

## 1.6 Anwendung/Umleitung und Verkettung

### 1.6.1 Problem

Die Marketing-Abteilung eines Betriebes möchte ein neues Produkt zuerst in Gegenden bewerben, in denen besonders viele Menschen zwischen 20 und 30 Jahren leben. Es gibt dazu von einer Behörde eine CSV-Datei mit Bevölkerungsdaten deutscher Städte und Landkreise. Wie kann man die passenden Daten möglichst schnell extrahieren und aufbereiten?

Ein alter Linux-/Unix-Hase wirft daraufhin Begriffe wie *Pipes*, *E-/A-Umleitung* und *Filter* in den Raum. Was hat es damit auf sich?

### 1.6.2 Schnittstellen

Wie schon beim Thema Prozesse angedeutet, hat jeder Prozess verschiedene Schnittstellen zur Außenwelt (Abbildung 1). Diese Schnittstellen sollen nun benutzt werden, um Informationen zwischen

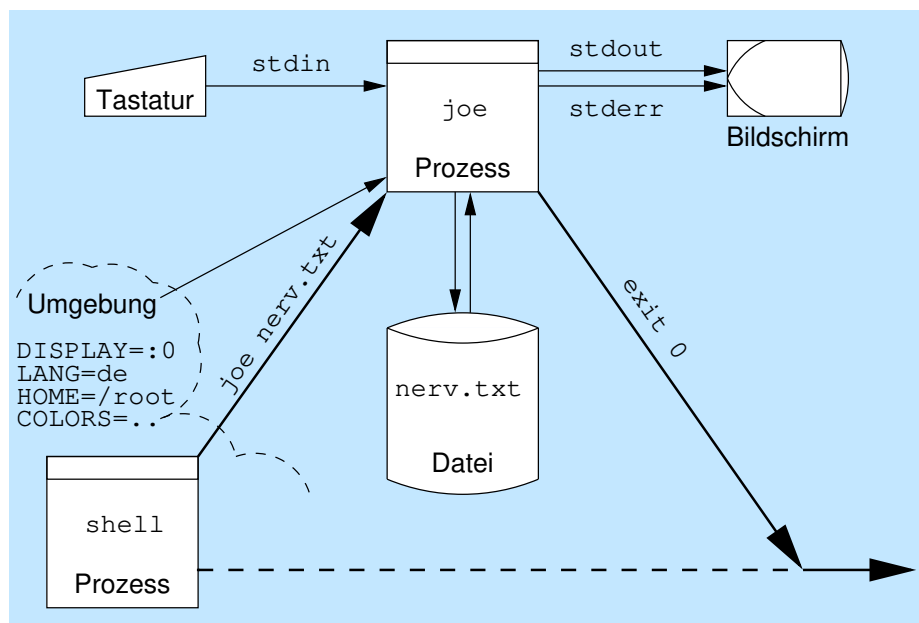


Abbildung 1: Schnittstellen eines Prozesses

einem Prozess und Dateien sowie zwischen Prozessen untereinander zu übermitteln.

### 1.6.3 Ausgabe-Umleitung: `stdout`

Die Bildschirmausgabe eines Prozesses ist der Kanal, der bei Programmiersprachen mit Methoden bzw. Funktionen wie `printf()` angesprochen wird. Er hat traditionell den Namen `stdout` (*standard output*) und die Kanalnummer 1. Das ist die Nummer des File-Deskriptors beim Systemaufruf.

Bei der Ausgabe-Umleitung wird diese Bildschirmausgabe in eine Datei umgeleitet (Abbildung 2).

Mit der folgenden Zeile wird die Bildschirmausgabe von `date` in die Datei `datum.txt` umgeleitet:

```
jk@r155pc1:~$ date 1> datum.txt
```

Man kann die Kanalnummer (die 1) weglassen, weil Kanal 1 der Standard ist:

```
jk@r155pc1:~$ date > datum.txt
```

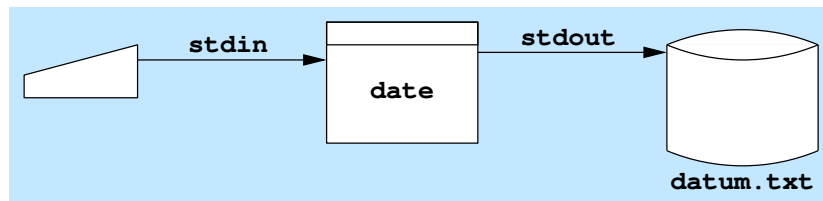


Abbildung 2: Ausgabe-Umleitung

Man sieht, dass dieser Befehl nun nichts auf den Bildschirm bringt. Seine Ausgabe liegt nun in `datum.txt`.

```
jk@r155pc1:~$ cat datum.txt
Mo 13. Nov 22:15:15 CET 2017
```

Gab es die Datei `datum.txt` schon vorher, so ist ihr bisheriger Inhalt für immer verloren. Zur Beruhigung: Mit dem Shell-Befehl `set -C` kann das Überschreiben der Datei wirkungsvoll verhindert werden:

```
jk@r155pc1:~$ set -C
jk@r155pc1:~$ date > datum.txt
jk@r155pc1:~$ date > datum.txt
bash: datum.txt: Kann existierende Datei nicht überschreiben.
```

Soll die Datei danach trotzdem überschrieben werden, gelingt das mit dem Operator `>|`:

```
jk@r155pc1:~$ date >| datum.txt
```

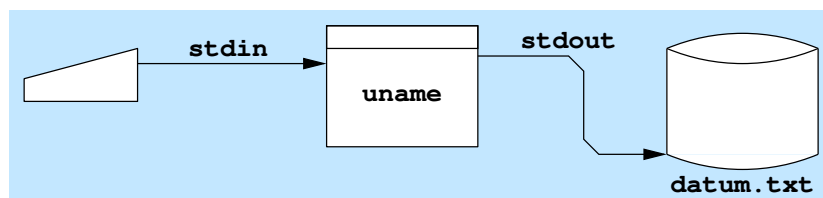


Abbildung 3: Ausgabe-Umleitung, anhängend

Oft möchte man die bisherige Information in einer Datei erhalten und neue Information an diese Datei **anhängen** (Abbildung 3). Auch das ist möglich, und zwar mit dem Operator `>>`:

```
jk@r155pc1:~$ date > datum.txt
jk@r155pc1:~$ uname >> datum.txt
```

Hätte es die Datei `datum.txt` vorher noch nicht gegeben, so wäre sie jetzt neu angelegt worden.

#### 1.6.4 Fehlerausgabe-Umleitung: `stderr`

Jeder Prozess hat nicht nur einen normalen Ausgabekanal für die auszugebenden Nutzdaten, sondern auch einen eigenen Kanal für Fehlermeldungen. Dieser Kanal ist nicht gepuffert, so dass Fehlermeldungen sofort ausgegeben werden können. Dieser Kanal hat den Namen `stderr` (*standard error*) und die Kanalnummer 2.

Auch dieser Kanal kann umgeleitet werden (Abbildung 4). Es wird wieder der Operator `>` verwendet, allerdings jetzt mit der Kanalnummer 2 davor (der Default-Wert war ja Kanal 1).

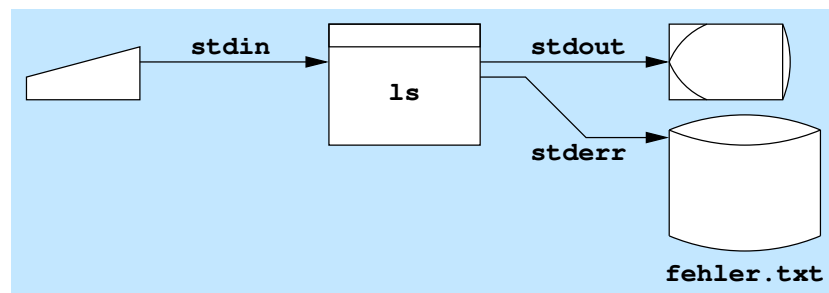


Abbildung 4: Fehlerausgabe-Umleitung

```
jk@r155pc1:~$ ls texte
supertext.txt
jk@r155pc1:~$ ls trxte
ls: Zugriff auf trxte nicht möglich: Datei oder Verzeichnis nicht gefunden
jk@r155pc1:~$ ls trxte 2> fehler.txt
```

Jetzt sind die Fehler in der Datei fehler.txt:

```
jk@r155pc1:~$ cat fehler.txt
ls: Zugriff auf trxte nicht möglich: Datei oder Verzeichnis nicht gefunden
```

Wenn es die Datei fehler.txt schon vorher gab, so ist ihr bisheriger Inhalt wiederum verloren. Auch hier kann das Überschreiben der Datei durch den Shell-Befehl `set -C` verhindert werden. Und auch hier erzwingt der Operator `2>|` das Überschreiben.

```
jk@r155pc1:~$ set -C
jk@r155pc1:~$ ls trxte 2> fehler.txt
jk@r155pc1:~$ ls trxte 2> fehler.txt
bash: fehler.txt: Kann existierende Datei nicht überschreiben.
jk@r155pc1:~$ ls trxte 2>| fehler.txt
```

Auch beim Kanal 2 ist es möglich, eine neue Information anzuhängen. Dazu dient der Operator `2>>`:

```
jk@r155pc1:~$ ls trxte 2> fehler.txt
jk@r155pc1:~$ ls tixte 2>> fehler.txt
jk@r155pc1:~$ cat fehler.txt
ls: Zugriff auf trxte nicht möglich: Datei oder Verzeichnis nicht gefunden
ls: Zugriff auf tixte nicht möglich: Datei oder Verzeichnis nicht gefunden
```

### 1.6.5 Kombination zweier Umleitungen

Häufig möchte man beide Ausgabe-Kanäle umleiten. Hier ist ein Beispiel:

```
jk@r155pc1:~$ ls texte trxte
ls: Zugriff auf trxte nicht möglich: Datei oder Verzeichnis nicht gefunden
texte:
supertext.txt
```

Die erste Zeile war die Fehlermeldung auf Kanal 2, die beiden anderen Zeilen sind die übliche Ausgabe des Verzeichnisses von `texte`.

Wenn man jeden der beiden Kanäle in eine andere Datei umleitet, so klappt es:

```
jk@r155pc1:~$ ls texte trxte > normal.txt 2> fehler.txt
jk@r155pc1:~$ cat normal.txt
texte:
supertext.txt
jk@r155pc1:~$ cat fehler.txt
ls: Zugriff auf trxte nicht möglich: Datei oder Verzeichnis nicht gefunden
```

Wenn man aber beide Kanäle in dieselbe Datei umleitet, gibt es Probleme:

```
jk@r155pc1:~$ ls texte trxte > gesamt.txt 2> gesamt.txt
jk@r155pc1:~$ cat gesamt.txt
texte:
supertext.txt
nicht möglich: Datei oder Verzeichnis nicht gefunden
```

Die Umleitungen vertragen sich nicht unbedingt. Es ist so, als wenn zwei verschiedene (Teil-) Prozesse gleichzeitig in dieselbe Datei schreiben wollen: Ohne Absprache geht es nicht; meist gewinnt derjenige, der als Letzter schreibt.

Die Lösung besteht darin, dass ein Kanal denselben File-Deskriptor benutzt wie der andere (Abbildung 5). Dazu wird der Operator `2>&1` benutzt. Er bedeutet: Kanal 2 schreibt in den Filedeskriptor, in den Kanal 1 schon jetzt schreibt.

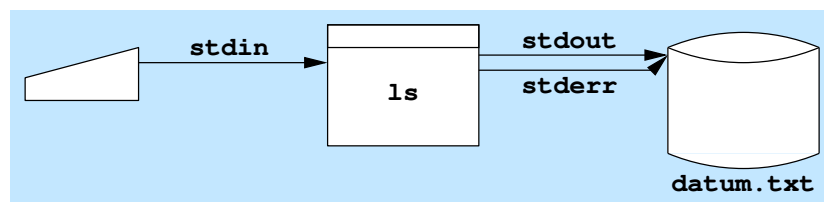


Abbildung 5: Umleitung beider Kanäle

```
jk@r155pc1:~$ ls texte trxte > gesamt.txt 2>&1
jk@r155pc1:~$ cat gesamt.txt
ls: Zugriff auf trxte nicht möglich: Datei oder Verzeichnis nicht gefunden
texte:
supertext.txt
```

Die Shell arbeitet die Umleitungen von links nach rechts ab: Zuerst wird Kanal 1 nach `gesamt.txt` umgeleitet, dann wird Kanal 2 dort angehängt, wo Kanal 1 hinschreibt.

Man kann denselben Effekt auch so erreichen:

```
jk@r155pc1:~$ ls texte trxte 2> gesamt.txt 1>&2
```

Für viele ist das noch zu unhandlich, deshalb gibt es einen Operator, der als Abkürzung beide Umleitungen vornimmt:

```
jk@r155pc1:~$ ls texte trxte &> gesamt.txt
```

### 1.6.6 Eingabe-Umleitung: `stdin`

Man kann auch die Eingabe eines Programms zu einer Datei umleiten. Die Eingabe ist der Kanal, der bei Programmen meist durch Methoden oder Funktionen wie `read()` oder `scanf()` abgefragt wird. Er hat den Namen `stdin` (*standard input*) und die Kanalnummer 0.

Der Operator für die Eingabeumleitung ist das Zeichen `<`. Das ist ein Unterschied zu allen Ausgabeumleitungen! Mit einer Zeile der Form `befehl < datei` nimmt das Programm `befehl` alle Eingaben statt von der Tastatur von der Datei `datei` entgegen (Abbildung 6).

Zuerst legt man eine Datei an, die die notwendigen Eingaben enthalten muss. In diesem Beispiel soll von einem FTP-Server eine Datei geholt werden. Die Datei sieht so aus:

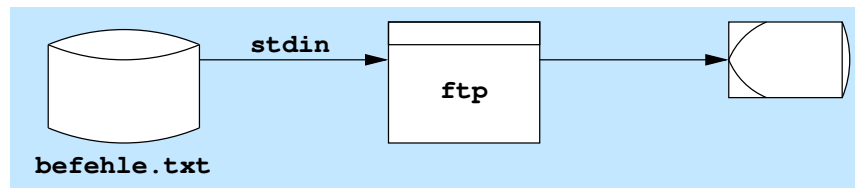


Abbildung 6: Eingabe-Umleitung

```

1 binary
2 cd public
3 cd CPAN
4 get ls-lR.gz
5 bye

```

Außerdem muss (nur in diesem Beispiel) zur automatischen Passwordeingabe eine Datei `$HOME/.netrc` vorhanden sein, die den Hostnamen, den Benutzernamen und das Passwort des ftp-Servers enthält:

```

1 machine      ftp.ruhr-uni-bochum.de
2 login       anonymous
3 password    user@csbme.de

```

Dann kann man die Befehlszeile abschicken:

```

jk@r155pc1:~$ ftp -v ftp.ruhr-uni-bochum.de < befehle.txt
Connected to vmrz0382.rz.ruhr-uni-bochum.de.
220 ProFTPD 1.3.4a Server (FTP-Server der Ruhr-Universitaet Bochum)
[2a05:3e00:1:1003::32:58]
...
jk@r155pc1:~$ ls -s ls-lR.gz
7148 ls-lR.gz

```

Man sieht, dass die Ausgaben wie gewohnt auf den Bildschirm ausgegeben werden. Nur von der Tastatur nimmt diese Befehlszeile nichts entgegen, ähnlich einem Programm, das im Hintergrund gestartet wurde. Eine Ausnahme sind Tastenkombinationen wie `Strg - C` und `Strg - Z`, die am Programm vorbei direkt mit der Shell kommunizieren.

### 1.6.7 Here-Dokument und Here-String

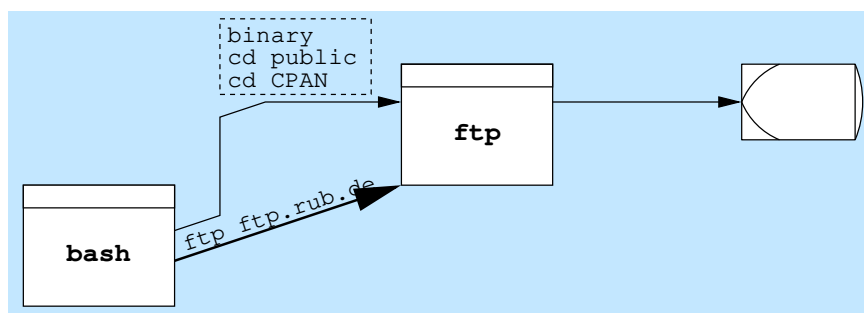


Abbildung 7: Here-Dokument

Für den Fall, dass man keine Eingabedatei anlegen will oder kann, kann man die Eingabebefehle auch temporär in der Befehlseingabe ablegen. Diese Technik heißt Here-Dokument (Abbildung 7). Hier ist ein Beispiel, die Benutzereingaben sind in Fettdruck.

```
jk@r155pc1:~$ ftp -v ftp.ruhr-uni-bochum.de << _ENDE_
> binary
> cd public/CPAN
> get ls-lR.gz
> bye
> _ENDE_
Connected to vmrz0382.rz.ruhr-uni-bochum.de.
220 ProFTPD 1.3.4a Server (FTP-Server der Ruhr-Universitaet Bochum)
[2a05:3e00:1:1003::32:58]
...
```

Nach dem Umleitungszeichen legt man ganz willkürlich ein Schlüsselwort fest. Hier heißt das Schlüsselwort `_ENDE_`. Alle weiteren Zeilen werden durch die Shell im Here-Dokument abgelegt. Die Shell macht das durch ein anderes Prompt deutlich<sup>1</sup>. Das geht so lange, bis man eine Zeile eingibt, die nur aus dem Schlüsselwort besteht. Erst dann wird die ganze Zeile abgeschickt – mit der Eingabe aus dem Here-Dokument.

Es geht noch einfacher: Die Bash erlaubt auch die Eingabe eines so genannten Here-Strings. Dabei werden die Eingabebefehle in einer Zeichenkette abgelegt, die durch Anführungszeichen geklammert ist und auch Zeilenendezeichen enthalten darf:

```
jk@r155pc1:~$ ftp -v ftp.ruhr-uni-bochum.de <<< "binary
cd public/CPAN
get ls-lR.gz
bye"
```

### 1.6.8 Befehlspipeline

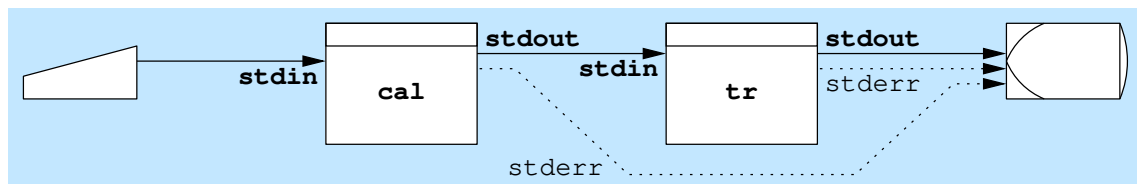


Abbildung 8: Befehlspipeline

- Bei `befehl1 | befehl2` geht die Tastatureingabe an `befehl1`. Dessen Ausgabe geht als Eingabe an `befehl2`. Die Ausgabe von `befehl2` geht auf den Bildschirm.
- Beispiel: `ls -l | grep rwx` — `ls` gibt den Verzeichnisinhalt aus, `grep rwx` filtert die Zeilen, in denen der Text `rwx` vorkommt.

### 1.6.9 Mehrere Befehle in einer Zeile

Bei der Befehlspipeline kommunizieren zwei Prozesse miteinander. Es gibt aber auch den Fall, dass zwei Prozesse einfach nebeneinander oder nacheinander laufen sollen. Davon soll jetzt die Rede sein.

Bei der Befehlssequenz werden zwei Befehle nacheinander ausgeführt (Abbildung 9). Erst dann, wenn der erste Befehl abgearbeitet ist, wird der zweite begonnen, wie man hier an der Ausgabe sieht:

```
jk@r155pc1:~$ hostname;uname
r155pc1
linux
```

<sup>1</sup>Festgelegt in PS2, hier ist es ein Größer-Zeichen.

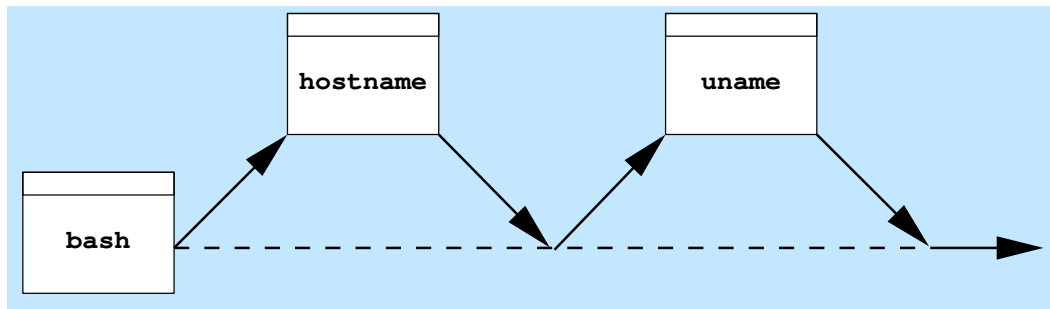


Abbildung 9: Befehlssequenz

Das Semikolon ersetzt hier einfach das Zeilenende.

Manchmal ist es nur dann sinnvoll, einen Befehl auszuführen, wenn der davor ausgeführte Befehl erfolgreich war. Die meisten Befehle geben bei Erfolg den Exitcode (Rückgabewert) 0 zurück. Ein Exitcode ungleich 0 gibt dann oft noch die Art des Fehlers an. Für die Kommunikation zwischen Prozessen ist so ein Exitcode schneller und eindeutiger als eine Fehlermeldung auf dem Bildschirm. Den Exitcode des letzten Prozesses erhält man wie folgt:

```

jk@r155pc1:~$ ls -l a.txt
ls: Zugriff auf a.txt nicht möglich: Datei oder Verzeichnis nicht gefunden
jk@r155pc1:~$ echo $?
2
  
```

Der fehlgeschlagene `ls`-Befehl hat also den Exitcode 2 gehabt.

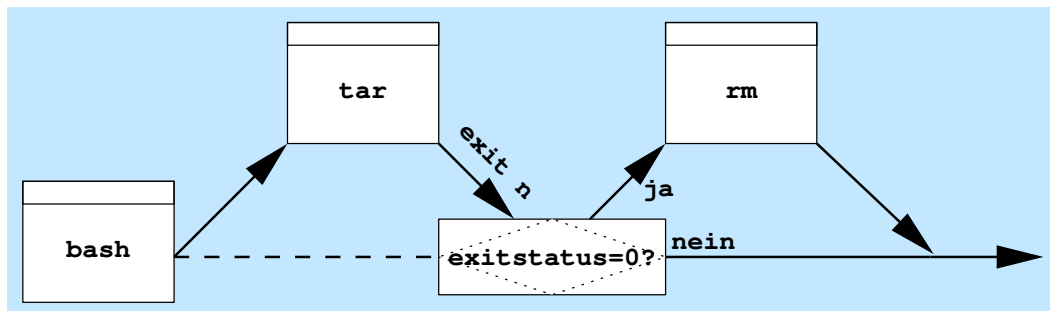


Abbildung 10: Befehlssequenz mit Und-Verknüpfung

Mit der Befehlszeile `befeh11 && befeh12` kann man diesen Exitcode ausnutzen: Hier wird `befeh12` nur dann ausgeführt, wenn `befeh11` erfolgreich war, also einen Exitcode gleich null ergab (Abbildung 10). Hier ist ein übliches Beispiel:

```

jk@r155pc1:~$ tar -cf sicherung.tar verz && rm -r verz
  
```

Mit `tar` wird das Verzeichnis `verz` gesichert. Nur dann, wenn die Sicherung erfolgreich war, kann das Verzeichnis anschließend gelöscht werden.

Mit der Befehlszeile `befeh11 || befeh12` ist es umgekehrt. Hier wird `befeh12` nur dann ausgeführt, wenn `befeh11` *nicht* erfolgreich war, also einen Rückgabewert ungleich null ergab (Abbildung 11). Ein Beispiel:

```

jk@r155pc1:~$ gzip sicherung.tar || compress sicherung.tar
  
```

Die Datei `sicherung.tar` soll unbedingt komprimiert werden. Wenn `gzip` nicht verfügbar ist, soll es mit `compress` versucht werden.

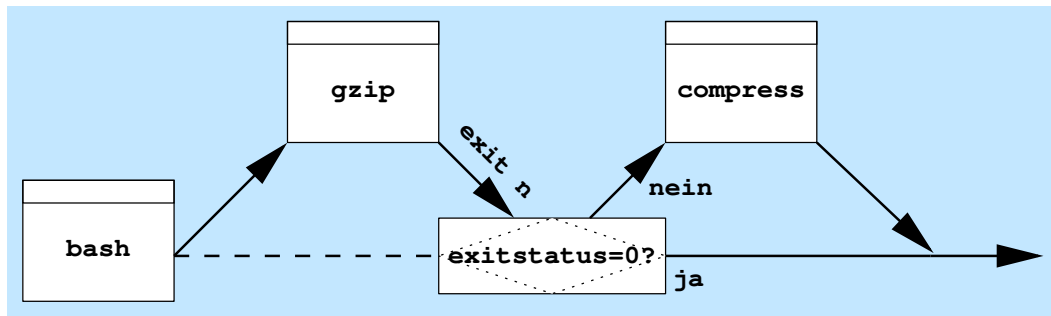


Abbildung 11: Befehlssequenz mit Oder-Verknüpfung

Wenn Befehle keine Tastatureingabe benötigen, kann man sie auch parallel laufen lassen – nämlich im Hintergrund. Der Operator `&`, mit dem man einen Befehl im Hintergrund startet, dient genauso als Trenner zwischen Befehlen wie das Semikolon.

```
jk@r155pc1:~$ tar -cf sicher1.tar texte & tar -cf sicher2.tar daten
```

### 1.6.10 Filterprogramme

Nun war oben von Filterprogrammen die Rede, die beim Extrahieren und Aufbereiten der Daten helfen sollten.

In Linux gibt es viele kleine Hilfsprogramme, mit denen man Textdateien und Datenströme bearbeiten kann. Etliche von ihnen kann man auf zwei Arten einsetzen.

- Mit (mindestens) einem Dateinamen als Parameter. Das Programm liest die Datei und gibt ein entsprechendes Ergebnis auf den Bildschirm aus (Abbildung 12).

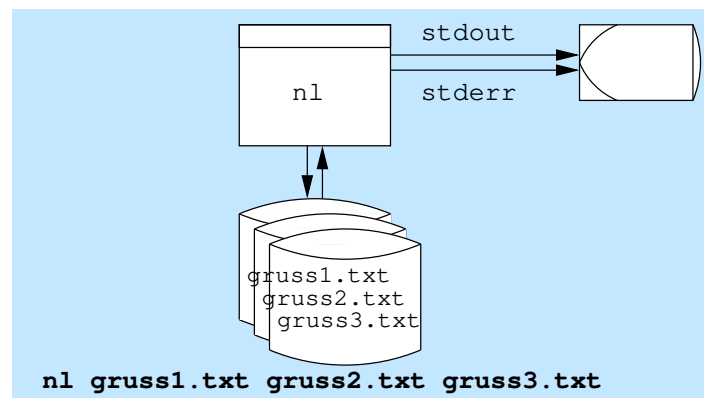


Abbildung 12: Start von nl mit Dateinamen

```
jk@r155pc1:~$ nl gruss.txt
 1 Guten Morgen
 2 Guten Abend
```

- Ohne einen Dateinamen als Parameter (Optionen dürfen trotzdem sein). Das Programm liest von der Standardeingabe (z. B. von der Tastatur) und gibt das entsprechende Ergebnis auf den Bildschirm aus, wie im folgenden Beispiel (Abbildung 13, Nutzereingabe in Fettdruck).

```
jk@r155pc1:~$ nl
```



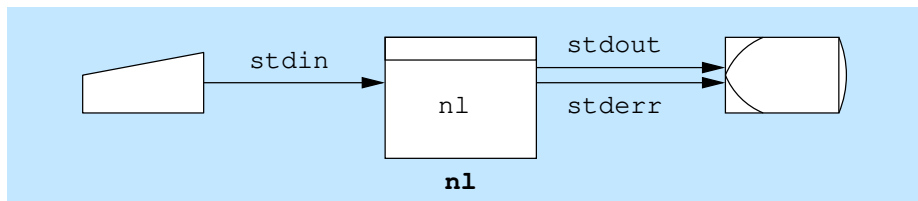


Abbildung 13: Start von nl ohne Dateinamen

```
Guten Morgen ↵
1 Guten Morgen
Guten Abend ↵
2 Guten Abend
[Strg] - [D]
```

Hier arbeitet das Programm als sogenannter *Filter*: Wie bei einem Filter in der Audio-technik wird ein Eingangssignal verarbeitet und das verarbeitete Signal ausgegeben. Das ergibt natürlich eher Sinn bei Verwendung von E-/A-Umleitung und/oder Befehlspipelines (Abbildung 14).

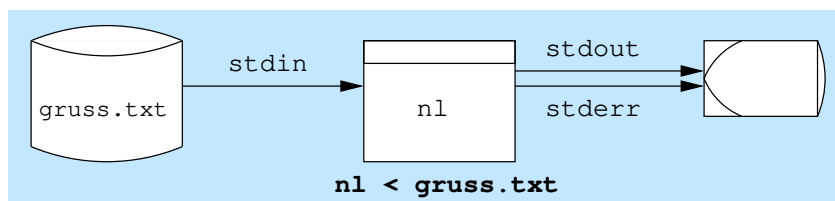


Abbildung 14: Start von nl ohne Dateinamen, aber mit Eingabeumleitung

```
jk@r155pc1:~$ nl < gruss.txt
1 Guten Morgen
2 Guten Abend
```

Programme dieser Art werden oft vereinfachend als Filterprogramme (kurz: Filter) bezeichnet. Tabelle 1 gibt einen Überblick über oft benutzte Filterprogramme.

Man sieht, dass das oben genannte Schema nicht auf jedes dieser Hilfsprogramme passt. So arbeitet `more` absichtlich nicht mit Eingabe über die Tastatur. `less` erlaubt dies, wenn man die Option `-` (einzelnes Minuszeichen) benutzt. Und `paste` und `file` erwarten die Option `-` immer, falls die Standardeingabe (egal, ob Tastatur oder Eingabeumleitung) benutzt wird.

Die genannten Filterprogramme lassen sich in der Regel problemlos kombinieren. Wenn man z. B. ein Verzeichnis nach Dateilängen sortiert (seitenweise) anzeigen will, kann man es so machen:

```
jk@r155pc1:~$ ls -l | rev | sort | rev | less
```

Programm	Zweck	mit Dateinamen	ohne Dateinamen	Ausgabe
cat more less	Einlesen (u. kleben) Pager Pager	einlesen einlesen einlesen	Standardeingabe Standardeingabe Standardeingabe	Standardausgabe Bildschirm (seitenw. nach EOF) Bildschirm (seitenweise)
head tail cut grep uniq	Dateianfang Dateiende Spalten schneiden Zeilen schneiden Doppelte Zeilen entf.	einlesen einlesen einlesen einlesen nur einer	Standardeingabe Standardeingabe Standardeingabe Standardeingabe Standardeingabe	Standardausgabe Standardausgabe Standardausgabe Standardausgabe Standardausgabe oder Datei
tr sed expand unexpand sort tac rev	Zeichen ersetzen Stream-Editor Tabs ersetzen Leerzeichen ersetzen Sortieren Spiegeln (vert.) Spiegeln (horiz.)	nicht möglich nicht möglich einlesen einlesen einlesen einlesen einlesen	Standardeingabe Standardeingabe Standardeingabe Standardeingabe Standardeingabe Standardeingabe Standardeingabe	Standardausgabe Standardausgabe Standardausgabe Standardausgabe Standardausgabe (nach EOF) Standardausgabe (nach EOF) Standardausgabe
nl fmt pr	Zeilennummerierung Textsatz erzeugen Druckformat erz.	einlesen einlesen einlesen	Standardeingabe Standardeingabe Standardeingabe	Standardausgabe Standardausgabe Standardausgabe
od hd	Oktaldump Hexdump	einlesen einlesen	Standardeingabe Standardeingabe	Standardausgabe Standardausgabe
split	Datei schneiden	nur einer	Standardeingabe	Dateien
paste join	Dateien kleben Inner Join	einlesen nur zwei	Standardeingabe nicht möglich	Standardausgabe Standardausgabe
file wc	Dateiart ermitteln Wortzähler	einlesen einlesen	Standardeingabe Standardeingabe	Standardausgabe Standardausgabe
tee sponge netcat	Abzweig in Datei Verz. Ausg.i. Datei Umleitung z. Netz	schreiben schreiben nicht möglich	Standardeingabe Standardausgabe Standardeingabe	Standardausgabe und Datei Standardausgabe oder Datei Standardausgabe

Tabelle 1: Filterprogramme