Vorlesung Linux-Praktikum

4. Mehr zu Daten manipulieren

Dirk Frettlöh

Technische Fakultät
Universität Bielefeld

Zusammenfassung heute

`wc`	zählt Zeichen, Worte, Zeilen
`$(...)`	Ausgabe des Befehls benutzen
	Variablen in der Shell
`grep`	nach Zeichenkette in Datei suchen
`sed`	Suchen und Ersetzen von Zeichenketten
`join`	Zusammenfügen von “Tabellen”
`csv`	zeitloses Tabellenformat
`cut`	einzelne Einträge aus Zeilen auswählen
`tr`	stutze Zeichenketten zurecht

...sowie Vorstellung des Programmierprojekts.

Shellskripte

Erinnerung: Shellskripte mit Parametern

Beispiel zur Übergabe von Parametern:

```
#!/bin/bash

echo "Erstes : $1"
echo "Zweites: $2"
echo "Drittes: $3"

echo "Anzahl : $#"
echo "Alle   : $*"
```

Zum Beispiel

```
$ ./skript.sh eins zwei vier
```

Shellskripte

Erinnerung: Shellskripte mit Parametern

Beispiel zur Übergabe von Parametern:

```
#!/bin/bash

echo "Erstes : $1"
echo "Zweites: $2"
echo "Drittes: $3"

echo "Anzahl : $#"
echo "Alle   : $*"
```

Zum Beispiel

```
$ ./skript.sh eins zwei vier
```

```
Erstes : eins
Zweites: zwei
Drittes: vier

Anzahl : 3
Alle   : eins zwei vier
```

Shellskripte

Erinnerung: Planeten sortieren

```
#!/bin/bash
head -2 $1 ; tail -n +3 $1 | sort -k $2 -n
```

```
$ ./hsort2.sh planeten2.txt 2
Planet Durchmesser Temperatur

Merkur      4878          440
Mars        6794          210
Venus ...
```

word count

`wc`: *word count*, zählt Zeichen, Worte, Zeilen.

```
$ echo "eins zwei drei" | wc
      1       3      15
```

word count

`wc`: *word count*, zählt Zeichen, Worte, Zeilen.

```
$ echo "eins zwei drei" | wc
      1       3      15
```

- `wc -m`: Zählt nur die Zeichen
- `wc -w`: Zählt nur die Worte
- `wc -l`: Zählt nur die Zeilen

```
$ echo "Linux" | wc -m
```

word count

`wc`: *word count*, zählt Zeichen, Worte, Zeilen.

```
$ echo "eins zwei drei" | wc
      1       3      15
```

- `wc -m`: Zählt nur die Zeichen
- `wc -w`: Zählt nur die Worte
- `wc -l`: Zählt nur die Zeilen

```
$ echo "Linux" | wc -m
6
$ echo -n "Linux" | wc -m
```

word count

`wc`: *word count*, zählt Zeichen, Worte, Zeilen.

```
$ echo "eins zwei drei" | wc
      1       3      15
```

- `wc -m`: Zählt nur die Zeichen
- `wc -w`: Zählt nur die Worte
- `wc -l`: Zählt nur die Zeilen

```
$ echo "Linux" | wc -m
6
$ echo -n "Linux" | wc -m
5
```

(Erinnerung: das `-n` unterdrückt den Zeilenvorschub)

Ein subtiler Punkt

Parameter in Eingaben umwandeln

Ein wichtiger Unterschied: Parameter oder Eingabe(-datei).

```
wc -m hallo
```

Zählt in der Datei `hallo` (falls es die gibt).

```
echo hallo | wc -m
```

Zählt Zeichen im Wort “hallo”.

Ein subtiler Punkt

Parameter in Eingaben umwandeln

Ein wichtiger Unterschied: Parameter oder Eingabe(-datei).

```
wc -m hallo
```

Zählt in der Datei `hallo` (falls es die gibt).

```
echo hallo | wc -m
```

Zählt Zeichen im Wort “hallo”.

Wichtig: Die Pipe — löst dieses Problem.

Ein nützliches Konstruktionselement

Ergebnisse von Programmaufrufen ausgeben

`$(...Aufruf ...)`: liefert Ausgabe des Aufrufs

Beispiel (`date` gibt Datum aus):

```
$ date "+%d. %B %Y"
11. November 2020

$ echo "Log vom $(date "+%d.  %B %Y") für $USER:"
Log vom 11.  November 2020 für frettloe:
```

Ein nützliches Konstruktionselement

Ergebnisse von Programmaufrufen ausgeben

`$(...Aufruf ...)`: liefert Ausgabe des Aufrufs

Beispiel (`date` gibt Datum aus):

```
$ date "+%d. %B %Y"
11. November 2020
```

```
$ echo "Log vom $(date "+%d.  %B %Y") für $USER:"
Log vom 11.  November 2020 für frettloe:
```

Es geht beliebig komplex (mit Pipes):

```
$ echo "Die Sonne hat $(tail -n +3 planeten2.txt | wc -l)
Planeten."
Die Sonne hat 8 Planeten.
```

Variablen

Variablen

Variablenzuweisungen

Wert an Variablen zuweisen: (ohne Leerzeichen!)

```
$ wort=eins
```

Variablen

Variablenzuweisungen

Wert an Variablen zuweisen: (ohne Leerzeichen!)

```
$ wort=eins
```

Variablenwert benutzen / ausgeben:

```
$ echo $wort
eins
```

Variablen

Variablenzuweisungen

Wert an Variablen zuweisen: (ohne Leerzeichen!)

```
$ wort=eins
```

Variablenwert benutzen / ausgeben:

```
$ echo $wort
eins
```

Variablen sind "schwach getypt"

- werden automatisch als Zeichenkette oder Zahl benutzt. Zum Beispiel:
- `a=7`: Zahl,
- `b=sieben`: Zeichenkette,
- `c="17"`:

Variablen

Variablenzuweisungen

Wert an Variablen zuweisen: (ohne Leerzeichen!)

```
$ wort=eins
```

Variablenwert benutzen / ausgeben:

```
$ echo $wort
eins
```

Variablen sind "schwach getypt"

- werden automatisch als Zeichenkette oder Zahl benutzt. Zum Beispiel:
- `a=7`: Zahl,
- `b=sieben`: Zeichenkette,
- `c="17"`: Zeichenkette.

Variablen

Variablenzuweisungen aus Shell-Aufrufen

Zwischenspeichern von Programmausgaben:

```
$ a=$(echo -n Linux | wc -m)
$ echo $a
5
```

Variablen

Variablenzuweisungen aus Shell-Aufrufen

Zwischenspeichern von Programmausgaben:

```
$ a=$(echo -n Linux | wc -m)
$ echo $a
5
```

Auch eine komplette Zeile kann man sinnvoll speichern:

```
$ a=$(ls -l eins.txt)
$ echo $a
-rw-r--r-- 1 cg stud 4502 17. Nov 16:38 eins.txt
```

- Mehrzeilige Ausgaben besser nicht in Variablen packen!

Variablen

Variablen als Zeichenketten verarbeiten

```
$ name=datei
$ verz=/home/juser
$ pfad=$verz/$name.jpg
$ echo $pfad
/home/juser/datei.jpg
```

Variablen

Variablen als Zeichenketten: Sonderfälle

Variablennamen durch Klammern vom Text abtrennen:

```
$ name=zeichen
$ echo ${name}kette
zeichenkette
```

Leerzeichen durch Anführungszeichen ("...") erhalten:

```
$ a=eins
$ b=zwei
$ c="$a $b"
$ echo $c
eins zwei
```

Kommandos zum Bearbeiten des Inhalts von Textdateien

Mehr dazu.

Kommandos zum Bearbeiten von Textdateien

Mehr zu grep

grep (global regular expression print)

```
$ grep Mars planeten.txt
planeten.txt:  Mars 6.749 210
```

Durchsucht die Datei `planeten.txt`,
ob sie den Text “Mars” enthält.

Man kann mit Wildcards natürlich mehrere Dateien durchsuchen:

```
$ grep Mars *.txt
planeten.txt:  Mars 6.749 210
planeten2.txt:  Mars 6.749 210
```

Kommandos zum Bearbeiten von Textdateien

Texte in Dateien suchen: grep

```
$ grep mars planeten.txt
```

findet keinen Treffer: `mars` $\neq$ `Mars`

Falls Groß-/Kleinschreibung (Datei/datei) egal sein soll:

```
grep -i mars planeten.txt
planeten.txt:4 Mars 6.749 210
...
```

Kommandos zum Bearbeiten von Textdateien

Ausgaben mit grep filtern

Filtern von Programmausgaben mit grep:

```
ls -la | grep 2023
```

- zeigt alle Dateien mit Datum 2023.

Kommandos zum Bearbeiten von Textdateien

Ausgaben mit grep filtern

Filtern von Programmausgaben mit grep:

`ls -la | grep 2023`

- zeigt alle Dateien mit Datum 2023.

`ls -la | grep :`

- zeigt alle Dateien jünger als ein Jahr
  (wegen des speziellen Formats von `ls -l`, ansehen!)

Shellskripte

grep: Suchtext am Zeilenanfang/-ende verankern

"^text" $\Rightarrow$ text muss am Zeilenanfang stehen

"text$" $\Rightarrow$ text muss am Zeilenende stehen

```
$ grep Text text.txt
Der Text steht in der Mitte
Text muss am Anfang stehen
Am Ende steht der Text

$ grep "^Text" text.txt
Text muss am Anfang stehen

$ grep "Text$" text.txt
Am Ende steht der Text
```

Suchen und Ersetzen

```
Bielefled;21243;mittel;Station 44;1.Januar 2020
Herford;5741;hoch;Mast 38;1.Januar 2020
Gütersloh;28759;mittel;Mast 92;1.Januar 2020
Bielefled;12535;hoch;Mast 81;2.Januar 2020
Herford;20885;niedrig;Mast 3;2.Januar 2020
...
```

- Erinnerung: `sed` ist “Suchen und Ersetzen” für die Kommandozeile

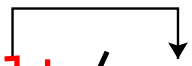
Suchen und Ersetzen

innerhalb von Textdateien

sed: script editor - "Suchen und Ersetzen" per Kommandozeile

Ersetzen des ersten Vorkommens:

```
$ echo "alt alt alt" | sed    "s/alt/neu/"
$ neu alt alt
```



Ersetzen aller Vorkommen:

```
$ echo "alt alt alt" | sed    "s/alt/neu/g"
$ neu neu neu
```

- s - Betriebsart (hier: Ausdruck suchen und ersetzen;
  es gibt noch weitere, aber s ist die häufigste)
- g - Modifier (hier: globale Ersetzung)

Suchen und Ersetzen

Fehler in der Tabelle korrigieren

```
$ sed "s/Bielefled/Bielefeld/g" < messung-typo.csv

Bielefeld;21243;mittel;Station 44;1.Januar 2020
Herford;5741;hoch;Mast 38;1.Januar 2020
Gütersloh;28759;mittel;Mast 92;1.Januar 2020
Bielefeld;12535;hoch;Mast 81;2.Januar 2020
...
```

Suchen und Ersetzen

Fehler in der Tabelle korrigieren

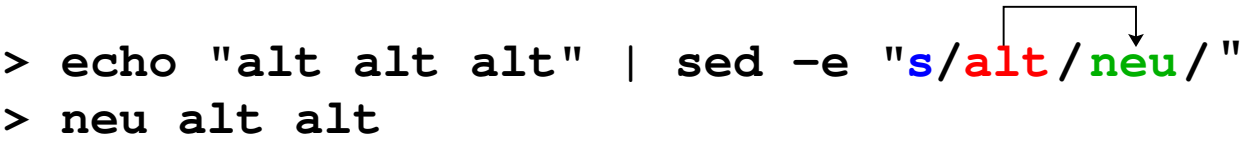
```
$ sed "s/Bielefled/Bielefeld/g" < messung-typo.csv

Bielefeld;21243;mittel;Station 44;1.Januar 2020
Herford;5741;hoch;Mast 38;1.Januar 2020
Gütersloh;28759;mittel;Mast 92;1.Januar 2020
Bielefeld;12535;hoch;Mast 81;2.Januar 2020
...
```

Das ist schon sehr, sehr nützlich!

(Zeige: in `demo-x` das jvx $\rightarrow$ obj)

Andere Betriebsart für sed



Betriebsart y:

Buchstaben aus Liste1 durch diejenigen aus Liste2 ersetzen

Suchen und Ersetzen

Beispiel

(den folgenden Ausdruck in eine Zeile schreiben!)

```
$ echo "HALLO" |
 sed -e "y/ABCDEFGHIJKLMNOPQRSTUVWXYZ
          /abcdefghijklmnopqrstuvywyz/"
hallo
```

Tabellen

Arbeiten mit Tabellen

CSV-Tabellen

Ein zeitloser Klassiker:

CSV: comma separated values oder *character separated values*

(siehe Excel, SQL, ...)

Darstellung von Tabellen als einfache Textdateien:

```
Bielefeld;21243;mittel;Station 44;1.Januar 2021
Herford;5741;hoch;Mast 38;1.Januar 2021
Gütersloh;28759;mittel;Mast 92;1.Januar 2021
Bielefeld;12535;hoch;Mast 81;2.Januar 2021
```

Trennzeichen (hier: ;) beliebig wählbar
solange es nicht innerhalb der Daten vorkommt!

Arbeiten mit Tabellen

CSV-Tabellen

Ein zeitloser Klassiker:

CSV: comma separated values oder *character separated values*

(siehe Excel, SQL, ...)

Darstellung von Tabellen als einfache Textdateien:

```
Bielefeld;21243;mittel;Station 44;1.Januar 2021
Herford;5741;hoch;Mast 38;1.Januar 2021
Gütersloh;28759;mittel;Mast 92;1.Januar 2021
Bielefeld;12535;hoch;Mast 81;2.Januar 2021
```

Trennzeichen (hier: `;`) beliebig wählbar
solange es nicht innerhalb der Daten vorkommt!

Falls wir andere Trennzeichen brauchen, ist das nun einfach: `sed`.

Zusammenfügen von Tabellen mit `join`

Manchmal (z.B. für das erste Programmierprojekt) möchte man Tabellen zusammenfügen. Beispiel:

```
$ cat naehrstoffe.txt
1 Protein
2 Kohlenhydrate
3 Fett
4 Alkohol

$ cat speisen.txt
1 Hühnchen
2 Kartoffeln
3 Butter
4 Schnaps
```

Zusammenfügen von Tabellen mit `join`

Manchmal (z.B. für das erste Programmierprojekt) möchte man Tabellen zusammenfügen. Beispiel:

```
$ cat naehrstoffe.txt
1 Protein
2 Kohlenhydrate
3 Fett
4 Alkohol

$ cat speisen.txt
1 Hühnchen
2 Kartoffeln
3 Butter
4 Schnaps
```

Die erste Spalte ist identisch, nach der wird gejoint:

```
$ join naehrstoffe.txt speisen.txt
1 Protein Hühnchen
2 Kohlenhydrate Kartoffeln
3 Fett Butter
4 Alkohol Schnaps
```

Zusammenfügen von Tabellen mit `join`

`join` möchte, dass die Join-Felder (gleich) sortiert sind.
Falls nicht:

```
$ cat naehrstoffe.txt
1 Protein
2 Kohlenhydrate
3 Fett
4 Alkohol

$ cat speisen.txt
2 Kartoffeln
1 Hühnchen
3 Butter
4 Schnaps
```

...dann gibt es eine Fehlermeldung:

```
$ join naehrstoffe.txt speisen.txt
join: speisen.txt:2: is not sorted: 1 Hühnchen
[...]
join: input is not in sorted order
```

Zusammenfügen von Tabellen mit `join`

Falls in *beiden* Spalten die Reihenfolge 2-1-3-4 ist, klappt's auch.

Gute Praxis ist: erst `sort`, dann `join`.

Zusammenfügen von Tabellen mit `join`

Falls in *beiden* Spalten die Reihenfolge 2-1-3-4 ist, klappt's auch.

Gute Praxis ist: erst `sort`, dann `join`.

Natürlich kann man die Join-Felder auch explizit angeben:

```
$ cat speisen2.txt
Hühnchen 1
Kartoffeln 2
Butter 3
Schnaps 4
```

Zusammenfügen von Tabellen mit `join`

Falls in *beiden* Spalten die Reihenfolge 2-1-3-4 ist, klappt's auch.

Gute Praxis ist: erst `sort`, dann `join`.

Natürlich kann man die Join-Felder auch explizit angeben:

```
$ cat speisen2.txt
Hühnchen 1
Kartoffeln 2
Butter 3
Schnaps 4

$ join -1 1 -2 2 naehrstoffe.txt speisen2.txt
1 Protein Hühnchen
2 Kohlenhydrate Kartoffeln
3 Fett Butter
4 Alkohol Schnaps
```

Zusammenfügen von Tabellen mit `join`

Falls in *beiden* Spalten die Reihenfolge 2-1-3-4 ist, klappt's auch.

Gute Praxis ist: erst `sort`, dann `join`.

Natürlich kann man die Join-Felder auch explizit angeben:

```
$ cat speisen2.txt
Hühnchen 1
Kartoffeln 2
Butter 3
Schnaps 4

$ join -1 1 -2 2 naehrstoffe.txt speisen2.txt
1 Protein Hühnchen
2 Kohlenhydrate Kartoffeln
3 Fett Butter
4 Alkohol Schnaps
```

Wie bei vielen anderen Befehlen: der Rest ist googlen.
(stackoverflow / stackexchange ist oft hilfreich!)

Arbeiten mit Tabellen

Spalten aus CSV-Tabellen auswählen

cut: Spalten aus Tabellen auswählen

Aufruf: `cut -d` *trennzeichen* `-f` *spalten*

Trennzeichen mit Bedeutung in der Shell "entschärfen":

```
cut -d \;   ...
cut -d \␣   ...
```

Arbeiten mit Tabellen

Spalten aus CSV-Tabellen auswählen

cut: Spalten aus Tabellen auswählen

Aufruf: `cut -d` *trennzeichen* `-f` *spalten*

Trennzeichen mit Bedeutung in der Shell "entschärfen":

```
cut -d \;   ...
cut -d \␣   ...
```

typische Spaltenauswahlen:

`cut -f 2,5,9`	Spalten 2,5,9 auswählen
`cut -f 2-4,7`	Spalten 2 bis 4 und 7
`cut -f 5-`	alle Spalten ab der 5.

Arbeiten mit Tabellen

Beispiele

Spalten 1,2 und 4 auswählen:

```
$ cut -d\; -f1,2,4 messung.csv
Bielefeld;21243;Station 44
Herford;5741;Mast 38
Gütersloh;28759;Mast 92
```

Arbeiten mit Tabellen

Beispiele

Spalten 1,2 und 4 auswählen:

```
$ cut -d\; -f1,2,4 messung.csv
Bielefeld;21243;Station 44
Herford;5741;Mast 38
Gütersloh;28759;Mast 92
```

Spalten 1,2 und 5 nur für Bielefeld auswählen:

```
$ grep Bielefeld messung.csv | cut -d\; -f 1-2,5
Bielefeld;21243;1.Januar.2021
Bielefeld;12535;2.Januar.2021
Bielefeld;24817;3.Januar.2021
...
```

Tabellen mit Leerzeichen als Spaltentrennern

Ausgabe von ls spaltenweise zerlegen

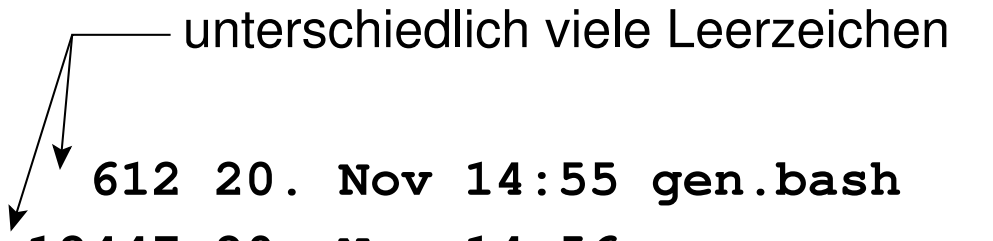
Ziel: In der Ausgabe von `ls -l` Größe und Namen von Dateien (Spalten 5,9) extrahieren.

Tabellen mit Leerzeichen als Spaltentrennern

Ausgabe von ls spaltenweise zerlegen

Ziel: In der Ausgabe von `ls -l` Größe und Namen von Dateien (Spalten 5,9) extrahieren.

Problem: `cut` betrachtet 3 Leerzeichen als 3 leere Spalten!




```
> ls -l
-rwxr--r-- 1 cg cg  612 20. Nov 14:55 gen.bash
-rw-r--r-- 1 cg cg 12447 20. Nov 14:56 messung.csv
```

```
$ ls -l | cut -d\  -f 5,9
 Nov
12447 messung.csv
```

Tabellen mit Leerzeichen als Spaltentrennern

einzelne Zeichen umwandeln oder zusammenfassen

tr: Zeichen umwandeln oder zusammenfassen

Zeichen komprimieren:

```
$ echo "abxxxbacxxxxxxb" | tr -s "x"
abxbacxb
```

Zeichen umwandeln:

```
$ echo "abxxxbaccxxxxxxb" | tr "xc" "yd"
abyyybaddyyyyyyb
```

Tabellen mit Leerzeichen als Spaltentrennern

einzelne Zeichen umwandeln oder zusammenfassen

tr: Zeichen umwandeln oder zusammenfassen

Zeichen komprimieren:

```
$ echo "abxxxbacxxxxxxb" | tr -s "x"
abxbacxb
```

Zeichen umwandeln:

```
$ echo "abxxxbaccxxxxxxb" | tr "xc" "yd"
abyyybaddyyyyyyb
```

Groß-/Kleinschreibung konvertieren:

```
$ echo GROSS | tr [:upper:]  [:lower:]
gross
```

Tabellen mit Leerzeichen als Spaltentrennern

Lösung zum Auswählen von Spalten aus ls -l

```
ls -l | tr -s " " | cut -d\ -f 5,9
```

```
612 gen.sh
238 ls-size.sh
12447 messung.csv
283 rechner.sh
4502 verbrauch.txt
4096 verzeichnis
```

Zusammenfassung heute

`wc`	zählt Zeichen, Worte, Zeilen
`$(...)`	Ausgabe des Befehls benutzen
	Variablen in der Shell
`grep`	nach Zeichenkette in Datei suchen
`sed`	Suchen und Ersetzen von Zeichenketten
`join`	Zusammenfügen von “Tabellen”
`csv`	zeitloses Tabellenformat
`cut`	einzelne Einträge aus Zeilen auswählen
`tr`	stutze Zeichenketten zurecht

...sowie Vorstellung des Programmierprojekts.

Sie probieren das alles gleich in den Tutorien aus.

Viel Erfolg beim Programmierprojekt