

Textbearbeitung

Martin Raden

Einführung von

- `awk` in Zeilenweise Daten verarbeiten
- `vim`, `nano` in Editoren in der Shell

[Video: Bashino - #09 Spalten verschieben und filtern mit awk und tr \[6 min\]](#)

[Video: Bashinho - #08 Texteditoren in der Bash \[9 min - letzten 2 irrelevant\]](#)

Zeilenweise Daten verarbeiten

- `awk` = “Programmiersprache” zur (**zeilenweisen**) Datenverarbeitung
 - Programme entweder direkt im Aufruf (`echo "A B" | awk '{print $2}'`) oder in Datei definiert (`awk -f programFile`)
 - * Programme im Direktaufufruf immer mit single tick ' quoten “-F” = field separator zur spaltenweisen Datentrennung pro Zeile (setzt “FS” und “OFS” Variable, siehe unten)
 - Variablen
 - * Spaltenelemente mittel `SPALTENNUMMER` ansprechbar; `$0` = ganze Zeile
 - * lokale Variablen via Zuweisung mit “=” angelegt (`echo 4 | awk '{i=$1+2; print i}'`)
 - * Spaltenelemente/-variablen können auch überschrieben werden!!
 - `echo A B | awk '{$2="D"; print $0}'`
 - `$0` = Zusammensetzung der aktuellen Spaltenelemente/Felder mit aktuellem (output) field separator (`OFS`) (kann auch geändert werden)
 - * `NF` = Anzahl Spalten (`$NF` = letzte Spalte) (number of fields), z.B. letzte Spalte ausgeben (`echo A B | awk '{print $NF}'`) oder Anzahl Spalten ausgeben (`echo A B | awk '{print NF}'`)
 - * `NR` = aktuelle Zeilennummer (number of rows), z.B. zeilennummerierte Ausgabe (`ls -l | awk '{print "Zeile "NR : "$0}'`)
 - * `FS` = verwendetes Spaltentrennzeichen (field separator)
 - entweder via “-F” setzen (`echo "A;B" | awk -F ";" '{print $1}'`) (setzt auch `OFS`)
 - oder explizit für alle Zeilen vorm Einlesen (`echo "A;B" | awk 'BEGIN{FS=";"}{print $1}'`)
 - * `OFS` = aktuelles Spaltentrennzeichen der Ausgabe (output field separator); muss (für alle Zeilen) im “BEGIN”-block (siehe unten) gesetzt werden (`echo "A B" | awk 'BEGIN{OFS=":"}{print $1,"lala"}`)
 - “;” = Befehlstrennzeichen (Beispiel weiter unten)
 - `print` = Ausgabe mit Zeilenumbruch (`echo "Hans Peter" | awk '{print "Name="$2"\nVorname="$1}'`)
 - * Ausgaben mit **double tick** " quoten
 - * **Variablen nicht in quotes** einschliessen (stehen vor/zwischen/hinter den gequoteten Strings, siehe obiges Beispiel oder hier nachfolgend)
 - `printf` = Ausgabe OHNE Zeilenumbruch (`echo "17" | awk '{printf "input="$0; printf "next"; print "linebreak"}`)

- Kontrollstrukturen
 - * `if(NR%2 == 0) {print "gerade"} else {print $0} = if conditional`
 - * `for(i=0; i<5; i++) { printf " i="i; } = for looping`
 - Vor- und Nachbereitung der Ausgabe (z.B. für Kopf-/Fusszeilen oder zum initialisieren von Variablen) mittels “BEGIN{ }” und “END{ }”-Block (jeweils optional), z.B. `echo "Doe,John" | awk -F " ," 'BEGIN{print "Vorname Nachname"}{print $2 " "$1}END{print "#Ende"}'`
-

> Tutorials <

Folgende kompakte Online awk Tutorials sind ein guter Einstieg

- [Getting started with awk](#) von riptutorial.com
 - [Extract specific column/field from specific line](#) von riptutorial.com
-

Editoren in der Shell

Warum bzw. wann muss man sich mit Kommandozeileneditoren herumschlagen?

- weil man ein Nerd ist und es kann. . .
- weil das Betriebssystem keine graphische Benutzeroberfläche hat (z.B. high performance Rechencluster etc.)
- weil die (Inter)netzverbindung schlecht ist
- weil man keine Maus braucht und ggf. damit “mal schnell(er)” was ändern kann (man ist ja schon vor Ort und muss sich nicht erst im “Datei öffnen”-Menü hinwühlen)
- ...

vi(m)

- **vi** oder **vim** = Kommandozeileneditor
 - **ESC**-Taste = Wechsel in den “Kommandomodus”
 - im Kommandomodus
 - * Taste “**i**” oder “**a**” = Wechsel in den “Einfügemodus” (insert/append), d.h. Eingaben werden an der Cursorposition eingefügt
 - * Taste “**R**” = Wechsel in den “Überschreibmodus” (replace), d.h. Eingaben überschreiben die aktuelle Cursorposition
 - * “:”-Präfix leitet Kommandos ein, z.B.
 - “:w” = speichert die aktuelle Datei (write)
 - “:w myFile.txt” = speichert den Inhalt in Datei “myFile.txt”
 - “:w!” = überschreiben der Datei auch wenn geschützt etc.
 - “:q” = **beendet vim** (quit); nicht möglich bei ungespeicherten Änderungen
 - “:q!” = wirklich beenden und ungespeicherte Änderungen gehen verloren
 - “:wq” = speichern und beenden (in einem Rutsch)
 - * “d”+ENTER = löscht aktuelle Zeile
 - * “d2”+ENTER = löscht aktuelle plus 2 Folgezeilen (also 3!)

Wer sich an vim versuchen will, sollte das schon direkt in Linux vorhandene “vim tutorial” bearbeiten, in dem man einfach **vimtutor** in der Konsole eingibt/aufruft!

nano

nano ist ein alternativer Editor zu **vim**, welcher einen anderen Befehlssatz bzw. eine andere Nutzerführung verwendet.

Um mit **nano** vertraut zu werden, sollte man einen ersten Blick auf dessen **eingebaute Hilfe** werfen. Dazu

- **nano** aufrufen
 - mit “STRG+G” die Hilfe aufrufen
 - mit “STRG+X” kann man diese und anschliessen das Programm verlassen
-