regular expression wieder einzusetzen. Hierfür setzt man den Teil, der wieder eingesetzt werden soll in die durch `\`geschützten Klammern.

Rückbezüge referenzieren gematchte Unterausdrücke.
 $\backslash n$ ($n=\{1,2,\dots,9\}$) setzt den n-ten Unterausdruck ein.

2006-06-05



Rückbezüge

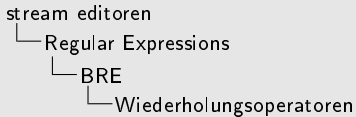
Rückbezüge n in einem gematchten Unterausdruck,
 $\backslash n$ ($n \in \{1, 2, \dots, 9\}$) setzt den oben Unterausdruck wieder ein.

Rückbezüge in der Form von $\backslash 0-9$ setzen den jeweiligen Unterausdruck wieder ein.

Es gibt die unbeschränkte und die Intervallwiederholung.

- ▶ $c[ab]^*$, matcht "c", "cab", "cabaababbbaa", etc.
- ▶ $ab\{n,m\}$
matcht "ab", "abab", zwischen n und m (inklusive)
- ▶ $\{n\}$ genau n Wiederholungen
- ▶ $\{n,\}$ mindestens n Wiederholungen
- ▶ Ist $m < n$, $n < 0$ oder 2 Wiederholungen angegeben → undefiniertes Verhalten.

2006-06-05



Wiederholungsoperatoren

Es gibt die schon bereits auf die letzte Folie gezeigte:

- `exp{1}` matcht "c", "ca", "caacaabbbba", etc.
- `exp{2,m}` matcht "c", "caab", zwischen `m` und `m` bis `2m`
- `{n}` genau `n` Wiederholungen
- `{n,}` mindestens `n` Wiederholungen
- In `m` `<n>`, `<n1 n2>` Wiederholungen angegeben → unendliches Verhalte.

Wenn eine regular expression mehrfach matchten darf, können die Wiederholungsoperatoren verwendet werden.

- ▶ `^` als erstes Zeichen steht es für den Zeilenanfang.
- ▶ `$`, als letztes Zeichen steht es für das Zeilenende.

Anker in Unterausdrücken sind ebenfalls undefiniert.

Prioritäten (BRE und ERE)

stream editoren

Stefan Krist

Was ist sed/awk

Funktionsweise

Regular Expressions

BRE

ERE

sed - stream editor

Allgemeines

Basiskommandos

gsed Erweiterungen

Aho, Weinberger,
Kernighan

Allgemeines

Ausdrücke und

Anweisungen

Funktionen

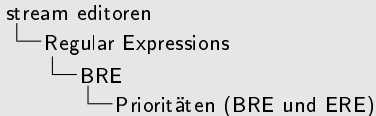
Spezielle Variablen

gawk Erweiterungen

$ab\{2,2\} = \text{“abb“}$ oder $ab\{2,2\} = \text{“abab“}$?

1. Alternative $\backslash|$ (niedrig)
2. Anker
3. Konkatenation
4. Zeichen, Wiederholungen
5. Unterausdruck, Rückbezug
6. Klammerausdruck $[]$
7. Geschützte Zeichen $\backslash.$
8. Ordnungssymbole $[::]$ (hoch)

2006-06-05



Prioritäten (BRE und ERE)

$a|b\{2,2\} = "ab"$ oder $a|b\{2,2\} = "aba"$?

1. Alternativen (|) niedrig
2. Anker
3. Konkatination
4. Zeichen, Wiederholungen
5. Unterstrich, Rückverzug
6. Klammern und ruck []
7. Geschützte Zeichen \.
8. Ordungssymbole {} | {} | {}

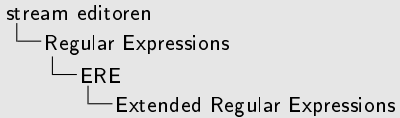
$ab\{2,2\} = "abb"$

könnte je nach Priorität auf abb oder auf abab matchen, kommt darauf an, ob die Konkatenation oder die Intervallwiederholung stärker ist.

EREs sind ebenfalls im SUSv3 spezifiziert, haben aber Erweiterungen:

- ▶ Zusätzliche Sonderzeichen `+`, `(,)`, `?`, `|`, `{, }`
- ▶ `(ab)*` anstatt `[ab]*`
- ▶ Unterausdrücke und Rückbezüge gibt es nicht.
- ▶ `$` und `^` verlieren nie ihre Bedeutung.

2006-06-05



- EREs sind eine **Wahl** im SUSv3 spezifiziert, haben aber Erweiterungen:
- Zusätzliche Sonderzeichen `^`, `|`, `?`, `[`, `{`, `}`
 - `[ab]*` anstatt `[a]*`
 - Unterstriche sind Rückzüge gibt es nicht
 - `$` und `^` verlieren ihre Bedeutung.

Auf die ERE gehen wir hier nicht so stark ein.

- ▶ Wiederholungen ohne `\`, `{n,m}`
- ▶ `a+` matcht "a", "aa",... nicht auf "" wie `a*`
- ▶ `ca?` matcht "c" und "ca".
- ▶ `c((ab)|(cd))` matcht "cab" und "ccd".

... und und und.

Überblick

Was ist sed/awk

Regular Expressions

sed - stream editor

Aho, Weinberger, Kernighan

stream editoren

Stefan Krist

Was ist sed/awk

Funktionsweise

Regular Expressions

BRE

ERE

sed - stream editor

Allgemeines

Basiskommandos

gsed Erweiterungen

Aho, Weinberger,
Kernighan

Allgemeines

Ausdrücke und

Anweisungen

Funktionen

Spezielle Variablen

gawk Erweiterungen

```
sed [-iext] [-n] script [file]
sed [-iext] [-n] -f script_file [file]
```

- ▶ -i → in-place editing
- ▶ -n → Defaultausgabe unterdrücken

Ein Skript besteht aus:

- ▶ Anweisungen
- ▶ # Kommentaren
- ▶ Leerzeilen

Eine Anweisung ist:

- ▶ Kommando; Kommando; Kommando;
- ▶ n,mKommando
- ▶ { Kommandogruppierungen }

- ▶ whitespace Kommando ist erlaubt.
- ▶ Kommandos sind 1 Zeichen lang.
- ▶ a, b, c, i, r, t, w und : sind Kommandos, die nicht mit ; verkettet werden können.

- ▶ Kommandosgruppen werden in { } gepackt.
- ▶ Der Klammer dürfen Adressen vorgestellt sein.
- ▶ whitespace } muss erstes Zeichen der Zeile sein.

- ▶ patternspace
- ▶ holdspace
- ▶ Beide sind == 8192 (standard)
- ▶ Kaum Versionen mit Beschränkung

stream editoren
└─ sed - stream editor
 └─ Allgemeines
 └─ Buffer

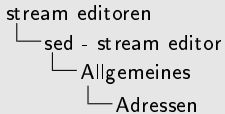
Buffer

- ▶ `patb` (pattern buffer)
- ▶ `holdb` (hold space)
- ▶ Beide sind `~ 8 KB` (je nach)
- ▶ Kaum Versionen mit Beschränkung

Beim Start wird die erste Zeile, bis zum `\n` in den Patternspace geladen. Die Operationen manipulieren den Patternspace und geben ihn, bevor die nächste Zeile geladen wird wieder aus. Der holdspace ist ein zusätzlicher Speicher, in den man den Patternspace oder andere Daten zwischenspeichern kann. Beide Speicher sind mindestens 8KB groß. Jedoch gibt es heute kaum noch eine Version die in der Größe beschränkt ist.

Adressen matchen auf Teile des Eingabe-Stream.

- 0 Befehl gilt für jeden Zyklus.
- 1 Der Befehl gilt für den, durch die Adresse beschriebenen Zyklus.
- 2 Der Befehl gilt für ab der, durch die erste Adresse beschriebenen und endet nach dem von der zweiten Adresse beschriebenen Zyklus.



Adressen

Adressen matches auf Teile des Eingabe-Streams.

- 1 Befehl gilt für jeden Zyklus.
- 2 Der Befehl gilt für den, den die Adresse beschriebene Zyklus.
- 3 Der Befehl gilt für alle, die auch die erste Adresse beschreiben und endet nach dem von der zweiten Adresse beschriebene Zyklus.

Über Adressen ist es möglich, ein Kommando nur für Bereiche und nicht für jeden Zyklus matchten zu lassen.

Eine Adresse ist:

1. Eine Zeilennummer, von 1...\$
2. Eine Kontextadresse, eine /regexp/flags[!].

2006-06-05

stream editoren
└─ sed - stream editor
 └─ Allgemeines
 └─ Adress-Aufbau

Adress-Aufbau

Eine Adresse ist:

1. Eine Zeile zusammen, von 1...5
2. Eine Kontextadresse, siehe /reg nr/Reg nr/.

Der \$ gilt bei Adressen als Variable für die letzte Zeile.

- ▶ / in Adressen, muss in [] verpackt sein
- ▶ // matcht auf die letzte im Zyklus (oder bei Substitute) gematchte BRE.
Wenn vorher auf nichts gematcht wurde, bricht sed mit einem Fehler ab.
- ▶ gsed verhält sich hier anders, // matcht auf alles.

Wenn sed:

1. mit -n gestartet wurde
 2. #n als erste Zeile im Skript hat
- gibt es keine automatische Ausgabe.

Kommandogruppierungen können 2 Adressen vorangestellt sein.

Dadurch kann man Befehle, die nur eine Adresse haben, auf 2 Adressen anwenden.

```
1,10 {  
=  
}
```

a\
Text

Mehrzeiligen Text ist erlaubt, wenn die Zeile zuvor mit einem \
abgeschlossen hat.

Der Text wird nicht nach dem Kommando ausgegeben,
erst bevor die nächste Zeile gelesen wird.
gsed erlaubt es, „a Text“ zu schreiben.

2006-06-05

stream editoren

└─ sed - stream editor

└─ Basiskommandos

└─ Append - [1]a

```
Append - [1]a
n)
Text
Merkwürdige Text ist erlaubt, wenn die Zeile zuvor mit
einem \n abgeschlossen ist.
Da! Text wird nicht nach dem Kommando ausgegeben,
es ist bevor die nächste Zeile gelesen wird.
gest erlaubt es, \n Text' zu schreiben.
```

Um zu Kennzeichnen, wieviel Adressen vor ein Kommando dürfen, wurde dies in [] vor den Befehl geschrieben.

- ▶ `b[label]`
branch springt zum angegebenen *label*. Wird *label* weggelassen springt b ans Skriptende.
- ▶ `d`
delete löscht den patternspace, c oder a wird ausgegeben und ein neuer Zyklus beginnt.
- ▶ `D`
deleteNewLine löscht den patternspace bis zum ersten newline-Zeichen.
Diese können über `substitute`, oder durch `G` und `N` erzeugt werden.

Was ist sed/awk

Funktionsweise

Regular Expressions

BRE

ERE

sed - stream editor

Allgemeines

Basiskommandos

gsed Erweiterungen

Aho, Weinberger,
Kernighan

Allgemeines

Ausdrücke und

Anweisungen

Funktionen

Spezielle Variablen

gawk Erweiterungen

c\
Text

change löscht den patternspace und gibt Text aus.

Ist Text leer wird newline ausgegeben.

Wurde a und c in einem Zyklus verwendet, wird zuerst der

Text von c und dann der von a ausgegeben.

gsed erlaubt es, „c Text“ zu schreiben.

- ▶ `g`
get kopiert den Inhalt des holdspace auf den patternspace.
- ▶ `G`
getNewline hängt an den patternspace `\n` und den holdspace an.
- ▶ `h`
hold kopiert den patternspace auf den holdspace.
- ▶ `H`
holdNewline hängt `\n` und den patternspace an den holdspace an.

Insert - [1]i

stream editoren

Stefan Krist

Was ist sed/awk

Funktionsweise

Regular Expressions

BRE

ERE

sed - stream editor

Allgemeines

Basiskommandos

gsed Erweiterungen

Aho, Weinberger,
Kernighan

Allgemeines

Ausdrücke und

Anweisungen

Funktionen

Spezielle Variablen

gawk Erweiterungen

i\
Text

Verhält sich wie append, bei i erfolgt die Ausgabe sofort erfolgt.

gsed erlaubt es, „i Text“ zu schreiben.

Label - [0]label

:label

:label definiert ein label, an das mit b oder t gesprungen werden kann.

labels sind mit allen, ausser dem Zeilenumbruch und dem \-Zeichen erlaubt.

Der gesunde Menschenverstand rät allerdings davon ab, whitespace am Begin oder am Ende eines *label* zu verwenden.

gsed hat keine Begrenzung in der *labellänge*, SUSv3 schreibt mindestens 8 vor.

stream editoren

Stefan Krist

Was ist sed/awk

Funktionsweise

Regular Expressions

BRE

ERE

sed - stream editor

Allgemeines

Basiskommandos

gsed Erweiterungen

Aho, Weinberger,
Kernighan

Allgemeines

Ausdrücke und

Anweisungen

Funktionen

Spezielle Variablen

gawk Erweiterungen

- ▶ =
linenumber gibt die aktuelle Zeilennummer und
newline aus.
- ▶ q
Quit macht die Ausgaben und beendet das Skript.
gsed kann q n verarbeiten, wobei n der exitcode ist.
- ▶ r file
Read gibt einfach die übergebene Datei aus,
unmittelbar bevor die nächste Zeile gelesen wird.
Ist file unlesbar oder ähnliches wird read dankend
ignoriert.

1

listing gibt den Patternspace aus. Die nicht druckbaren Zeichen werden wie folgt ausgegeben.

- ▶ Backslash → \\
- ▶ Bell (ASCII 7) → \a
- ▶ Formfeed (ASCII 12) → \f
- ▶ Newline (ASCII 10) → \n
- ▶ Carriage Return (ASCII 13) → \r
- ▶ Tab (ASCII 9) → \t
- ▶ Vertical Tab (ASCII 11) → \v

Was ist sed/awk

Funktionsweise

Regular Expressions

BRE

ERE

sed - stream editor

Allgemeines

Basiskommandos

gsed Erweiterungen

Aho, Weinberger,
Kernighan

Allgemeines

Ausdrücke und

Anweisungen

Funktionen

Spezielle Variablen

gawk Erweiterungen

Listing (2)

Alle anderen Zeichen werden mit `\ooo`, also Octal ausgegeben.

Zu lange Zeilen werden mit abschließendem `\` unterbrochen. Bei `gsed` kann die Länge mit `-l` beim aufruf definiert werden.

Das Ende der Zeile wird mit `$` dargestellt. `$` wird mit `\044` ausgegeben.

stream editoren

Stefan Krist

Was ist sed/awk

Funktionsweise

Regular Expressions

BRE

ERE

sed - stream editor

Allgemeines

Basiskommandos

gsed Erweiterungen

Aho, Weinberger,
Kernighan

Allgemeines

Ausdrücke und

Anweisungen

Funktionen

Spezielle Variablen

gawk Erweiterungen

n

Ausgaben werden erledigt.

Der patternspace wird mit der nächsten Eingabezeile gefüllt. Ist das Ende der Eingabe erreicht wird hier beendet.

N

NextNewLine gibt gegebenenfalls die Ausgabe von a aus.
An den patternspace wird \n und die nächste Eingabezeile
angefügt.

Endet die Eingabe, macht gsed im Skript weiter.

BSD sed beendet sich ohne den aktuellen patternspace
auszugeben.

Besonders witzig, wenn Skripte portabel sein sollten.

- ▶ `p`
Wie erwartet, dumped print einfach den patternspace.
- ▶ `P`
`printNewline` gibt den patternspace bis zum folgenden eingebettenden newline-Zeichen aus.
- ▶ `w file`
Write schreibt den patternspace in die angegebene Datei.
Der Dateiname geht bis zum Zeilenende.

Substitute - [2]s

s/regexp/replace/flags

Matcht regexp auf den patternspace, wird der Teil durch replace ersetzt.

Ersetzte Teile werden nicht mehr auf die regexp untersucht!

Besonderheiten bei replace:

- ▶ & Setzt den gesamten Teil ein, auf den die regexp gematcht hat.
- ▶ \n Setzt den n-ten Unterausdruck ein. Ist n größer als die Anzahl an Unterausdrücken gibt es einen Fehler.
- ▶ \n ersetzt regexp mit newline.

Es ist auch sAfooAbarA möglich

Substitute(2)

Flags, die für das substitute Kommando angewandt werden können:

- ▶ `n`, $n \in \{1, 2, \dots, 9\}$, Ersetzt anstatt dem ersten Vorkommen das n -te.
- ▶ `g`, Ersetzt jedes Vorkommen der regexp durch `replace`, nicht nur das erste.
- ▶ `p`, Nach der letzten Ersetzung im patternspace wird dieser ausgegeben.
- ▶ `w file`, Muss das letzte Flag sein, da der Dateiname bis zum Ende der Zeile reicht, schreibt den patternspace in `file`.

Das angeben von `g` und `1..9` führt zu undefiniertem Verhalten.

Was ist sed/awk

Funktionsweise

Regular Expressions

BRE

ERE

sed - stream editor

Allgemeines

Basiskommandos

gsed Erweiterungen

Aho, Weinberger,
Kernighan

Allgemeines

Ausdrücke und

Anweisungen

Funktionen

Spezielle Variablen

gawk Erweiterungen

- ▶ `t label`
test überprüft, ob es seit dem letzten auftreten von `t` eine erfolgreiche Ersetzung gab, wenn ja wird an das `label` gesprungen.
- ▶ `x`
`xchange` vertauscht den Inhalt des `pattern-` mit dem `holdspace`.

Yank - [2]y

```
y/str1/str2/
```

Yank vertauscht ein Zeichen aus der Liste str1 mit dem korrespondierendem Zeichen aus str2.

Ist eine Liste leer, oder sind sie verschieden lang wird mit einem Fehler beendet.

stream editoren

Stefan Krist

Was ist sed/awk

Funktionsweise

Regular Expressions

BRE

ERE

sed - stream editor

Allgemeines

Basiskommandos

gsed Erweiterungen

Aho, Weinberger,
Kernighan

Allgemeines

Ausdrücke und

Anweisungen

Funktionen

Spezielle Variablen

gawk Erweiterungen

Die Neuerungen im Überblick

stream editoren

Stefan Krist

Was ist sed/awk

Funktionsweise

Regular Expressions

BRE

ERE

sed - stream editor

Allgemeines

Basiskommandos

gsed Erweiterungen

Aho, Weinberger,
Kernighan

Allgemeines

Ausdrücke und

Anweisungen

Funktionen

Spezielle Variablen

gawk Erweiterungen

- ▶ a,c und i können Einzeiler werden.
- ▶ l n kennt n als die maximalen Zeilenbreite.
- ▶ q kann einen exitcode bekommen.
- ▶ s kann mehr flags und replace Kommandos.
- ▶ gsed interpretiert /dev/stdout, /dev/stdin oder /dev/stderr wie man es erwartet.

- ▶ `\+` steht für mindestens eine Wiederholung.
- ▶ `\?` entspricht `\{0,1\}`.
- ▶ `a\b` matcht auf a oder b.

e [cmd]

- ▶ e ohne Parameter führt den Inhalt des patternspace als Befehl aus und speichert die Ausgabe im patternspace.
- ▶ e cmd führt cmd aus und dumpst die Ausgabe auf stdout.

1 [n]

- ▶ Fill n bricht die Zeile im pattern space nach n Zeichen um. Whitespaces bleiben dabei erhalten.
- ▶ Fill ohne Parameter erwartet die Länge von der Kommandozeile mit -l n.

Ansonsten ist das Verhalten undefiniert.

QuitImmediately - [1]Q

stream editoren

Stefan Krist

Was ist sed/awk

Funktionsweise

Regular Expressions

BRE

ERE

sed - stream editor

Allgemeines

Basiskommandos

gsed Erweiterungen

Aho, Weinberger,
Kernighan

Allgemeines

Ausdrücke und

Anweisungen

Funktionen

Spezielle Variablen

gawk Erweiterungen

Q [exit]

Q beendet das Skript ohne den pattern space auszugeben.

Der exitcode ist default 0.

R [file]

R liest file Zeilenweise in den patternspace.

gsed Erweiterungen für substitute flags.

- ▶ e, Interpretiert, falls substituiert wurde, den patternspace als shell-Kommando und führt dieses aus.
- ▶ i,l, matcht nicht mehr case-sensitiv.
- ▶ m,M, erlauben es, dass ^ und \$ bei jedem Zeilenwechsel passen.

Um an die Grenzen des buffers zu kommen, gilt \ ' \ '.

gsed Erweiterungen für substitute replace.

- ▶ `\L, s/$/\LAAA/` hängt an jede Zeile aaa.
- ▶ `\l, s/$/\lAAA/` hängt an jede Zeile aAA.
- ▶ `\u, s/$/\uaaa/` hängt an jede Zeile Aaa.
- ▶ `\U, s/$/\Uaaa/` hängt an jede Zeile AAA.
- ▶ `\E, s/$/\Uaa\Ea/` hängt an jede Zeile AAa.

T [label]

Yes, you guess it!

Genau wie t, nur es springt zum label wenn nichts
substituiert wurde.

v [x.y]

Erlaubt es, gsed mit einer kleineren Version, als der installierten zu testen.

Ist die Version neuer, als die installierte beendet sich gsed mit einem Fehler.

WriteNewline - [2]W

W [file]

Speichert den patternspace bis zum ersten newline-Zeichen in file.

stream editoren

Stefan Krist

Was ist sed/awk

Funktionsweise

Regular Expressions

BRE

ERE

sed - stream editor

Allgemeines

Basiskommandos

gsed Erweiterungen

Aho, Weinberger,
Kernighan

Allgemeines

Ausdrücke und

Anweisungen

Funktionen

Spezielle Variablen

gawk Erweiterungen

Was ist sed/awk

Regular Expressions

sed - stream editor

Aho, Weinberger, Kernighan

```
awk [-F pat] [-v assign] programm [arg]
awk [-F pat] -f progfile [-v assign] [arg]
```

- ▶ -F → setzt FS auf pattern
- ▶ -v → setzt Variablen der Form FS=pat

2006-06-05

stream editoren

└─ Aho, Weinberger, Kernighan

└─ Allgemeines

└─ Kommandozeilenoption

```
Kommandozeilenoption  
  
awk -F pat1 -v asig1 program larg  
awk -F pat1 -F progfile -v asig1 larg  
  > -F → setzt FS auf pat1  
  > -v → setzt Variable der Form FS=pat
```

Auch in awk hat man das Problem, dass nicht jede Version zur anderen Kompatibel ist. Die Paramter unterscheiden sich im Namen und in ein paar Eigenschaften.

Ein Skript besteht aus:

- ▶ # Kommentaren
- ▶ Funktionsdefinitionen `function name (arg1,..argn) {}`
- ▶ Adressen/Aktionen `(adr[,adr]){Anweisung}`
- ▶ Leerzeilen

- ▶ Adresse ist optional → Anweisung gilt immer
- ▶ Anweisung ist optional → default ist print \$0

- ▶ Adressen gelten für Records oder Bereiche.
- ▶ Zwischen 0 und 2 Adressen sind erlaubt.

Als Adresse wird interpretiert:

- ▶ `/ERE/`
- ▶ Ein Ausdruck der Ausgewertet wird.
Beispielsweise: `NR==2`
Matcht auf die zweite Zeile.

awk hat ERE und folgende Erweiterungen implementiert.

- ▶ \`“` schützt das `“`
- ▶ \`a`, Bell
- ▶ \`b`, Backspace
- ▶ \`f`, Formfeed
- ▶ \`n`, Newline
- ▶ \`r`, Carriage Return
- ▶ \`t`, Tab
- ▶ \`v`, Vertical Tab
- ▶ \`/`, Slash
- ▶ \`ooo`, das `ooo`-te Zeichen des ASCII (`ooo` Oktal)

Was ist sed/awk

Funktionsweise

Regular Expressions

BRE

ERE

sed - stream editor

Allgemeines

Basiskommandos

gsed Erweiterungen

Aho, Weinberger,
Kernighan

Allgemeines

Ausdrücke und

Anweisungen

Funktionen

Spezielle Variablen

gawk Erweiterungen

Aktionen sind { Anweisungen }.

Whitespace dürfen fast überall eingebaut werden.

Ausnahme:

- ▶ Bei Funktionen.

“a=myFnct (10)“ ist verboten.

awk-Programme sind eine Folge von Anweisungen.
Anweisungen werden durch:

- ▶ ;
- ▶ }
- ▶ \n

abgeschlossen.

- ▶ Eingabestrom zerlegt in Records.
- ▶ Durch den Inhalt von RS.
- ▶ RS ist default = newline

2006-06-05

stream editoren
└─ Aho, Weinberger, Kernighan
 └─ Allgemeines
 └─ Records

Records

- `RS` bestimmt wo `RS` in Records.
- `RS` ist `\n` ist default von `RS`.
- `RS` ist default = `wurde`

Bei `awk` wird die Eingabe in einzelne Records unterteilt. Ähnlich wie bei `sed` wird auch hier per default `\n` als Zeilentrenner benutzt. Dieser Wert steht in der Variable `RS` und kann schon beim Aufruf oder zur Laufzeit verändert werden.

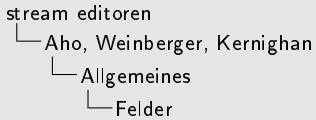
RS kann auf verschiedene Arten gesetzt werden:

- ▶ -v 'RS=%' auf der Kommandozeile
- ▶ 'RS=%' auf der Kommandozeile
- ▶ RS="b" im Programm selbst.

Steht in RS mehr als ein Zeichen ist das Verhalten undefiniert.

- ▶ Records werden in Felder unterteilt.
- ▶ Durch den Inhalt von FS.
- ▶ Default ist FS=" "
- ▶ Führende und abschließende Whitespace werden entfernt

2006-06-05



- ▶ Records werden in Felder unterteilt.
- ▶ Der Inhalt eines Feldes vor FS.
- ▶ Default FS=" "
- ▶ Filenames und absolute/relative Wikipage werden entfernt

Analog zu den Records werden auch die Felder, in die diese eingeteilt werden über den Inhalt der Variable FS bestimmt.

- ▶ FS kann ERE enthalten.
- ▶ Ist FS="" ist das Verhalten nach SUSv3 undefiniert.
- ▶ gawk macht bei FS="" jedes Zeichen = ein Feld.

Der Record und die Felder stehen in Feldvariablen:

- ▶ $\$0$, Der ganze Record
- ▶ $\$n$, Das n-te Feld
- ▶ $\$(1+3)$, Das 4-te Feld

- ▶ Zahlen
- ▶ Zeichen

“2” + 3 = 5, “2” wird implizit gecastet.

Aus diesen Datentypen lassen sich Arrays bilden.
Diese sind assoziativ (über einen String indiziert).

- ▶ Zeichen, "String"
- ▶ Ganze Zahlen
- ▶ Fließkommazahlen
- ▶ Numerische Strings → "42"
- ▶ Boolesche Werte

Können ganze oder Fließkommazahlen sein.

Unterschied beim konvertieren zu Strings.

- ▶ Ganze Zahlen werden normal ausgegeben.
- ▶ Fließkommazahlen werden über den Inhalt von Variablen formatiert. Per default wird nach “%.6g” formatiert.

gawk kann nicht nur Dezimal $256 == 0 \times 100 == 0400$

2006-06-05

stream editoren

Aho, Weinberger, Kernighan

Allgemeines

Zahlen

Zahlen

Können geschrieben: Hexadezimalzahlen sein.
Unterschied beim Konvertieren zu Strings.

- Ganze Zahlen werden normal ausgegeben.
- Hexadezimalzahlen werden über die Variable `printf` formatiert. Periodisch wird `"%x"` formatiert.

ganke la se skit sa: Dezimal 255 == 1 x 100 == 1401

Welche Variablen für die Konvertierung verantwortlich sind wird gegen Ende erklärt.

Wahr ist:

- ▶ Zahlen $\neq 0$
- ▶ Zeichen $\neq ""$

Das Ergebnis einer boolschen Berechnung ist 0 oder 1.

Arrays sind:

- ▶ Assoziativ (Zahlen werden in Strings umgewandelt).
- ▶ Eindimensional.
Man kann über eine spezielle Notation trotzdem mehrdimensionale Arrays erzeugen. Welches Zeichen dafür verwendet wird steht in SUBSEP.
- ▶ In der Lage, Zeichen und Zahlen gleichermaßen abzuspeichern.

Der Typ einer Variable hängt vom Kontext ab.

Man unterscheidet den:

- ▶ Numerischen Kontext
- ▶ String Kontext
- ▶ Booleschen Kontext

- ▶ "42Zahlen" - 19 = 23, "Zahlen" wird ignoriert.
- ▶ "Sieben" + 0 = 0.
- ▶ "drei" + "zwanzig" = 0.

Der numerische Kontext hat, einen numerischen Ergebnistyp.

Der Stringkontext tritt auf wenn:

1. Die print Anweisung ausgeführt wird.
2. Wenn die Konkatenation- oder Vergleichsanweisung ausgeführt wird.

Der Stringkontext erzwingt, ein Ergebnis als String.

Der boolsche Kontext wird verwendet wenn:

1. Muster, Adressen oder Aktionspaare auftreten.
2. if, while, do oder for vorkommen.
3. ?:, ||, &&, ! verwendet werden.

Der boolsche Kontext erzwingt, dass sein Ergebnis 0 oder 1 ist.

```
if(1 == "001") true oder false?
```

Wenn eines der Argumente ein String ist, muss zu String konvertiert werden.

```
if(1 == $1) true oder false?
```

Hier kommt auf "001" true zurück. Feldvariablen werden zuerst als numerische Strings interpretiert.

Ausdrücke werden ausgewertet und liefern je nach Ausdruck

1. Einen String
2. Einen numerischen Wert
3. Einen numerischen String
4. Ein Array

zurück.

varname

- ▶ Der einfachste Ausdruck ist eine Variable.
- ▶ Diese müssen nicht deklariert werden, schlecht bei Tippfehlern.
- ▶ Der Wert ist der Inhalt der Variablen selbst.
- ▶ Ein Skalar oder ein Array, darf nicht mehr umgewandelt werden.

stream editoren

└ Aho, Weinberger, Kernighan

└└ Ausdrücke und Anweisungen

└└└ Variablen

various

- ▶ Der einfachste Ausdruck ist eine Variable.
- ▶ Diese müssen nicht deklariert werden, sondern sind Typinferenzen.
- ▶ Der Wert für den Inhalt der Variable muss.
- ▶ Es sollte immer ein Array, da es nicht mehr angepasst werden.

Variablen sind nicht typisiert und müssen vorher nicht deklariert werden. Die einzigen Einschränkungen bestehen in den Namenskonventionen und der, dass ein Array immer ein Array und ein Skalar immer ein Skalar bleiben muss.

- ▶ `array[exp1 [, exp2, ..., expn]]`
- ▶ `fnct([exp1 , exp2, ..., expn])`

- ▶ Klammerung (15) (exp)
Wird verwendet um die vordefinierten Prioritäten zu ändern.
- ▶ Feldzugriff (14) \$n
Greift auf das n-te Feld zu.
Ist der Wert <0 wird mit einem Fehler abgebrochen.
- ▶ Increment und decrement (13) ++n, n--
- ▶ Exponentiation (12R) Basis[^] Exponent
Rechtsassoziatives Potenzieren.
gawk erlaubt auch Basis**Exponent
- ▶ Negierung (11) !exp

Prioritäten (2)

- ▶ Multiplikation und Division (10L) $exp * exp$, exp / exp
Sind Operatoren mit numerischen Ergebnistyp.
- ▶ Modulo (10L) $exp \% exp$
- ▶ Addition und Subtraktion (9L) $exp + exp$, $exp - exp$
- ▶ Konkatenation (8L) $exp \ exp$
- ▶ Vergleiche (7) $exp < exp$, $exp \leq exp$, $exp == exp$, $exp != exp$, ...
Der Ergebnistyp ist numerisch. 0 für false.

Prioritäten (3)

- ▶ Matching (6) `exp~regexp`, `exp!~regexp`
Numerischen Ergebnistyp ob das Pattern matcht.
- ▶ Arrays (5L) `exp in array`
Numerischen Ergebnistyp ob `exp` in `array` ist.
- ▶ Logisches und (4L) `exp && exp`
- ▶ Logisches oder (3L) `exp || exp`
Numerischen Ergebnistyp.
- ▶ Ausdrucksconditional (2R) `?exp1:exp2`
Ist `exp1` true wird `exp2` ausgeführt.
- ▶ Zuweisungen (1R) `exp=exp`, `exp+=exp`,
`exp**exp`, ...

Namen von Variablen oder eigenen Funktionen dürfen:

- ▶ nur mit einem Buchstaben oder `_` beginnen
- ▶ Buchstaben, `_` und 0-9 enthalten
- ▶ Nicht mit Schlüsselwörtern oder vordefinierten Funktionen kollidieren.

Die Sprungbefehle

- ▶ break
- ▶ continue

sind innerhalb von Schleifen möglich.

```
delete array[exp]
```

Löscht ein Element aus dem Array.

gawk kann delete array um das Array ganz zu löschen.

```
for([stmt];[exp];[sstmt2]) stmt
```

```
for(var in array) stmt
```

Die zweite Form ist als Iterator über ein Array vorgesehen.

- ▶ `if(exp) stmt1 [else stmt2]`
- ▶ `while(exp) stmt`
- ▶ `do stmt while(exp)`
- ▶ `exit [exp]`
exp wird ausgewertet ist der exitcode

1. `print exp,exp,exp`
2. `print exp,exp,exp> exp`
3. `print exp,exp,exp » exp`
4. `print exp,exp,exp| exp`

Verhalten sich wie man es gewohnt ist.

2006-06-05

stream editoren

└ Aho, Weinberger, Kernighan

└┬ Ausdrücke und Anweisungen

└┬ Print

```
Print
1. print exp,exp,exp
2. print exp,exp,exp > exp
3. print exp,exp,exp > *exp
4. print exp,exp,exp | *exp
Verhalten des wie man es gemacht ist.
```

Hier verhält sich print wie man es erwartet. Das einfache redirect schreibt in eine Datei, das doppelte hängt es an das Ende an. Die Pipe gibt die Ausgabe von print an ein anderes Programm weiter.

Wie print, mit den aus C bekannten Formatierungen.

`%flags[breite][.praez]form`

Flags:

- ▶ -, Ausgabe ist linksbündig
- ▶ +, Ausgabe mit Vorzeichen
- ▶ #n, $n = \{o, x, X, e, E, f, F, g, G\}$

Breite ist die minimale Breite der Ausgabezeichen.

praez gibt die Nachkommastellen an.

- ▶ **Return [exp]**
Nur Innerhalb von Funktionen. Gibt exp oder nichts zurück.
Erwartet der Funktionsaufruf einen Wert gibt es einen Fehler.
- ▶ **next** Ließt den nächsten Rekord ein und verwirft den alten.

```
function name(arg1,..., argn) { stmt1; stmt2 }
```

- ▶ Weniger Argumente als definiert → uninitialisierte Werte
- ▶ Mehr Argumente → Fehler.
- ▶ Arrays by reference
- ▶ Skalare by value

- ▶ $\text{atan2}(y,x) \rightarrow$ Arcus Tangenz von y/x
- ▶ $\cos(x)$
- ▶ $\exp(x) \rightarrow e^x$
- ▶ $\text{int}(x) \rightarrow$ abrunden
- ▶ $\log(x) \rightarrow \log_n(x)$

Arithmetische Funktionen (2)

- ▶ `rand()` → eine Zahl im Bereich von 0-1
- ▶ `sin(x)`
- ▶ `sqrt(x)`
- ▶ `srand([x])` → initialisiert den Zufallsgenerator neu

- ▶ `gsub(e,r[,i])` → ersetzt Muster `e` in `i` (oder `$0`) durch `r`
- ▶ `index(s,t)` → findet erstes Vorkommen von `t` in `s`
- ▶ `length([s])` → Länge von `s` oder `$0`
- ▶ `match(s,e)` → erste Position von `s` an der `e` matcht
- ▶ `split(s,a[,f])` → Teilt String `s` in `a[1]`, `a[2]`, ..., `f` ist der Feltrenner

Was ist sed/awk

Funktionsweise

Regular Expressions

BRE

ERE

sed - stream editor

Allgemeines

Basiskommandos

gsed Erweiterungen

Aho, Weinberger,
Kernighan

Allgemeines

Ausdrücke und

Anweisungen

Funktionen

Spezielle Variablen

gawk Erweiterungen

String Funktionen (2)

- ▶ `sprintf(f,e,...)` → Ausgabe der Argumente anhand von Format `f`
- ▶ `sub(e,r[,i])` → Ersetzt Muster `e` durch `r` in `i` (oder `$0`)
- ▶ `substr(s,m[,n])` → Gibt `s` ab Position `m` bis `n` (oder ende) zurück
- ▶ `tolower(s)`
- ▶ `toupper(s)`

stream editoren

Stefan Krist

Was ist sed/awk

Funktionsweise

Regular Expressions

BRE

ERE

sed - stream editor

Allgemeines

Basiskommandos

gsed Erweiterungen

Aho, Weinberger,
Kernighan

Allgemeines

Ausdrücke und

Anweisungen

Funktionen

Spezielle Variablen

gawk Erweiterungen

- ▶ `close(f)` → schließt die pipe oder das file `f`.
- ▶ `e | getline [v]` → Liest aus der pipe `e` in `v` oder `$0`
- ▶ `e |& getline [v]` → Liest aus einem Co-Prozess `e` in `v` oder `$0`
- ▶ `getline [v] < e` → Liest einen Record aus Datei `e` in `v` oder `$0`
- ▶ `system(e)` → Führt `e` in der Shell aus

Was ist sed/awk

Funktionsweise

Regular Expressions

BRE

ERE

sed - stream editor

Allgemeines

Basiskommandos

gsed Erweiterungen

Aho, Weinberger,
Kernighan

Allgemeines

Ausdrücke und

Anweisungen

Funktionen

Spezielle Variablen

gawk Erweiterungen

BEGIN, Vor dem ersten Befehl

- ▶ Der Reihenfolge im Skript nach

END, Nach dem letzten Befehl

- ▶ Wenn keine Eingabe mehr vorhanden ist.

2006-06-05

stream editoren

└ Aho, Weinberger, Kernighan

└ Funktionen

└ BEGIN & END

BEGIN & END

BEGIN, Vor dem ersten Befehl

- Die Reihenfolge im Skript nicht

END, Nach dem letzten Befehl

- Wenn keine Eingabe mehr vorhanden ist.

Zuerst werden sämtliche BEGIN-Blöcke des Vorkommens nach abgearbeitet. Dann werden die restlichen Blöcke abgearbeitet. Ist keine Eingabe mehr vorhanden wird jeder END-Block der Reihenfolge nach behandelt.

- ▶ Argumente an awk stehen in ARGV
- ▶ Die Anzahl in ARGC
- ▶ Konflikt mit awk - Optionen → - -

- ▶ CONVFMt → Umwandlung von Zahlen zu Strings
- ▶ ENVIRON["PATH"] → Enthält die Umgebungsvariablen
- ▶ ERRNO → Returnstring von getline oder close
- ▶ FILENAME → Der gerade gelesenen Datei

Weitere Variablen (2)

- ▶ FNR → Recordnummer relativ zur Eingabedatei.
- ▶ FS → Fieldseparatorregexp.
- ▶ IGNORECASE → Muss true ergeben.
- ▶ NF → Anzahl Felder im Record
- ▶ NR → Anzahl gesamter gelesener Records.

stream editoren

Stefan Krist

Was ist sed/awk

Funktionsweise

Regular Expressions

BRE

ERE

sed - stream editor

Allgemeines

Basiskommandos

gsed Erweiterungen

Aho, Weinberger,
Kernighan

Allgemeines

Ausdrücke und

Anweisungen

Funktionen

Spezielle Variablen

gawk Erweiterungen

- ▶ Ist IGNORECASE = true wird bei FS="[c]" wird c und C benutzt.
- ▶ Hat FS einen einstelligen Wert FS="c" ist IGNORECASE ungültig.

2006-06-05

stream editoren

└ Aho, Weinberger, Kernighan

└ Spezielle Variablen

└ Designfehler

Designfehler

- Ist IGNORECASE = true und bei PS="[" nicht e und C gesetzt.
- Hat PS einen einstelligen Wert PS="[" ist IGNORECASE unglg.

Zumindest ist es meiner Meinung nach ein Designfehler, da mir kein Anweisungsfall einfällt, wo dieses Verhalten sinnvoll sein könnte.

Weitere Variablen (3)

- ▶ OFMT → Umwandlungen von Zahlen zu Strings bei print.
- ▶ OFS → Das Trennzeichen zwischen 2 print-Argumenten (“ “)
- ▶ RS → Record Seperator (Ein Zeichen), gawk regexp
- ▶ SUBSEP → Wert der bei Pseudomehrdimensionalenarrays eingefügt wird.

gawk kann:

- ▶ /dev/stderr
- ▶ /dev/stdout
- ▶ /dev/stdin

vernünftig interpretieren.

Was ist sed/awk

Funktionsweise

Regular Expressions

BRE

ERE

sed - stream editor

Allgemeines

Basiskommandos

gsed Erweiterungen

Aho, Weinberger,
Kernighan

Allgemeines

Ausdrücke und

Anweisungen

Funktionen

Spezielle Variablen

gawk Erweiterungen

nextfile

Ähnlich zu next, gawk liest hier den ersten Record der nächsten Datei.

Über Co-Prozesse ist es möglich, eine bidirektionale Pipe aufzubauen.

Co-Prozesse können nicht nur Shell-Kommandos sondern auch Netzwerkverbindungen nutzen.

- ▶ $\text{and}(v,w)$ → Liefert bitweises und von v & w
- ▶ $\text{compl}(v)$ → das bitweise Komplement von v
- ▶ $\text{lshift}(v,c)$ → Das Ergebnis des lshift von v um c
- ▶ $\text{or}(v,w)$ → Liefert bitweises oder von v & w
- ▶ $\text{rshift}(v,c)$ → Das Ergebnis des rshift von v um c
- ▶ $\text{xor}(v,w)$ → Liefert bitweises XOR von v & w

- ▶ `asort(s [,d])` → sortiert array `s` in sich oder in `d`
- ▶ `gensub(e,r,w[,i])` → ersetzt Muster `e` in `i` (oder `$0`) durch `r`, `w` ist die Art der Ersetzung
- ▶ `strtonum(s)` → Hex, Oct zu Zahl

- ▶ `fflush([f])` → flusht f oder alle buffer
- ▶ `mktime(datum)` → Wandelt ein Datum in Sekunden seit 01.01.1970
- ▶ `strftime([f [,t]])` → Wandelt timestamp in durch format definierten String
- ▶ `systeme()` → Systemzeit als timestamp

Was ist sed/awk

Funktionsweise

Regular Expressions

BRE

ERE

sed - stream editor

Allgemeines

Basiskommandos

gsed Erweiterungen

Aho, Weinberger,
Kernighan

Allgemeines

Ausdrücke und

Anweisungen

Funktionen

Spezielle Variablen

gawk Erweiterungen

- ▶ [sed & awk GE-PACKT\[.de\]](#)
- ▶ <http://sed.sourceforge.net/>
- ▶ <http://www.faqs.org/faqs/computer-lang/awk/faq/>
- ▶ <http://www.uga.edu/~ucns/wsg/unix/awk/>
- ▶ [comp.lang.awk](#)

Was ist sed/awk

Funktionsweise

Regular Expressions

BRE**ERE**

sed - stream editor

Allgemeines**Basiskommandos****gsed Erweiterungen**Aho, Weinberger,
Kernighan**Allgemeines****Ausdrücke und****Anweisungen****Funktionen****Spezielle Variablen****gawk Erweiterungen**

Danke für die Aufmerksamkeit.

<stefan@krist.cn>

7209 0A38 07A6 A387 7EB2

8A52 F44C D7D7 1DF7 AB6D