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

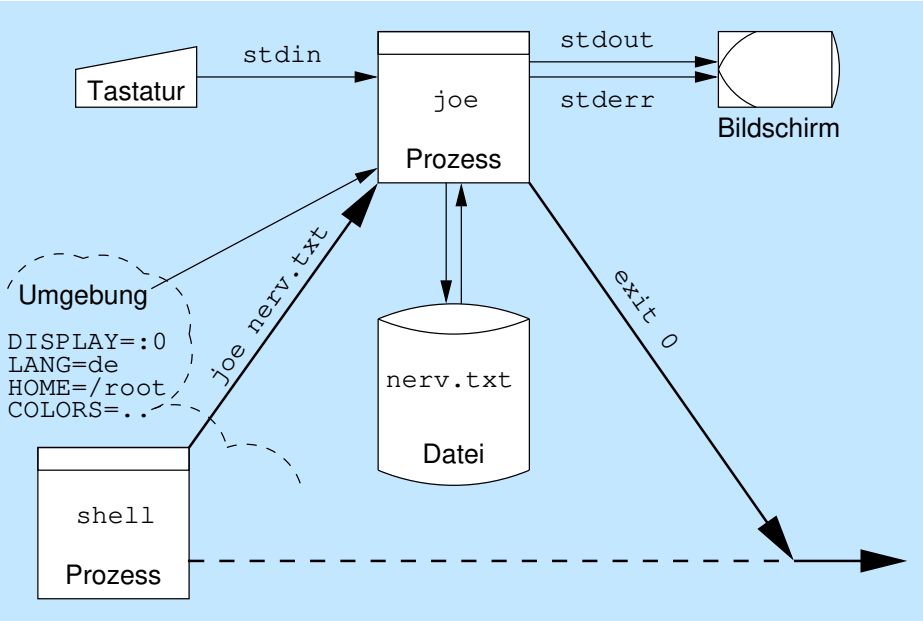

Abbildung 1: Schnittstellen eines Prozesses

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:

Terminal

```
schueler@debian964:~$ date 1> datum.txt
```

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

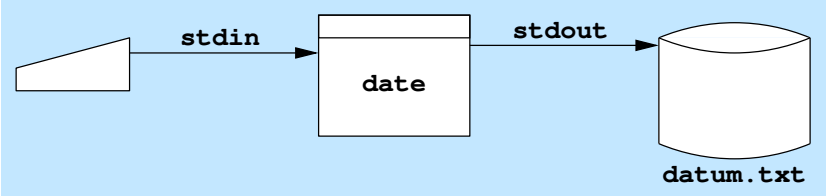

Abbildung 2: Ausgabe-Umleitung

Terminal
```
schueler@debian964:~$ date > datum.txt
```

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

Terminal
```
schueler@debian964:~$ 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:

Terminal
```
schueler@debian964:~$ set -C
schueler@debian964:~$ date > datum.txt
schueler@debian964:~$ date > datum.txt
bash: datum.txt: Kann existierende Datei nicht überschreiben.
```

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

Terminal
```
schueler@debian964:~$ date >| datum.txt
```

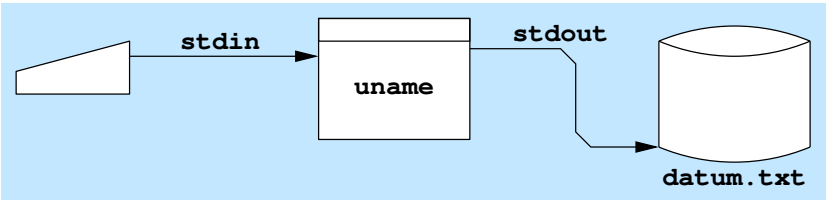

Abbildung 3: Ausgabe-Umleitung, anhängend

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 `>>`:

Terminal
```
schueler@debian964:~$ date > datum.txt
schueler@debian964:~$ 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

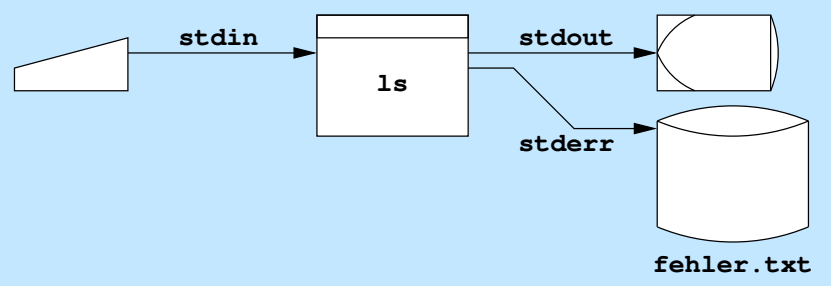

Abbildung 4: Fehlerausgabe-Umleitung

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).

Terminal
```
schueler@debian964:~$ ls texte
supertext.txt
schueler@debian964:~$ ls trxte
ls: Zugriff auf trxte nicht möglich: Datei oder Verz. nicht gefunden
schueler@debian964:~$ ls trxte 2> fehler.txt
```

Jetzt sind die Fehler in der Datei `fehler.txt`:

Terminal
```
schueler@debian964:~$ cat fehler.txt
ls: Zugriff auf trxte nicht möglich: Datei oder Verz. 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.

Terminal
```
schueler@debian964:~$ set -C
schueler@debian964:~$ ls trxte 2> fehler.txt
schueler@debian964:~$ ls trxte 2> fehler.txt
bash: fehler.txt: Kann existierende Datei nicht überschreiben.
schueler@debian964:~$ ls trxte 2>| fehler.txt
```

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

Terminal
```
schueler@debian964:~$ ls trxte 2> fehler.txt
schueler@debian964:~$ ls tixte 2>> fehler.txt
schueler@debian964:~$ cat fehler.txt
ls: Zugriff auf trxte nicht möglich: Datei oder Verz. nicht gefunden
ls: Zugriff auf tixte nicht möglich: Datei oder Verz. nicht gefunden
```

1.6.5 Kombination zweier Umleitungen

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

Terminal
```
schueler@debian964:~$ ls texte trxte
ls: Zugriff auf trxte nicht möglich: Datei oder Verz. nicht gefunden
texte:
supertext.txt
```

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

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

Terminal
```
schueler@debian964:~$ ls texte trxte > normal.txt 2> fehler.txt
schueler@debian964:~$ cat normal.txt
texte:
supertext.txt
schueler@debian964:~$ cat fehler.txt
ls: Zugriff auf trxte nicht möglich: Datei oder Verz. nicht gefunden
```

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

Terminal
```
schueler@debian964:~$ ls texte trxte > gesamt.txt 2> gesamt.txt
schueler@debian964:~$ cat gesamt.txt
texte:
supertext.txt
 nicht möglich: Datei oder Verz. 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.

De 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.

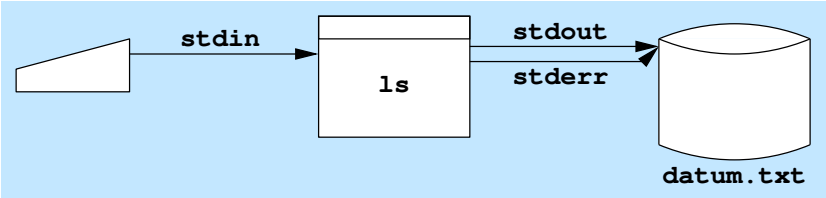

Abbildung 5: Umleitung beider Kanäle

Terminal
```
schueler@debian964:~$ ls texte trxte > gesamt.txt 2>&1
schueler@debian964:~$ cat gesamt.txt
ls: Zugriff auf trxte nicht möglich: Datei oder Verz. 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:

Terminal
```
schueler@debian964:~$ 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:

Terminal
```
schueler@debian964:~$ ls texte trxte &> gesamt.txt
```

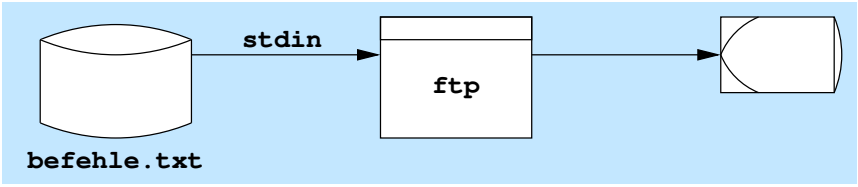

Abbildung 6: Eingabe-Umleitung

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 `stdout` (*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:

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

Außerdem muss (nur in diesem Beispiel) zur automatischen Passworteingabe 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:

Terminal

```
schueler@debian964:~$ 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]
...
schueler@debian964:~$ 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

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.

Terminal

```
schueler@debian964:~$ ftp -v ftp.ruhr-uni-bochum.de << _ENDE_
> binary
```

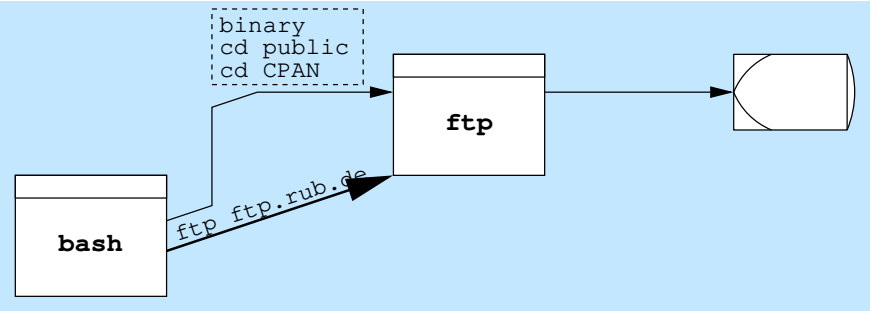

Abbildung 7: Here-Dokument

```
> 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[1]. 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:

Terminal

```
schueler@debian964:~$ ftp -v ftp.ruhr-uni-bochum.de <<< "binary
> cd public/CPAN
> get ls-lR.gz
> bye"
...
schueler@debian964:~$
```

1.6.8 Ein- und Ausgabeumleitung in derselben Zeile

Oft soll ein Text zuerst von einem Programm bearbeitet werden, danach von einem zweiten Programm und danach vielleicht noch von einem dritten Programm. Im folgenden Beispiel soll eine Einkaufsliste (in `eink1.txt`) zuerst sortiert, dann in Großbuchstaben umgewandelt und zuletzt nummeriert werden (die drei benötigten Befehle heißen `sort`, `tr "a-z" "A-Z"` und `nl`). Man benutzt dann in jedem Schritt die Eingabeumleitung *und* die Ausgabeumleitung. Jedes Zwischenergebnis wird in einer temporären Datei gespeichert:

Terminal

```
schueler@debian964:~$ cat eink1.txt
Nudeln
Zitronen
Birnen
schueler@debian964:~$ sort < eink1.txt > eink2.txt # sortieren
```

[1] Festgelegt in `PS2`, hier ist es ein Größer-Zeichen.

```
schueler@debian964:~$ tr "a-z" "A-Z" < eink2.txt > eink3.txt
schueler@debian964:~$ nl < eink3.txt > eink4.txt # nummerieren
schueler@debian964:~$ cat eink4.txt
     1  BIRNEN
     2  NUDELN
     3  ZITRONEN
```

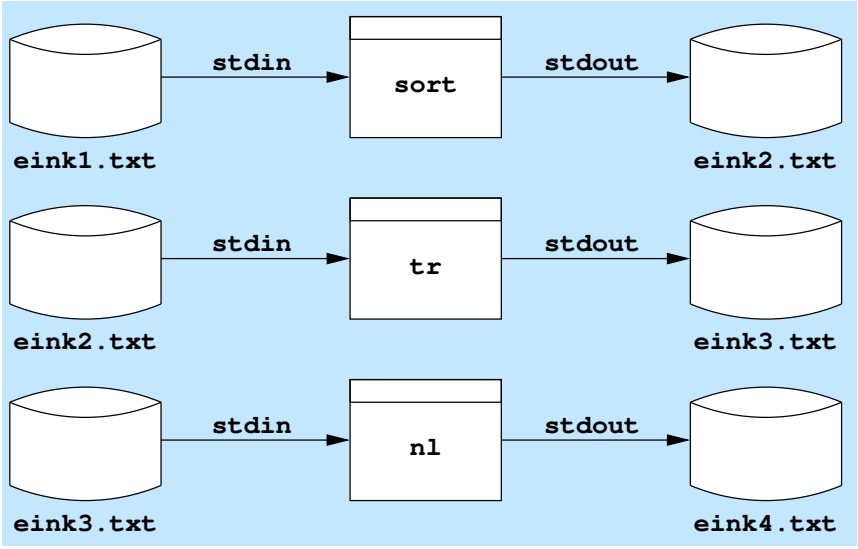

Abbildung 8: Einkaufsliste, Version 1: Ein- und Ausgabeumleitung in derselben Befehlszeile

Abbildung 8 zeigt die Vorgehensweise. In jeder Zeile wird mit der Eingabeumleitung aus einer Datei gelesen und gleichzeitig mit der Ausgabeumleitung in eine andere Datei geschrieben. Wichtig ist: **Es darf niemals in einer Zeile derselbe Dateiname für Ein- und Ausgabeumleitung verwendet werden!** Dann wird nämlich die Eingabedatei sofort (durch das Öffnen zum Schreiben) gelöscht, und alle Daten sind weg[2].

1.6.9 Befehlspipeline

Schön wäre es, wenn man Daten *direkt* von einem Programm zum anderen leiten könnte: Dann brauchte man keine temporären Dateien mehr.

Dazu gibt es in Linux eine Lösung: Die FIFO-Objekte (so genannte *pipes*). Ein FIFO ist eine Datei, in die man auf der einen Seite schreiben und auf der anderen Seite lesen kann. So ein FIFO nimmt (fast) keinen Platz auf einem Massenspeicher weg, und man kann ihn quasi parallel bearbeiten. Mit dem Befehl `mkfifo` kann man solche Objekte manuell anlegen.

Aber hier geht es noch einfacher: Man lässt die Shell unbenannte FIFOs anlegen und auch wieder löschen, indem man ein einziges Zeichen eingibt, den senkrechten Strich. Der wird deshalb oft *pipe* genannt:

Bei `befehl1 | befehl2` geht zuerst die Tastatureingabe an `befehl1`. Die Ausgabe von `befehl1` geht (über eine *pipe*) als Eingabe an `befehl2`. Die Ausgabe von `befehl2` geht wie üblich auf den Bildschirm. Hier ein Beispiel:

Terminal

```
schueler@debian964:~$ cal 2025 | tr "a-z" "A-Z"
                              2025
       JANUAR                 FEBRUAR                    MäRZ
```

[2]Zur Lösung dieses Problems gibt es das Programm `sponge`, das erst dann die Ausgabedatei öffnet, wenn am Eingang keine Daten mehr kommen.

```
SO MO DI MI DO FR SA  SO MO DI MI DO FR SA  SO MO DI MI DO FR SA
12 13 14 15 16 17 18   9 10 11 12 13 14 15  14 15 16 17 18 19 20
12 13 14 15 16 17 18   9 10 11 12 13 14 15   9 10 11 12 13 14 15
          1  2  3  4                    1                    1
...
```

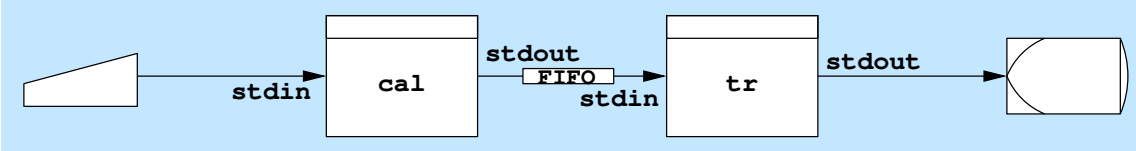

Abbildung 9: Befehlspipeline

Abbildung 9 zeigt, was passiert ist: `cal 2025` gibt den Kalender von 2025 aus; `tr "a-z" "A-Z"` wandelt die erhaltenen Daten um und gibt sie aus.

Das kann man nutzen, um das Problem mit der obigen Einkaufsliste zu lösen (Abbildung10):

Terminal

```
schueler@debian964:~$ sort <eink1.txt | tr "a-z" "A-Z" | nl >eink2.txt
schueler@debian964:~$ cat eink2.txt
     1        BIRNEN
     2        NUDELN
     3        ZITRONEN
```

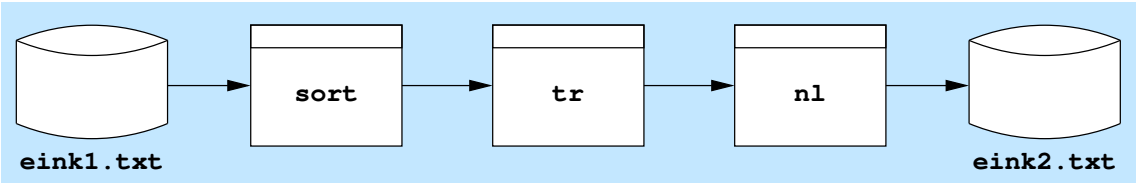

Abbildung 10: Einkaufsliste, Version 2: Befehlspipeline

Statt drei Zeilen braucht man nur noch eine; Fehler mit den Dateinamen sind nicht mehr so wahrscheinlich. Diese Version kann komplett im RAM laufen und ist damit wesentlich schneller und effizienter.

1.6.10 Filterprogramme

Filterprogramme helfen beim Extrahieren und Aufbereiten von Daten. Sie sind so konstruiert, dass sie mit Ein-/Ausgabeumleitung und Befehlspipelines gut zusammenarbeiten. Sie sind sehr gut auf größtmöglichen Datenumsatz optimiert. In Linux gibt es viele dieser kleinen Helfer. Manche von ihnen kann man auf zwei Arten einsetzen:

a) Aufruf mit (mindestens) einem Dateinamen als Parameter. Das Programm liest die Datei(en) und gibt ein entsprechendes Ergebnis auf den Bildschirm aus (Abbildung 11).

Terminal

```
schueler@debian964:~$ nl gruss.txt
     1        Guten Morgen
     2        Guten Abend
```

b) Aufruf ohne einen Dateinamen als Parameter (Optionen dürfen trotzdem sein). Das Programm liest von der Standardeingabe (z. B. von der Tastatur) und gibt das entsprechende

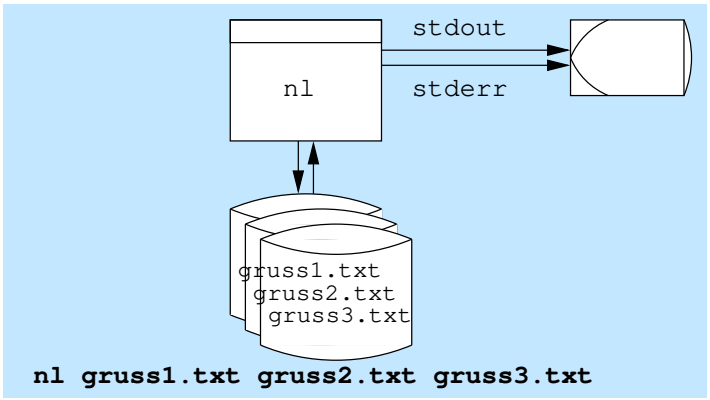

Abbildung 11: Start von `nl` mit Dateinamen

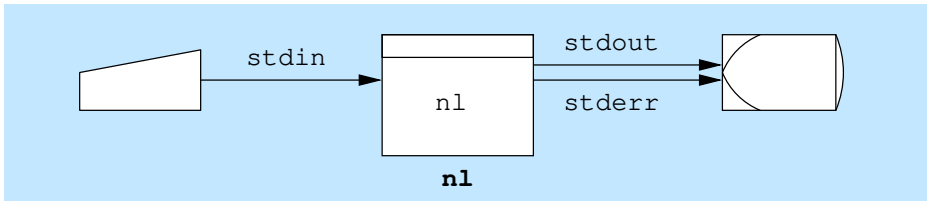

Abbildung 12: Start von `nl` ohne Dateinamen

Ergebnis auf den Bildschirm aus, wie im folgenden Beispiel (Abbildung 12, Nutzereingabe in Fettdruck).

Terminal

```
schueler@debian964:~$ nl
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 Audiotechnik wird ein Eingangssignal verarbeitet und das verarbeitete Signal ausgegeben. Das passt zur Verwendung mit E-/A-Umleitung und/oder Befehlspipelines (Abbildung 13).

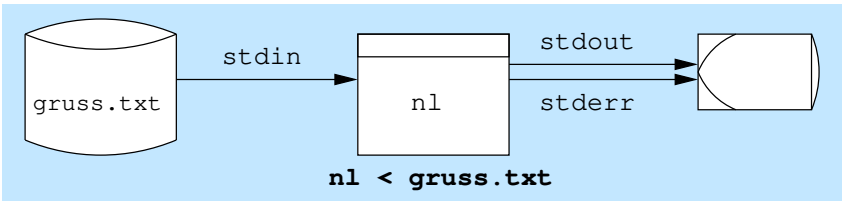

Abbildung 13: Start von `nl` ohne Dateinamen, aber mit Eingabeumleitung

Terminal

```
schueler@debian964:~$ nl < gruss.txt
     1  Guten Morgen
     2  Guten Abend
```

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

Tabelle 1: Filterprogramme

Tabelle 1 gibt einen Überblick über oft benutzte Filterprogramme.

Man sieht, dass das oben genannte Schema nicht auf jedes dieser Hilfsprogramme passt [3]. Die genannten Filterprogramme lassen sich in der Regel problemlos miteinander kombinieren. Wenn man z. B. ein Verzeichnis nach Dateiendungen sortiert (seitenweise) anzeigen will, kann man es so machen:

Terminal
```
schueler@debian964:~$ ls -l | rev | sort | rev | less
```

1.6.11 Mehrere Befehle in einer Zeile: Nacheinander

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 14). Erst dann, wenn der erste Befehl abgearbeitet ist, wird der zweite begonnen, wie man hier an der Ausgabe sieht:

Terminal
```
schueler@debian964:~$ hostname;uname
debian964
linux
```

Das Semikolon ersetzt hier einfach das Zeilenende.

[3] So arbeitet more absichtlich nicht mit Eingabe von Befehlen über eine 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.

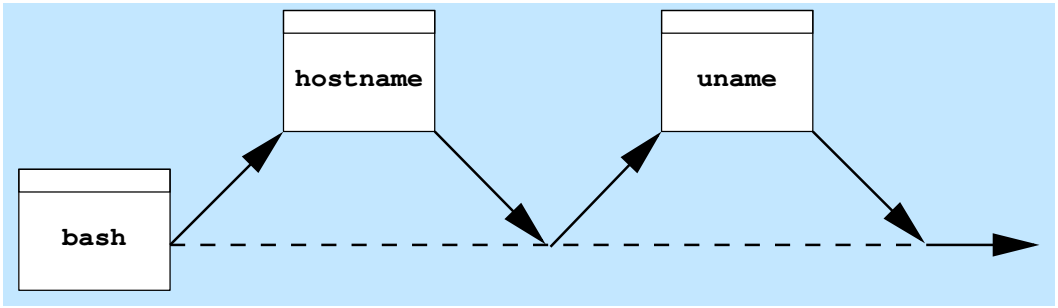

Abbildung 14: Befehlssequenz

1.6.12 Mehrere Befehle in einer Zeile: Bei Erfolg

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:

Terminal
```
schueler@debian964:~$ ls -l a.txt
ls: Zugriff auf a.txt nicht möglich: Datei oder Verz. nicht gefunden
schueler@debian964:~$ echo $?
2
```

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

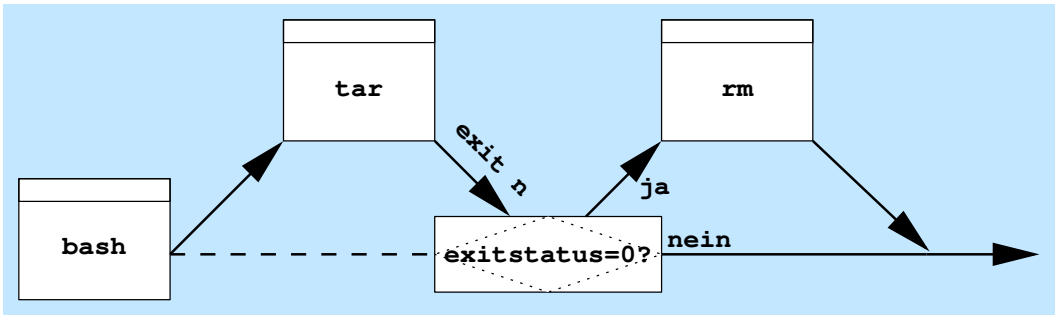

Abbildung 15: Befehlssequenz mit Und-Verknüpfung

Mit der Befehlszeile `befehl1 && befehl2` kann man diesen Exitcode ausnutzen: Hier wird `befehl2` nur dann ausgeführt, wenn `befehl1` erfolgreich war, also einen Exitcode gleich null ergab (Abbildung 15). Hier ist ein übliches Beispiel:

Terminal
```
schueler@debian964:~$ 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.

1.6.13 Mehrere Befehle in einer Zeile: Bei Fehlschlag

Mit der Befehlszeile `befehl1 || befehl2` ist es umgekehrt. Hier wird `befehl2` nur dann ausgeführt, wenn `befehl1` *nicht* erfolgreich war, also einen Rückgabewert ungleich null ergab (Abbildung 16). Ein Beispiel:

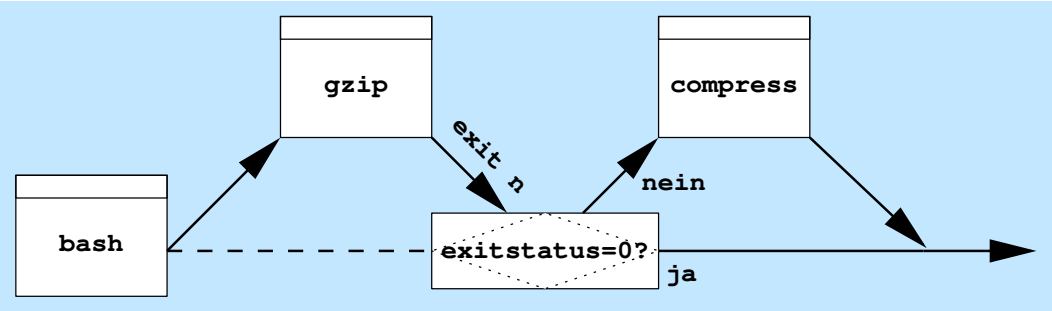

Abbildung 16: Befehlssequenz mit Oder-Verknüpfung

Terminal
```
schueler@debian964:~$ 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.

1.6.14 Mehrere Befehle in einer Zeile: Parallel

Wenn ein Befehl keine Tastatureingabe benötigt, kann man ihn auch parallel mit einem anderen Befehl starten. Es kann immer nur ein Befehl mit der Tastatur verbunden sein; der andere läuft im Hintergrund. Der Operator `&`, mit dem man einen Befehl im Hintergrund startet, dient genauso als Trenner zwischen Befehlen wie das Semikolon:

Terminal
```
schueler@debian964:~$ tar -cf s1.tar texte & tar -cf s2.tar daten
```

Nun läuft der linke `tar`-Befehl im Hintergrund; der rechte wird im Vordergrund gestartet.