

# Mit Pipelines Erstaunliches leisten

In diesem Kapitel erledigen wir verschiedene relativ einfache Textverarbeitungsaufgaben. Das Interessante an all den Beispielen hier ist, dass es sich um Skripten handelt, die aus einfachen Pipelines aufgebaut sind: Ketten, bei denen ein Befehl auf den anderen folgt. Dennoch erfüllt jeder eine wichtige Aufgabe.

Wenn Sie ein Textverarbeitungsproblem in Unix angehen, ist es wichtig, an die Unix-Philosophie zu denken: Fragen Sie sich, wie das Problem in einfachere Aufgaben zerlegt werden kann, für die es jeweils bereits ein Werkzeug gibt oder für die Sie mit wenigen Zeilen eines Shell-Programms oder mit einer Skriptsprache schnell selbst eines schreiben können.

## 5.1 Daten aus strukturierten Textdateien extrahieren

Die meisten administrativen Dateien in Unix sind einfache Textdateien, die Sie ohne besondere dateispezifische Werkzeuge bearbeiten, drucken und lesen können. Viele von ihnen befinden sich im Standardverzeichnis `/etc`. Verbreitete Beispiele sind die Passwort- und Gruppendateien (`passwd` und `group`), die Mount-Tabelle des Dateisystems (`fstab` oder `vfstab`), die Hosts-Datei (`hosts`), die vorgegebene Shell-Startdatei (`profile`) sowie die Shell-Skripten zum Starten und Herunterfahren des Systems, die in den Unterverzeichnisbäumen `rc0.d`, `rc1.d` usw. bis `rc6.d` gespeichert sind. (Es können auch noch andere Verzeichnisse vorhanden sein.)

Die Dateiformate werden traditionell in Abschnitt 5 des Unix-Handbuchs dokumentiert, so dass der Befehl `man 5 passwd` Informationen über die Struktur von `/etc/passwd` liefert.<sup>1</sup>

Trotz ihres Namens muss die Passwortdatei immer öffentlich lesbar sein. Vielleicht hätte man sie einfach als Benutzerdatei bezeichnen sollen, da sie grundlegende Informationen über alle Benutzerzugänge auf dem System enthält. Diese Informationen sind auf jeweils einer eigenen Zeile pro Zugang zusammengefasst, wobei die einzelnen Felder durch Dop-

---

<sup>1</sup> Auf einigen Systemen finden Sie die Dateiformate in Abschnitt 7; das heißt, Sie müssen stattdessen `man 7 passwd` benutzen.

pelpunkte getrennt sind. Wir haben das Format der Datei in »Konventionen für Textdateien« [3.3.1] beschrieben. Hier sind einige typische Einträge:

```
jones:*:32713:899:Adrian W. Jones/OSD211/555-0123:/home/jones:/bin/ksh
dorothy:*:123:30:Dorothy Gale/KNS321/555-0044:/home/dorothy:/bin/bash
toto:*:1027:18:Toto Gale/KNS322/555-0045:/home/toto:/bin/tcsh
ben:*:301:10:Ben Franklin/OSD212/555-0022:/home/ben:/bin/bash
jhancock:*:1457:57:John Hancock/SIG435/555-0099:/home/jhancock:/bin/bash
betsy:*:110:20:Betsy Ross/BMD17/555-0033:/home/betsy:/bin/ksh
tj:*:60:33:Thomas Jefferson/BMD19/555-0095:/home/tj:/bin/bash
george:*:692:42:George Washington/BST999/555-0001:/home/george:/bin/tcsh
```

Zur Erinnerung, die sieben Felder eines Eintrags in der Passwortdatei sind:

1. Der Benutzername
2. Das verschlüsselte Passwort oder ein Hinweis darauf, dass das Passwort in einer anderen Datei gespeichert ist
3. Die numerische Benutzer-ID
4. Die numerische Gruppen-ID
5. Der persönliche Name des Benutzers und möglicherweise weitere relevante Daten (Zimmernummer, Telefonnummer usw.)
6. Das Home-Verzeichnis
7. Die Login-Shell

Alle Felder bis auf eines haben eine Bedeutung für die verschiedenen Unix-Programme. Die Ausnahme bildet das fünfte Feld, das üblicherweise Benutzerinformationen enthält, die nur für Leser an dem jeweiligen Standort relevant sind. Früher wurde es als `gecos`-Feld bezeichnet, da es in den 70er Jahren an den Bell Labs hinzugefügt wurde, als Unix-Systeme mit anderen Computern kommunizieren mussten, auf denen das General Electric Comprehensive Operating System lief und zusätzliche Informationen über den Unix-Benutzer für dieses System benötigt wurden. Heutzutage wird dieses Feld an den meisten Standorten dafür verwendet, um den Namen des Benutzers aufzuzeichnen, weshalb wir es einfach als Namensfeld bezeichnen.

Für unser Beispiel nehmen wir an, dass am lokalen Standort weitere Informationen im Namensfeld aufgezeichnet werden: ein Identifier für das Gebäude und die Zimmernummer (OSD211 im ersten Beispieleintrag) und eine Telefonnummer (555-0123), die vom Namen durch Schrägstriche getrennt sind.

Mit einer solchen Datei können wir Software für ein Büroverzeichnis schreiben. Auf diese Weise müsste nur eine einzige Datei, nämlich `/etc/passwd`, auf dem neuesten Stand gehalten werden. Abgeleitete Dateien können erzeugt werden, wenn die Master-Datei geändert wurde, oder – viel besser – mit einem cron-Job, der in geeigneten Zeitintervallen ausgeführt wird. (Wir werden cron im Abschnitt »crontab: Zu bestimmten Zeiten erneut ausführen« [13.6.4] besprechen.)

In unserem ersten Versuch ist das Büroverzeichnis eine einfache Textdatei mit folgenden Einträgen:

Franklin, Ben	•OSD212•555-0022
Gale, Dorothy	•KNS321•555-0044
...	

Dabei repräsentiert • ein ASCII-Tabulatorzeichen. Wir geben den Namen in einer normalen Verzeichnisreihenfolge an (Familiename zuerst) und füllen das Namensfeld mit Leerzeichen auf, bis eine bequeme feste Länge erreicht ist. Der Zimmernummer und der Telefonnummer stellen wir als Präfix Tabulatorzeichen voran, um eine sinnvolle Struktur zu erreichen, die von anderen Werkzeugen ausgenutzt werden kann.

Skriptsprachen wie `awk` sind dazu gedacht, solche Aufgaben zu vereinfachen, da sie eine automatisierte Eingabeverarbeitung und Aufspaltung der eingegebenen Datensätze in Felder ermöglichen. Wir könnten also die Umwandlung vollständig mit Hilfe einer solchen Sprache erledigen. Wir wollen allerdings zeigen, wie Sie das mit anderen Unix-Werkzeugen erreichen.

Wir müssen aus jeder Zeile der Passwortdatei Feld fünf extrahieren, es in drei Unterfelder aufteilen, die Namen im ersten Unterfeld neu anordnen und dann eine Zeile des Büroverzeichnisses an einen Sortierprozess übergeben.

`awk` und `cut` sind die passenden Werkzeuge für die Extraktion des Feldes:

```
... | awk -F: '{ print $5 }' | ...
... | cut -d: -f5 | ...
```

Etwas verkompliziert wird die ganze Geschichte durch die Tatsache, dass wir zwei Feldverarbeitungsaufgaben haben, die wir aus Gründen der Einfachheit getrennt halten wollen, deren Ausgaben wir aber kombinieren müssen, um einen Verzeichniseintrag zu erhalten. Der Befehl `join` eignet sich genau dafür: Er erwartet zwei Eingabefelder, die jeweils durch einen gemeinsamen eindeutigen Schlüsselwert geordnet sind, und fasst Zeilen, die den gleichen Schlüssel aufweisen, zu einer einzigen Ausgabezeile zusammen. Der Benutzer kann angeben, welche Felder ausgegeben werden.

Da unsere Verzeichniseinträge drei Felder enthalten, müssen wir, um `join` benutzen zu können, drei Zwischendateien anlegen, die folgende durch Doppelpunkte getrennte Paare enthalten: *schlüssel:person*, *schlüssel:zimmer* und *schlüssel:telefon*, mit jeweils einem Paar pro Zeile. Dabei kann es sich um temporäre Dateien handeln, da sie automatisch aus der Passwortdatei abgeleitet werden.

Welchen Schlüssel sollten wir verwenden? Er muss lediglich eindeutig sein, so dass wir die Datensatznummer aus der ursprünglichen Passwortdatei nehmen könnten. In diesem Fall kann es aber auch der Benutzername sein, da wir wissen, dass die Benutzernamen in der Passwortdatei ebenfalls eindeutig sind. Für Menschen sind die Benutzernamen außerdem sinnvoller als Nummern. Falls wir später beschließen, unser Verzeichnis um weitere Informationen zu erweitern, wie etwa die Arbeitsbezeichnung des Benutzers, können wir eine weitere nichttemporäre Datei mit dem Paar *schlüssel:arbeitsbezeichnung* erzeugen und zu den Verarbeitungsstufen hinzufügen.

Anstatt die Eingabe- und Ausgabedateinamen fest in unser Programm zu codieren, ist es flexibler, das Programm als *Filter* zu schreiben, so dass es die Standardeingabe liest und auf die Standardausgabe schreibt. Für Befehle, die selten verwendet werden, ist es ratsam, beschreibende anstelle von kurzen und kryptischen Namen zu verwenden. Wir beginnen unser Shell-Programm deshalb folgendermaßen:

```
#!/bin/sh
# Filtert einen Eingabestrom, der wie /etc/passwd formatiert ist,
# und gibt ein Büroverzeichnis aus, das aus diesen Daten abgeleitet wurde.
#
# Verwendung:
#   passwd-to-directory < /etc/passwd > office-directory-file
#   ypcat passwd | passwd-to-directory > office-directory-file
#   niscat passwd.org_dir | passwd-to-directory > office-directory-file
```

Da die Passwortdatei öffentlich lesbar ist, sind alle daraus abgeleiteten Daten ebenfalls öffentlich. Es ist also nicht notwendig, den Zugriff auf die Zwischendateien unseres Programms zu beschränken. Da wir jedoch alle manchmal mit geheimen Daten zu tun haben, sollten Sie sich es beim Programmieren angewöhnen, den Dateizugriff nur solchen Benutzern oder Prozessen zu gewähren, die ihn auch wirklich brauchen. Wir setzen daher als erste Aktion unseres Programms die `umask` (siehe »Vorgegebene Berechtigungen« in Anhang B) zurück:

```
umask 077 Beschränkt den Zugriff auf die temporäre Datei auf uns
```

Aus Gründen der Übersichtlichkeit und für eine einfachere Fehlersuche ist es hilfreich, wenn man bei den Namen der temporären Dateien einige Gemeinsamkeiten hat; außerdem vermeidet man es auf diese Weise, das aktuelle Verzeichnis mit ihnen zu vermüllen: Wir setzen das Präfix `/tmp/pd.` vor die Namen. Um uns vor Namensüberschneidungen zu schützen, falls mehrere Instanzen unseres Programms gleichzeitig laufen, müssen die Namen darüber hinaus eindeutig sein: Die Prozessnummer, verfügbar in der Shell-Variablen `$$`, bietet ein Suffix zur Unterscheidung. (Diese Verwendung von `$$` wird in Kapitel 10 ausführlicher beschrieben.) Wir definieren deshalb folgende Shell-Variablen, die unsere temporären Dateien repräsentieren sollen:

```
PERSON=/tmp/pd.key.person.$$ Eindeutige temporäre Dateinamen
OFFICE=/tmp/pd.key.office.$$
TELEPHONE=/tmp/pd.key.telephone.$$
USER=/tmp/pd.key.user.$$
```

Wenn die Aufgabe beendet wird, egal ob auf normale oder auf ungewöhnliche Weise, sollen die temporären Dateien gelöscht werden. Deshalb benutzen wir den Befehl `trap`:

```
trap "exit 1" HUP INT PIPE QUIT TERM
trap "rm -f $PERSON $OFFICE $TELEPHONE $USER" EXIT
```

Während der Entwicklung können wir das zweite `trap` auskommentieren, wodurch wir die temporären Dateien für eine nachfolgende Untersuchung aufbewahren. (Der `trap`-Befehl wird in »Prozesssignale abfangen« [13.3.2] beschrieben. Im Moment reicht es, wenn Sie verstehen, dass der `trap`-Befehl automatisch `rm` mit den angegebenen Argumenten aufruft, wenn das Skript beendet wird.)

Wir brauchen die Felder eins und fünf mehrmals. Sobald wir sie haben, benötigen wir den Eingabestrom von der Standardeingabe nicht mehr, sondern können damit beginnen, die Felder in eine temporäre Datei zu extrahieren:

```
awk -F: '{ print $1 ":" $5 }' > $USER Das liest die Standardeingabe
```

Das Paar *schlüssel:person* kommt zuerst. Das erledigen wir mit einem aus zwei Schritten bestehenden *sed*-Programm, gefolgt von einer einfachen Zeilensortierung. Der *sort*-Befehl wird in »Text sortieren« [4.1] ausführlich besprochen.

```
sed -e 's=/.*= ' \
    -e 's=^\([^:]*\):\(.*\) \([^ ]*\)=\1:\3, \2=' <$USER | sort >$PERSON
```

Das Skript verwendet `=` als Trennzeichen für den *s*-Befehl von *sed*, da sowohl Slashes als auch Doppelpunkte in den Daten auftauchen. Beim ersten Bearbeitungsschritt wird alles vom ersten Slash bis zum Ende der Zeile entfernt, wodurch sich eine solche Zeile:

```
jones:Adrian W. Jones/OSD211/555-0123 Eingabezeile
```

folgendermaßen verkürzt:

```
jones:Adrian W. Jones Ergebnis des ersten Bearbeitungsschritts
```

Der zweite Bearbeitungsschritt ist komplexer – es werden drei Untermuster im Datensatz gefiltert. Der erste Teil, `^\([^:]*\)`, filtert das Benutzernamenfeld (z. B. `jones`). Der zweite Teil, `\(.*\)□`, filtert Text bis zu einem Leerzeichen (z. B. `Adrian□W.□`; das `□` steht für ein Leerzeichen). Der letzte Teil, `\([^□]*\)`, filtert den verbleibenden nichtleeren Text in dem Datensatz (z. B. `Jones`). Der Ersetzungstext ordnet die Treffer neu an und erzeugt auf diese Weise etwas wie `Jones,□Adrian W.` Das Ergebnis dieses einzelnen *sed*-Befehls ist die gewünschte Neuordnung:

```
jones:Jones, Adrian W. Ausgegebenes Ergebnis des zweiten  
Bearbeitungsschritts
```

Als Nächstes erstellen wir die Datei mit dem *schlüssel:büro*-Paar:

```
sed -e 's=^\([^:]*\):[/]*\[^\(.*\)\/.*$=\1:\2=' <$USER | sort >$OFFICE
```

Das Ergebnis ist eine Liste der Benutzer und Büros:

```
jones:OSD211
```

Die Erzeugung der Datei mit dem *schlüssel:telefon*-Paar verläuft ähnlich; wir müssen lediglich die Musterfilterung anpassen:

```
sed -e 's=^\([^:]*\):[/]*\[^\(.*\)\/\1:\2=' <$USER | sort >$TELEPHONE
```

Auf dieser Stufe haben wir drei separate Dateien, die jeweils sortiert sind. Jede Datei besteht aus dem Schlüssel (dem Benutzernamen), einem Doppelpunkt und den speziellen Daten (persönlicher Name, Zimmernummer, Telefonnummer). Der Inhalt der Datei `$PERSON` sieht so aus:

```
ben:Franklin, Ben
betsy:Ross, Betsy
...
```

Die Datei \$OFFICE enthält Benutzernamen und Bürodaten:

```
ben:OSD212
betsy:BMD17
...
```

In der Datei \$TELEPHONE stehen Benutzernamen und Telefonnummern:

```
ben:555-0022
betsy:555-0033
...
```

Standardmäßig gibt `join` den gemeinsamen Schlüssel und dann die verbleibenden Felder der Zeile aus der ersten Datei, gefolgt von den verbleibenden Feldern der Zeile aus der zweiten Datei aus. Der gemeinsame Schlüssel steht laut Vorgabe im ersten Feld; dies kann allerdings mit Hilfe einer Kommandozeilenoption geändert werden: Wir brauchen diese Funktion hier nicht. Normalerweise werden die Felder für `join` durch Leerzeichen getrennt, wir ändern jedoch das Trennzeichen mit der Option `-t`: Wir benutzen sie als `-t:`.

Die Operationen zum Zusammenfügen werden mit einer fünfstufigen Pipeline erledigt:

1. Kombinieren der persönlichen Informationen und der Büroinformationen:

```
join -t: $PERSON $OFFICE | ...
```

Die Ergebnisse dieser Operation, die zur Eingabe für die nächste Stufe werden, sehen so aus:

```
ben:Franklin, Ben:OSD212
betsy:Ross, Betsy:BMD17
...
```

2. Hinzufügen der Telefonnummer:

```
... | join -t: - $TELEPHONE | ...
```

Die Ergebnisse dieser Operation, die zur Eingabe für die nächste Stufe werden, sehen so aus:

```
ben:Franklin, Ben:OSD212:555-0022
betsy:Ross, Betsy:BMD17:555-0033
...
```

3. Entfernen des Schlüssels (der im ersten Feld steht), da er nicht mehr benötigt wird. Am einfachsten geht das mit `cut` und einem Bereich, der besagt »Benutze die Felder zwei bis zum Ende«, etwa so:

```
... | cut -d: -f 2- | ...
```

Die Ergebnisse dieser Operation, die zur Eingabe für die nächste Stufe werden, sehen so aus:

```
Franklin, Ben:OSD212:555-0022
Ross, Betsy:BMD17:555-0033
...
```

4. Neusortieren der Daten. Die Daten waren zuvor anhand des Login-Namens sortiert; nun sollen sie aber nach dem Nachnamen sortiert werden. Das erledigen Sie mit `sort`:

```
... | sort -t: -k1,1 -k2,2 -k3,3 | ...
```

Dieser Befehl verwendet einen Doppelpunkt, um die Felder zu trennen, wobei auf den Feldern 1, 2 und 3 sortiert wird. Die Ergebnisse dieser Operation, die zur Eingabe für die nächste Stufe werden, sehen so aus:

```
Franklin, Ben:OSD212:555-0022
Gale, Dorothy:KNS321:555-0044
...
```

5. Schließlich wird die Ausgabe mit Hilfe der `awk`-Anweisung `printf` umformatiert, um die einzelnen Felder mittels Tabulatoren zu trennen. Dazu kommt folgender Befehl zum Einsatz:

```
... | awk -F: '{ printf("%-39s\t%s\t%s\n", $1, $2, $3) }'
```

Aus Gründen der Flexibilität und guten Wartbarkeit sollte man sich die Formatierung immer bis zum Ende aufheben. Bis zu dieser Stelle hatten wir einfach nur Textstrings beliebiger Länge.

Hier ist die vollständige Pipeline:

```
join -t: $PERSON $OFFICE |
  join -t: - $TELEPHONE |
  cut -d: -f 2- |
  sort -t: -k1,1 -k2,2 -k3,3 |
  awk -F: '{ printf("%-39s\t%s\t%s\n", $1, $2, $3) }'
```

Die hier verwendete `awk`-Anweisung `printf` ist vergleichbar dem Shell-Befehl `printf`, so dass ihre Bedeutung klar sein sollte: linksbündige Ausgabe des ersten durch einen Doppelpunkt abgetrennten Feldes in einem 39-Zeichen-Feld, gefolgt von einem Tabulatorzeichen, dem zweiten Feld, einem weiteren Tabulator und dem dritten Feld. Hier sind die vollständigen Ergebnisse:

Franklin, Ben	•OSD212•555-0022
Gale, Dorothy	•KNS321•555-0044
Gale, Toto	•KNS322•555-0045
Hancock, John	•SIG435•555-0099
Jefferson, Thomas	•BMD19•555-0095
Jones, Adrian W.	•OSD211•555-0123
Ross, Betsy	•BMD17•555-0033
Washington, George	•BST999•555-0001

Das ist schon alles! Unser ganzes Skript ist ohne Kommentare kaum länger als 20 Zeilen und umfasst fünf Hauptverarbeitungsschritte. In Beispiel 5-1 sehen Sie es noch einmal komplett.

*Beispiel 5-1: Anlegen eines Büroverzeichnisses*

```
#!/bin/sh
# Filtert einen Eingabestrom, der wie /etc/passwd formatiert ist,
# und gibt ein Büroverzeichnis aus, das aus diesen Daten abgeleitet wurde.
#
# Verwendung:
```

### Beispiel 5-1: Anlegen eines Büroverzeichnisses (Fortsetzung)

```
#      passwd-to-directory < /etc/passwd > office-directory-file
#      ypcat passwd | passwd-to-directory > office-directory-file
#      niscat passwd.org_dir | passwd-to-directory > office-directory-file

umask 077

PERSON=/tmp/pd.key.person.$$
OFFICE=/tmp/pd.key.office.$$
TELEPHONE=/tmp/pd.key.telephone.$$
USER=/tmp/pd.key.user.$$

trap "exit 1"                                HUP INT PIPE QUIT TERM
trap "rm -f $PERSON $OFFICE $TELEPHONE $USER" EXIT

awk -F: '{ print $1 ":" $5 }' > $USER

sed -e 's=/.*= ' \
    -e 's=^\([^:]*\):\(.*\) \([^ ]*\)=\1:\3, \2=' < $USER | sort > $PERSON

sed -e 's=^\([^:]*\):[/]*[/\([^/]*\)/.*$=\1:\2=' < $USER | sort > $OFFICE

sed -e 's=^\([^:]*\):[/]*[/\([^/]*\)=\1:\2=' < $USER | sort > $TELEPHONE

join -t: $PERSON $OFFICE |
  join -t: - $TELEPHONE |
  cut -d: -f 2- |
  sort -t: -k1,1 -k2,2 -k3,3 |
  awk -F: '{ printf("%-39s\t%s\t%s\n", $1, $2, $3) }'
```

Die wahre Stärke von Shell-Skripten zeigt sich von selbst, wenn wir das Skript so abändern wollen, dass es eine etwas andere Aufgabe erledigt, wie etwa das Einfügen der Arbeitsbezeichnung aus einer getrennt gepflegten *schlüssel:arbeitsbezeichnung*-Datei. Wir müssen lediglich die letzte Pipeline entsprechend anpassen:

```
join -t: $PERSON /etc/passwd.job-title |           Zusätzliches join mit der Arbeitsbezeichnung
  join -t: - $OFFICE |
  join -t: - $TELEPHONE |
  cut -d: -f 2- |
  sort -t: -k1,1 -k3,3 -k4,4 |           Modifizieren des sort-Befehls
  awk -F: '{ printf("%-39s\t%-23s\t%s\t%s\n",
    $1, $2, $3, $4) }'           und des Formatierungsbefehls
```

Der Gesamtaufwand für das zusätzliche Verzeichnisfeld besteht in einem weiteren join, einer Änderung in den sort-Feldern und in einer kleinen Änderung am abschließenden awk-Formatierungsbefehl.

Da wir so vorsichtig waren, besondere Feldtrenner in unserer Ausgabe zu behalten, können wir ganz einfach nützliche alternative Verzeichnisse vorbereiten:

```
passwd-to-directory < /etc/passwd | sort -t'•' -k2,2 > dir.by-office
passwd-to-directory < /etc/passwd | sort -t'•' -k3,3 > dir.by-telephone
```

Wie gewöhnlich repräsentiert • ein ASCII-Tabulatorzeichen.

Eine wesentliche Annahme unseres Programms ist, dass es einen *eindeutigen Schlüssel* für jeden Datensatz gibt. Mit Hilfe dieses eindeutigen Schlüssels können getrennte Sichten auf die Daten als *schlüssel:wert*-Paare in Dateien vorgehalten werden. Hier war der Schlüssel ein Unix-Benutzername, in größeren Kontexten könnte es sich jedoch um eine ISBN-Nummer, eine Kreditkartennummer, eine Personalnummer, eine Sozialversicherungsnummer, eine Teilenummer, eine Matrikelnummer usw. handeln. Jetzt wissen Sie, weshalb uns so viele Nummern zugewiesen werden! Sie können auch erkennen, dass nicht unbedingt Nummern erforderlich sind: Hauptsache, es sind eindeutige Textstrings.

## Ein Exkurs über Datenbanken

Die meisten kommerziellen Datenbanken sind heutzutage als *relationale Datenbanken* aufgebaut: Die Daten sind als *schlüssel:wert*-Paare verfügbar, es werden *join*-Operationen eingesetzt, um mehrspaltige Tabellen zu konstruieren, die Sichten auf ausgewählte Teilmengen der Daten bieten. Relationale Datenbanken wurden im Jahr 1970 zum ersten Mal von E. F. Codd vorgeschlagen,<sup>a</sup> der sie aktiv angepriesen hat, obwohl die damalige Datenbankindustrie behauptete, dass man solche Datenbanken nicht effizient implementieren könne. Glücklicherweise fanden schlaue Programmierer schnell heraus, wie man das Effizienzproblem lösen konnte. Codds Arbeit ist so wichtig, dass er 1981 den angesehenen ACM Turing Award erhielt, in der Informatik dem Nobelpreis vergleichbar.

Heute gibt es mehrere ISO-Standards für die *Structured Query Language* (SQL), wodurch ein herstellerunabhängiger Datenbankzugriff möglich ist. Eine der wichtigsten SQL-Operationen ist *join*. Es wurden Hunderte von Büchern über SQL veröffentlicht; um mehr zu lernen, nehmen Sie ein allgemeines Werk wie *SQL in a Nutshell*.<sup>b</sup> Unsere einfache Büroverzeichnisaufgabe enthält daher eine wichtige Lektion über das zentrale Konzept moderner relationaler Datenbanken, und Unix-Software-Werkzeuge können beim Vorbereiten der Eingaben für Datenbanken und beim Verarbeiten ihrer Ausgaben außerordentlich nützlich sein.

a E. F. Codd, *A Relational Model of Data for Large Shared Data Banks*, Communications of the ACM, 13(6) 377–387, Juni (1970) und *Relational Database: A Practical Foundation for Productivity*, Communications of the ACM, 25(2) 109–117, Februar (1982) (Turing Award-Vortrag).

b Von Kevin Kline, Daniel Kline und Brand Hunt, O'Reilly Verlag 2005, ISBN 3-89721-340-0. Unter <http://www.math.utah.edu/pub/tex/bib/sqlbooks.html> finden Sie eine ausführliche Liste mit SQL-Büchern.

## 5.2 Strukturierte Daten für das Web

Die außerordentliche Beliebtheit des World Wide Web lässt es wünschenswert erscheinen, Daten wie das im letzten Abschnitt entwickelte Büroverzeichnis in einer Form zu präsentieren, die ein bisschen netter ist als unsere einfache Textdatei.

Web-Dateien werden meist in einer Auszeichnungssprache namens *HyperText Markup Language* (HTML) geschrieben. Das ist eine Familie von Sprachen, die spezielle Ausprägungen der *Standard Generalized Markup Language* (SGML) bilden. SGML wurde in mehreren ISO-Standards seit 1986 definiert. Das Manuskript für dieses Buch wurde in

DocBook/XML geschrieben, ebenfalls eine besondere Form von SGML. Eine vollständige Beschreibung von HTML finden Sie in *HTML & XHTML: Das umfassende Referenzwerk* (O'Reilly).<sup>2</sup>

Für die Zwecke dieses Abschnitts benötigen wir nur eine winzige Teilmenge von HTML, die wir hier in einer kleinen Anleitung vorstellen. Falls Sie bereits mit HTML vertraut sind, überspringen Sie einfach die nächsten ein oder zwei Seiten.

Hier ist eine minimale standardkonforme HTML-Datei, die von einem wunderbaren Werkzeug erzeugt wurde, das einer von uns geschrieben hat:<sup>3</sup>

```
$ echo Hello, world. | html-pretty
<!-- -*-html-*- -->
<!-- Prettyprinted by html-pretty flex version 1.01 [25-Aug-2001] -->
<!-- on Wed Jan  8 12:12:42 2003 -->
<!-- for Adrian W. Jones (jones@example.com) -->

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<HTML>
  <HEAD>
    <TITLE>
      <!-- Geben Sie hier einen aussagekräftigen Titel an -->
    </TITLE>
    <!-- Geben Sie hier bitte eine korrekte E-Mail-Adresse an -->
    <LINK REV="made" HREF="mailto:jones@example.com">
  </HEAD>
  <BODY>
    Hello, world.
  </BODY>
</HTML>
```

Folgendes ist zu dieser HTML-Ausgabe anzumerken:

- HTML-Kommentare werden in `<!--` und `-->` eingeschlossen.
- Besondere Prozessor-Befehle sind in `<!` und `>` eingeschlossen: Hier teilt der Befehl `DOCTYPE` einem SGML-Parser mit, welches der Dokumenttyp ist und wo dessen Grammatikdatei zu finden ist.
- Die Auszeichnungen werden durch Wörter angegeben, die in spitzen Klammern stehen, so genannte *Tags*. In HTML ist die Groß- und Kleinschreibung der Tag-Namen *nicht* wichtig: `html-pretty` setzt die Tag-Namen normalerweise in Großbuchstaben, damit sie sich besser abheben.
- Auszeichnungsumgebungen bestehen aus einem Anfangs-Tag, `<NAME>`, und einem End-Tag, `</NAME>`. Bei vielen Tags können die Umgebungen entsprechend den Regeln, die in der HTML-Grammatik definiert sind, ineinander geschachtelt werden.

---

2 Zusätzlich zu diesem Buch (das in der Bibliographie aufgeführt ist) finden Sie Hunderte von Büchern über SGML und seine Abkömmlinge unter <http://www.math.utah.edu/pub/tex/bib/sgml.html> und <http://www.math.utah.edu/pub/tex/bib/sgml2000.html>.

3 Verfügbar unter <http://www.math.utah.edu/pub/sgml/>.

- Ein HTML-Dokument ist als ein HTML-Objekt strukturiert, das ein HEAD- und ein BODY-Objekt enthält.
- Innerhalb des HEAD definiert ein TITLE-Objekt den Dokumenttitel, den Webbrowser in der Titelleiste des Fensters sowie in Lesezeichenlisten anzeigen. Das ebenfalls im HEAD befindliche LINK-Objekt enthält im Allgemeinen Informationen über den Webseitenbetreiber.
- Der sichtbare Teil des Dokuments, den die Browser zeigen, ist der Inhalt des BODY.
- Whitespace hat außerhalb geschützter Strings keine Bedeutung. Wir können also ebenso wie der HTML-Prettyprinter großzügig horizontale und vertikale Abstände einsetzen, um die Struktur zu betonen.
- Alles andere ist einfach druckbarer ASCII-Text, mit drei Ausnahmen. Literal verwendete spitze Klammern müssen mit Hilfe besonderer Codierungen ausgedrückt werden, so genannten *Entities*, die aus einem Ampersand, einem Identifier und einem Semikolon bestehen: `&lt;` und `&gt;`. Da das Ampersand ein Entity startet, verfügt es über einen eigenen literalen *Entity*-Namen: `&amp;`. HTML unterstützt ein bescheidenes Repertoire von Entities für akzentuierte Zeichen, das die meisten westeuropäischen Sprachen abdeckt, so dass wir beispielsweise `café`; `du bon goût` schreiben können, um `café du bon goût` zu erhalten.
- Auch wenn es in unserem Beispiel nicht zu sehen war: Änderungen des Schriftstils werden in HTML mit den Umgebungen `B` (fett), `EM` (Betonung), `I` (kursiv), `STRONG` (extra fett) und `TT` (Nichtproportionalschrift) erreicht: Schreiben Sie `<B>fett gedruckter Passus</B>`, um **fett gedruckter Passus** zu erhalten.

Um unser Büroverzeichnis in richtiges HTML zu konvertieren, benötigen wir nur noch eine weitere Information: wie eine Tabelle formatiert wird, da unser Verzeichnis nichts anderes ist und wir nicht die Schreibmaschinenschrift bemühen wollen, um eine bestimmte Ausrichtung im Browser-Fenster zu erhalten.

In HTML 3.0 und später besteht eine Tabelle aus einer TABLE-Umgebung, in der sich Zeilen, jeweils in einer TR-Umgebung, befinden. In den einzelnen Zeilen gibt es Zellen, die so genannten Tabellendaten, jeweils in einer TD-Umgebung. Beachten Sie, dass Spalten mit Daten keine besonderen Auszeichnungen erhalten: Eine Datenspalte ist einfach die Gruppe von Zellen, die in allen Zeilen der Tabelle an der gleichen Zeilenposition stehen. Glücklicherweise müssen wir die Anzahl der Zeilen und Spalten nicht im Voraus deklarieren. Die Aufgabe des Browsers oder Formatierers besteht darin, alle Zellen zu sammeln, in jeder Spalte die breiteste Zelle zu ermitteln und die Tabelle dann so zu formatieren, dass die Spalten gerade breit genug sind, um diese breiteste Zelle aufzunehmen.

Für unser Büroverzeichnisbeispiel benötigen wir nur drei Spalten, so dass unser Beispiel eintrag folgendermaßen ausgezeichnet werden könnte:

```
<TABLE>
  ...
  <TR>
    <TD>
```

```

        Jones, Adrian W.
    </TD>
    <TD>
        555-0123
    </TD>
    <TD>
        OSD211
    </TD>
</TR>
...
</TABLE>

```

Eine äquivalente, aber auch kompakte und schwer zu lesende Codierung könnte so aussehen:

```

<TABLE>
...
<TR><TD>Jones, Adrian W.</TD><TD>555-0123</TD><TD>OSD211</TD></TR>
...
</TABLE>

```

Da wir besondere Feldtrennzeichen in der Textversion des Büroverzeichnisses beibehalten, haben wir ausreichende Informationen, um die Zellen in den einzelnen Zeilen zu identifizieren. Und da Whitespace in HTML-Dateien (außer für Menschen) meist nicht wichtig ist, müssen wir nicht besonders darauf achten, dass die Tags hübsch ausgerichtet sind: Falls das später erforderlich sein sollte, kann `html-pretty` es perfekt erledigen. Unser Konvertierungsfilter umfasst dann drei Schritte:

1. Ausgabe des führenden Textbausteins am Anfang des Dokument-Body.
2. Einpacken der einzelnen Zeilen des Verzeichnisses in Tabellen-Tags.
3. Ausgabe des abschließenden Textbausteins.

Wir müssen nur eine kleine Änderung an unserem Minimalbeispiel vornehmen: Der `DOCTYPE`-Befehl muss auf eine spätere Grammatik aktualisiert werden, so dass er nun folgendermaßen aussieht:

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN//3.0">
```

Sie müssen sich das nicht merken: `html-pretty` besitzt Optionen, um die Ausgabe in einer der Standard-HTML-Grammatiken zu erzeugen, so dass Sie nur noch einen passenden `DOCTYPE`-Befehl aus dieser Ausgabe kopieren müssen.

Offensichtlich besteht die meiste Arbeit darin, einfach Textbausteine zu schreiben, aber das ist einfach, da wir den Text aus dem minimalen HTML-Beispiel kopieren können. Der einzige erforderliche Programmierschritt ist der mittlere, den wir mit Hilfe einiger Zeilen in `awk` ausführen könnten. Wir können diesen Schritt aber sogar mit noch weniger Arbeit ausführen, indem wir eine `sed`-Ersetzung mit zwei Bearbeitungsbefehlen einsetzen: einen, um die eingebetteten Tabulatortrennzeichen durch `</TD><TD>` zu ersetzen, und einen folgenden, um die gesamte Zeile in `<TR><TD>...</TD></TR>` einzubetten. Wir nehmen für den Augenblick an, dass im Verzeichnis keine akzentuierten Zeichen erforderlich sind. Allerdings könnten wir auch ganz einfach spitze Klammern und Ampersands im Eingabestrom

zulassen, indem wir am Anfang drei sed-Schritte hinzufügen. Das vollständige Programm zeigen wir in Beispiel 5-2.

*Beispiel 5-2: Konvertierung eines Büroverzeichnisses nach HTML*

```
#!/bin/sh
# Konvertiert eine Datei mit durch Tabulatoren getrennten Werten in standardkonformes HTML.
#
# Verwendung:
#   tsv-to-html < infile > outfile

cat << EOFILE                                Führender Textbaustein
<DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN//3.0">
<HTML>
  <HEAD>
    <TITLE>
      Büroverzeichnis
    </TITLE>
    <LINK REV="made" HREF="mailto:$USER@`hostname`">
  </HEAD>
  <BODY>
    <TABLE>
EOFILE

sed -e 's=&=\&amp;=g' \                       Konvertiert Sonderzeichen in Entities
    -e 's=<=\&lt;=g' \
    -e 's=>=\&gt;=g' \
    -e 's=\t=</TD><TD>=g' \                   Und gibt Tabellenauszeichnungen an
    -e 's=^.*$= <TR><TD>&</TD></TR>='

cat << EOFILE                                Abschließender Textbaustein
  </TABLE>
</BODY>
</HTML>
EOFILE
```

Die <<-Notation wird als *Hier-Dokument* bezeichnet. Sie wird in »Weitere Umleitungsoperatoren« [7.3.1] genauer erläutert. In Kürze: Die Shell liest alle Zeilen bis zu dem Trennzeichen, das auf << folgt (in diesem Fall EOFILE), führt Variablen- und Befehlersetzungen auf den enthaltenen Zeilen durch und übergibt die Ergebnisse als Standardeingabe an den Befehl.

Es gibt einen wichtigen Punkt bei dem Skript in Beispiel 5-2: Es ist unabhängig von der Anzahl der Spalten in der Tabelle! Das bedeutet, dass es dazu verwendet werden kann, *jede* Datei mit durch Tabulatoren getrennten Werten nach HTML zu konvertieren. Tabellenkalkulationen können Daten normalerweise in einem solchen Format speichern, so dass unser einfaches Werkzeug in der Lage ist, aus Tabellenkalkulationsdaten korrektes HTML zu erzeugen.

Wir haben sorgfältig darauf geachtet, in tsv-to-html die Struktur der Zwischenräume des originalen Büroverzeichnisses beizubehalten, da es dadurch einfacher wird, später wei-

tere Filter einzusetzen. Tatsächlich wurde `html-pretty` genau aus diesem Grund geschrieben: Die Standardisierung des Layouts der HTML-Auszeichnungen vereinfacht andere HTML-Werkzeuge radikal.

Wie würden wir die Konvertierung akzentuierter Zeichen in HTML-Entities durchführen? Wir *könnten* den `sed`-Befehl um zusätzliche Bearbeitungsschritte wie `-e 's=é=&eacute;;=g'` erweitern, allerdings gibt es etwa 100 Entities, um die man sich kümmern müsste, und wir benötigen wahrscheinlich ähnliche Ersetzungen, wenn wir andere Arten von Textdateien nach HTML konvertieren.

Es ist daher sinnvoll, diese Aufgabe an ein eigenes Programm zu delegieren, das wir später wiederverwenden können, entweder als Pipeline-Stufe nach dem `sed`-Befehl in Beispiel 5-2 oder als ein später angewendeter Filter. (Dies ist das Prinzip »Mache einen Umweg, um spezialisierte Werkzeuge zu erstellen« in Aktion.) Solch ein Programm ist nur eine lästige Aufstellung der Ersetzungsbefehle, und wir brauchen eines für jede der lokalen Textcodierungen, wie etwa die verschiedenen ISO 8859-*n*-Codeseiten, die im Abschnitt »Wie werden Dateien benannt?« in Anhang B erwähnt werden. Wir zeigen einen solchen Filter hier nicht vollständig, ein Fragment eines solchen Filters in Beispiel 5-3 vermittelt allerdings einen allgemeinen Eindruck. Leser, die so einen Filter brauchen, finden das vollständige Programm zum Verarbeiten der allgemeinen Fälle der westeuropäischen Zeichen in der ISO 8859-1-Codierung bei den Beispielprogrammen dieses Buches. Das Repertoire an HTML-Entities reicht für andere akzentuierte Zeichen nicht aus, aber da sich das World Wide Web in Richtung Unicode und XML anstelle von ASCII und HTML bewegt, wird dieses Problem auf eine andere Weise gelöst, nämlich indem die Beschränkungen der Zeichensätze verschwinden.

*Beispiel 5-3: Fragment des Programms iso8859-1-to-html*

```
#!/bin/sh
# Konvertiert einen Eingabestrom, der Zeichen in ISO 8859-1-Codierung
# aus dem Bereich 128..255 enthält, in die HTML-Äquivalente in ASCII.
# Die Zeichen 0..127 werden als normale ASCII-Zeichen bewahrt.
#
# Verwendung:
#     iso8859-1-to-html infile(s) >outfile

sed \
    -e 's= =\&nbsp;;=g' \
    -e 's=¡=\&iexcl;;=g' \
    -e 's=¢=\&cent;;=g' \
    -e 's=£=\&pound;;=g' \
    ...
    -e 's=ü=\&uuml;;=g' \
    -e 's=ý=\&yacute;;=g' \
    -e 's= =\&thorn;;=g' \
    -e 's=ÿ=\&yuml;;=g' \
    "$@"
```

Hier ist ein Beispiel für die Verwendung dieses Filters:

```
$ cat danish Zeigt dänischen Beispieltext in ISO 8859-1-Codierung
Øen med æn lå i læ af én halvø,
og én stor ø, langs den græske kyst.

$ iso8859-1-to-html danish Konvertiert den Text in HTML-Entities
&Oslash;en med &aring;en l&aring; i l&aring; af &acute;n halv&oslash;;
og &acute;n stor &oslash;; langs den gr&aelig;ske kyst.
```

## 5.3 Beim Kreuzworträtsel schummeln

Kreuzworträtsel präsentieren Ihnen Hinweise zu Wörtern, allerdings bleiben die meisten von uns hängen, wenn wir uns kein Wort vorstellen können, das mit einem b beginnt und entweder ein x oder ein z an der siebten Position hat.

Das schreit dann förmlich nach einer Mustererkennung mit regulären Ausdrücken mit Hilfe von `awk` oder `grep`, aber welche Dateien durchsuchen wir? Eine gute Wahl ist das Unix-Wörterbuch, das auf vielen Systemen als `/usr/dict/words` verfügbar ist. (Andere beliebte Stellen, an denen Sie nach dieser Datei suchen können, sind `/usr/share/dict/words` und `/usr/share/lib/dict/words`.) Es handelt sich hierbei um eine einfache Textdatei mit einem Wort pro Zeile, sortiert in lexikografischer Reihenfolge. Wir können aus einer beliebigen Sammlung von Textdateien ganz einfach weitere ähnlich aussehende Dateien erzeugen:

```
cat datei(en) | tr A-Z a-z | tr -c a-z '\n' | sort -u
```

Die zweite Pipeline-Stufe konvertiert Großbuchstaben in Kleinbuchstaben, die dritte ersetzt Nichtbuchstaben durch Newline-Zeichen und die letzte sortiert das Ergebnis, wobei nur eindeutige Zeilen aufbewahrt werden. Die dritte Stufe behandelt Apostrophe wie Buchstaben, da sie in Zusammenziehungen verwendet werden. Jedes Unix-System verfügt über Text, der auf diese Weise durchsucht werden kann – beispielsweise die formatierten Manpages in `/usr/man/cat*/*` und `/usr/local/man/cat*/*`. Auf einem unserer Systeme lieferten sie mehr als 1 Million Zeilen Prosa und erzeugten eine Liste mit ungefähr 44.000 einzelnen Wörtern. In den verschiedenen Internet-Archiven gibt es auch Wortlisten für Dutzende von Sprachen.<sup>4</sup>

Lassen Sie uns annehmen, dass wir auf diese Weise eine Sammlung mit Wortlisten aufgebaut haben und sie an einer Standardstelle speichern, auf die wir aus einem Skript heraus verweisen können. Wir können dann das Programm schreiben, das in Beispiel 5-4 gezeigt wird.

---

<sup>4</sup> Verfügbar unter <ftp://ftp.ox.ac.uk/pub/wordlists/>, <ftp://qiclab.scn.rain.com/pub/wordlists/>, <ftp://ibiblio.org/pub/docs/books/gutenberg/etext96/pgw> und <http://www.phreak.org/html/wordlists.shtml>. Eine Suche nach »word list« (und auch nach »Wortliste«) in einer Internet-Suchmaschine bringt noch viel mehr Ergebnisse.

#### Beispiel 5-4: Lösungshilfe für Kreuzworträtsel

```
#!/bin/sh
# Filtert ein egrep(1)-artiges Muster anhand einer Sammlung von
# Wortlisten.
#
# Verwendung:
#     puzzle-help egrep-muster [wortlistendateien]

FILES="
    /usr/dict/words
    /usr/share/dict/words
    /usr/share/lib/dict/words
    /usr/local/share/dict/words.biology
    /usr/local/share/dict/words.chemistry
    /usr/local/share/dict/words.general
    /usr/local/share/dict/words.knuth
    /usr/local/share/dict/words.latin
    /usr/local/share/dict/words.manpages
    /usr/local/share/dict/words.mathematics
    /usr/local/share/dict/words.physics
    /usr/local/share/dict/words.roget
    /usr/local/share/dict/words.sciences
    /usr/local/share/dict/words.unix
    /usr/local/share/dict/words.webster
"
pattern="$1"
```

```
egrep -h -i "$pattern" $FILES 2> /dev/null | sort -u -f
```

Die Variable `FILES` enthält die integrierte Liste der Wortlistendateien, die an die lokale Site angepasst ist. Die `grep`-Option `-h` unterdrückt die Dateinamen aus dem Bericht, die Option `-i` ignoriert die Schreibweise, und außerdem werfen wir die Standardfehlerausgabe mit `2> /dev/null`, falls eine der Wortlistendateien nicht existiert oder Ihnen die notwendige Leseberechtigung fehlt. (Diese Art der Umleitung wird in »Manipulation der Dateideskriptoren« [7.3.2] beschrieben.) Die abschließende `sort`-Stufe reduziert den Bericht auf eine Liste einmalig vorkommender Wörter, wobei Groß- und Kleinschreibung ignoriert werden.

Nun können wir nach dem gewünschten Wort suchen:

```
$ puzzle-help '^b....[xz]...$' | fmt
bamboozled Bamboozler bamboozles bdDenizens bdWheezing Belshazzar
botanizing Brontozoom Bucholzite bulldozing
```

Können Sie sich ein englisches Wort vorstellen, in dem sechs Konsonanten aufeinander folgen? Hier kommt eine kleine Hilfe:

```
$ puzzle-help '^[^aeiouy]{6}' /usr/dict/words
Knightsbridge
mightn't
oughtn't
```

Falls Sie *y* nicht als Vokal zählen, kommen sogar noch mehr heraus: *encryption*, *klystron*, *porphyry*, *syzygy* und so weiter.

Wir könnten Zusammenziehungen leicht aus den Wortlisten ausschließen, indem wir einen letzten Filterschritt einführen (`egrep -i '^[a-z]+$'`), es schadet aber nichts, sie in den Wortlisten zu lassen.

## 5.4 Wortlisten

Von 1983 bis 1987 schrieb der Bell Labs-Forscher Jon Bentley eine interessante Kolumne in *Communications of the ACM* mit dem Titel *Programming Pearls*. Manche der Kolumnen wurden später mit einigen Änderungen in zwei Büchern zusammengefasst, die in der Bibliographie aufgeführt sind. In einer der Kolumnen stellte Bentley folgende Herausforderung dar: Schreiben Sie ein Programm zum Verarbeiten einer Textdatei und geben Sie eine Liste der  $n$  am häufigsten auftretenden Wörter aus. Die Häufigkeit des Auftretens soll gezählt werden, die Wörter sollen nach absteigender Häufigkeit sortiert werden. Die berühmten Informatiker Donald Knuth und David Hanson antworteten jeder für sich mit interessanten und raffiniert geschriebenen Programmen,<sup>5</sup> die jeweils mehrere Stunden des Schreibens in Anspruch nahmen. Bentleys Originalspezifikation war ungenau, so dass Hanson sie auf diese Weise neu formulierte: Bei einer vorgegebenen Textdatei und einem Integer-Wert  $n$  sollen Sie in absteigender Häufigkeit diejenigen Wörter (und deren Auftretenshäufigkeit) ausgeben, deren Auftretenshäufigkeit unter den  $n$  größten liegt.

Im ersten von Bentleys Artikeln begutachtete der Bell Labs-Forscherkollege Doug McIlroy Knuths Programm und bot eine sechs Schritte umfassende Unix-Lösung, die nur einige Minuten an Entwicklungsarbeit erforderte und gleich beim ersten Mal korrekt funktionierte. Im Gegensatz zu den beiden anderen Programmen ist das von McIlroy außerdem frei von expliziten magischen Konstanten, die die Wortlängen, die Anzahl der eindeutigen Wörter und die Größe der Eingabedatei begrenzen. Darüber hinaus wird seine Vorstellung davon, was ein Wort ausmacht, völlig von einfachen Mustern bestimmt, die in den ersten beiden Anweisungen definiert werden, wodurch es einfach ist, Änderungen am Worterkennungsalgorithmus vorzunehmen.

McIlroys Programm verdeutlicht die Stärke des Ansatzes mit Unix-Werkzeugen: Teilen Sie ein komplexes Problem in kleinere Teile auf, deren Behandlung Ihnen bereits bekannt ist. Um das Worthäufigkeitsproblem zu lösen, konvertierte McIlroy die Textdatei in eine Liste mit Wörtern – mit jeweils einem Wort pro Zeile – (das erledigt `tr`), wandelte die Wörter in eine einheitliche Schreibweise um (noch einmal `tr`), sortierte die Liste (`sort`),

---

5 *Programming Pearls: A Literate Program: A WEB program for common words*, Comm. ACM **29**(6), 471–483, Juni (1986) und *Programming Pearls: Literate Programming: Printing Common Words*, **30**(7), 594–599, Juli (1987). Knuths Artikel finden Sie auch in seinem Buch *Literate Programming*, Stanford University Center for the Study of Language and Information, 1992, ISBN 0-937073-80-6 (broschiert) und 0-937073-81-4 (fester Einband).

reduzierte sie auf eine Liste einmaliger Wörter mit Zählern (`uniq`), sortierte diese Liste nach absteigenden Zählern (`sort`) und gab schließlich die ersten paar Einträge in der Liste aus (`sed`, obwohl `head` auch funktioniert hätte).

Das resultierende Programm verdient es, einen Namen zu erhalten (`wf` für `word frequency`) und in ein Shell-Skript mit einem kommentierenden Header eingepackt zu werden. Wir erweitern außerdem den ursprünglichen `sed`-Befehl von McIlroy dahin gehend, dass wir das Argument der Länge der Ausgabeliste optional machen und die `sort`-Optionen modernisieren. Das vollständige Programm zeigen wir in Beispiel 5-5.

#### Beispiel 5-5: Worthäufigkeitsfilter

```
#!/bin/sh
# Liest einen Textstrom an der Standardeingabe und gibt eine Liste der
# n (Vorgabe: 25) am häufigsten auftretenden Wörter und
# ihrer Auftretenshäufigkeiten in absteigender Reihenfolge auf der
# Standardausgabe aus.
#
# Verwendung:
#     wf [n]

tr -cs A-Za-z\' '\n' |           Ersetzt Nichtbuchstaben durch Newline-Zeichen
tr A-Z a-z |                   Wandelt Großbuchstaben in Kleinbuchstaben um
sort |                          Sortiert die Wörter in aufsteigender Reihenfolge
uniq -c |                       Eliminiert Duplikate, zeigt deren Zähler an
sort -k1,1nr -k2 |             Sortiert nach absteigendem Zähler und dann nach aufsteigendem Wort
sed ${1:-25}q                   Gibt nur die ersten n (Vorgabe: 25) Zeilen aus; siehe Kapitel 3
```

POSIX-`tr` unterstützt alle Escape-Sequenzen von ISO-Standard-C. Die ältere X/Open Portability Guide-Spezifikation hatte nur oktale Escape-Sequenzen, und das ursprüngliche `tr` hatte überhaupt keine, wodurch das Newline-Zeichen tatsächlich geschrieben werden musste. Das war denn auch einer der Kritikpunkte am Originalprogramm von McIlroy. Glücklicherweise kennt der `tr`-Befehl auf allen Systemen, die wir getestet haben, inzwischen die POSIX-Escape-Sequenzen.

Eine Shell-Pipeline ist nicht die einzige Möglichkeit, dieses Problem mit Unix-Werkzeugen zu lösen: Bentley lieferte in einer früheren Kolumne eine sechszeilige `awk`-Implementierung dieses Programms<sup>6</sup>, die annähernd äquivalent zu den ersten vier Stufen der Pipeline von McIlroy ist.

Knuth und Hanson diskutierten die rechnerische Komplexität ihrer Programme und Hanson setzte eine Laufzeitprofilierung ein, um mehrere Varianten seines Programms zu untersuchen und die schnellste zu ermitteln.

Die Komplexität des McIlroy-Programms ist leicht zu ermitteln. Alle Stufen bis auf `sort` benötigen eine Zeitspanne, die direkt proportional zur Größe ihrer Eingabe ist, und diese

---

<sup>6</sup> *Programming Pearls: Associative Arrays*, Comm. ACM 28(6), 570–576, Juni (1985). Das ist eine ausgezeichnete Einführung in die Leistungsfähigkeit assoziativer Arrays (Tabellen, die anhand von Strings und nicht anhand von Integer-Werten indiziert sind), eine verbreitete Funktion der meisten Skriptsprachen.

Größe wird normalerweise nach der `uniq`-Stufe stark reduziert. Der Schritt, der die Geschwindigkeit drosselt, ist das erste `sort`. Ein guter Sortieralgorithmus, der auf Vergleichen basiert, wie der in `Unix-sort`, kann  $n$  Objekte in einer Zeit sortieren, die proportional zu  $n \log_2 n$  ist. Der Logarithmus-zur-Basis-2-Faktor ist klein: Bei einem  $n$  von etwa 1 Million liegt er bei ungefähr 20. In der Praxis erwarten wir also, dass `wf` einige Male langsamer ist, als wenn wir seinen Eingabestrom einfach nur mit `cat` kopieren würden.

Hier ist ein Beispiel, bei dem wir das Skript auf den Text von Shakespeares beliebtestem Stück *Hamlet* anwenden.<sup>7</sup> Die Ausgabe wird mit `pr` auf eine vierspaltige Anzeige umformatiert:

```
$ wf 12 < hamlet | pr -c4 -t -w80
1148 the          671 of           550 a            451 in
 970 and          635 i            514 my           419 it
 771 to           554 you          494 hamlet       407 that
```

Die Ergebnisse sind für englische Prosa zu erwarten. Interessanter ist es vielleicht, einmal nachzufragen, wie viele einzelne Wörter in diesem Stück vorkommen:

```
$ wf 999999 < hamlet | wc -l
4548
```

Und welche Wörter am seltensten auftauchen:

```
$ wf 999999 < hamlet | tail -n 12 | pr -c4 -t -w80
 1 yaw            1 yesterday      1 yielding       1 younger
 1 yawn           1 yesternight    1 yon            1 yourselves
 1 yeoman         1 yesty          1 yond           1 zone
```

Das Argument `999999` hat nichts Magisches an sich: Die Zahl muss einfach nur größer sein als der größte zu erwartende Zähler für einzelne Wörter, und eine sich wiederholende Ziffer ist einfach leicht zu tippen.

Wir können auch fragen, wie viele der 4548 einzelnen Wörter nur einmal benutzt worden sind:

```
$ wf 999999 < hamlet | grep -c '^ *1•'
2634
```

Das Zeichen `•`, das der Ziffer `1` in dem `grep`-Muster folgt, repräsentiert einen Tabulator. Dieses Ergebnis ist überraschend und möglicherweise atypisch für die meiste moderne englische Prosa: Obwohl das Vokabular des Stücks groß ist, tauchen fast 58 Prozent der Wörter nur einmal auf. Und auch das Kernvokabular der häufig auftretenden Wörter ist ziemlich klein:

```
$ wf 999999 < hamlet | awk '$1 >= 5' | wc -l
740
```

Das ist ungefähr die Anzahl der Wörter, die ein Student innerhalb eines Semesters in einem Fremdsprachenkurs lernen könnte oder die ein Kind lernt, bevor es in die Schule kommt.

---

<sup>7</sup> Verfügbar in den Archiven des wunderbaren Project Gutenberg unter <http://www.gutenberg.net/>.

Shakespeare hatte keine Computer, die ihn bei der Analyse seiner Arbeit unterstützten,<sup>8</sup> aber wir können davon ausgehen, dass ein Teil seines Genies darin bestand, den größten Teil dessen, was er schrieb, dem breiten Publikum seiner Zeit verständlich zu machen.

Als wir wf auf die einzelnen Texte der Shakespeare-Stücke angewendet haben, stellten wir fest, dass *Hamlet* das größte Vokabular umfasste (4548), während die *Komödie der Irrungen* das kleinste besitzt (2443). Die gesamte Anzahl der einzelnen Wörter in allen Shakespeare-Stücken und -Sonetten beträgt fast 23.700, was bedeutet, dass man sich mehrere Stücke anschauen muss, um den Reichtum seines Werks richtig genießen zu können. Ungefähr 36 Prozent dieser Wörter werden nur einmal verwendet, und nur ein Wort beginnt mit x: Xanthippe in *Der Widerspenstigen Zähmung*. Offensichtlich gibt es für Kreuzworträtselenthusiasten und Vokabelanalytiker eine Menge Stoff bei Shakespeare!

## 5.5 Tag-Listen

Die Benutzung des Befehls `tr` zum Beziehen von Wortlisten oder allgemeiner zum Transformieren eines Zeichensatzes in einen anderen, wie in Beispiel 5-5 im vorhergehenden Abschnitt gezeigt, ist sehr praktisch. Das führt natürlich zu einer Lösung eines Problems, das wir beim Schreiben dieses Buches hatten: Wie stellen wir sicher, dass wir durch 50.000 Zeilen von Manuskriptdateien konsistente Auszeichnungen behalten? Beispielsweise könnte ein Befehl mit `<command>tr</command>` ausgezeichnet werden, wenn wir im Fließtext von ihm sprechen. An anderer Stelle könnten wir wiederum ein Beispiel für etwas haben, das Sie eintippen müssen und das durch `<literal>tr</literal>` gekennzeichnet ist. Eine dritte Möglichkeit ist die Manpage-Referenz in der Form `<emphasis>tr</emphasis>(1)`.

Das Programm `taglist` aus Beispiel 5-6 bietet eine Lösung. Es sucht alle Anfangs/End-Tag-Paare, die auf derselben Zeile stehen, und gibt eine sortierte Liste aus, die die Tag-Benutzung mit den Eingabedateien verknüpft. Außerdem markiert es Fälle mit einem Pfeil, in denen das gleiche Wort auf mehr als eine Weise ausgezeichnet wurde. Hier ist ein Ausschnitt seiner Ausgabe aus einer Datei mit einer Version dieses Kapitels:

```
$ taglist ch05.xml
...
  2 cut                command      ch05.xml
  1 cut                emphasis   ch05.xml <----
...
  2 uniq              command      ch05.xml
  1 uniq              emphasis   ch05.xml <----
  1 vfstab            filename    ch05.xml
...

```

---

<sup>8</sup> In der Tat, das einzige von Shakespeare verwendete Wort, das mit der Wurzel von »Computer« zu tun hat, ist »computation«; es kommt in jedem dieser beiden Stücke jeweils einmal vor: *Komödie der Irrungen* und *König Richard III.* »Arithmetic« kommt sechsmal in seinen Stücken vor, »calculate« zweimal und »mathematics« dreimal.

Die Tag-Listen-Aufgabe ist ziemlich komplex und wäre in den meisten konventionellen Programmiersprachen schwer auszuführen, selbst in solchen mit großen Klassenbibliotheken wie C++ und Java, und selbst wenn Sie mit den Programmen von Knuth oder Hanson für das ähnlich gelagerte Worthäufigkeitsproblem beginnen würden. Dennoch reichen dafür gerade einmal neun Schritte in einer Unix-Pipeline mit den bisher vertrauten Werkzeugen aus.

Beim Worthäufigkeitsproblem ging es nicht um benannte Dateien, es wurde einfach nur ein einzelner Datenstrom angenommen. Das ist keine ernsthafte Einschränkung, da wir dem Programm mit `cat` ganz einfach mehrere Eingabedateien übergeben könnten. Hier benötigen wir jedoch einen Dateinamen, da wir natürlich nicht von einem Problem berichten können, ohne zu sagen, wo dieses Problem auftritt. Der Dateiname ist das einzige Argument von `taglist`, im Skript verfügbar als `$1`.

1. Wir schicken die Eingabedatei mit `cat` in die Pipeline. Wir könnten diesen Schritt natürlich eliminieren, indem wir die Eingabe der nächsten Stufe aus `$1` umleiten, allerdings finden wir, dass es in komplexen Pipelines wichtig ist, die *Datenproduktion* von der *Datenverarbeitung* zu trennen. Außerdem ist es dann einfacher, eine weitere Stufe in die Pipeline einzufügen, falls das Programm später weiterentwickelt wird.

```
cat "$1" | ...
```

2. Wir wenden `sed` an, um die ansonsten komplexen Auszeichnungen zu vereinfachen, die für Web-URLs benötigt werden:

```
... | sed -e 's#systemitem *role="url"#URL#g' \  
-e 's#/systemitem#/URL#' | ...
```

Dies wandelt Tags wie `<systemitem role="URL">` und `</systemitem>` in die einfacheren Tags `<URL>` bzw. `</URL>` um.

3. Die nächste Stufe verwendet `tr`, um Leerzeichen und paarweise Trennzeichen durch Newline-Zeichen zu ersetzen:

```
... | tr ' ( ) { } [ ] ' '\n\n\n\n\n\n\n' | ...
```

4. An dieser Stelle besteht die Eingabe aus einem »Wort« pro Zeile (oder aus leeren Zeilen). Wörter sind entweder tatsächlicher Text oder SGML/XML-Tags. Mit Hilfe von `egrep` werden auf der nächsten Stufe Wörter ausgewählt, die in Tags eingeschlossen sind:

```
... | egrep '>[^<>]+</' | ...
```

Dieser reguläre Ausdruck filtert in Tags eingeschlossene Wörter: eine rechte spitze Klammer, gefolgt von wenigstens einer nichtspitzen Klammer, gefolgt von einer linken spitzen Klammer, gefolgt von einem Slash (für das schließende Tag).

5. An dieser Stelle besteht die Eingabe aus Zeilen mit Tags. Die erste `awk`-Stufe verwendet spitze Klammern als Feldtrennzeichen, so dass das eingegebene `<literal>` `tr</literal>` in vier Felder aufgeteilt wird: ein leeres Feld, gefolgt von `literal`, `tr` und `/literal`. Der Dateiname wird auf der Kommandozeile an `awk` übergeben,

wobei die Option `-v` die `awk`-Variable `FILE` auf den Dateinamen setzt. Diese Variable wird dann in der `print`-Anweisung benutzt, die das Wort, das Tag und den Dateinamen ausgibt:

```
... | awk -F'[<>]' -v FILE="$1" \
    '{ printf("%-31s\t%-15s\t%s\n", $3, $2, FILE) }' | ...
```

6. Die `sort`-Stufe sortiert die Zeilen in der Wortreihenfolge:

```
... | sort | ...
```

7. Der Befehl `uniq` liefert das Zählerfeld. Die Ausgabe ist eine Liste mit Datensätzen, in denen es folgende Felder gibt: *Zähler*, *Wort*, *Tag*, *Datei*.

```
... | uniq -c | ...
```

8. Ein zweites `sort` ordnet die Ausgabe nach Wort und Tag (zweites und drittes Feld):

```
... | sort -k2,2 -k3,3 | ...
```

9. Die letzte Stufe verwendet ein kleines `awk`-Programm, um aufeinander folgende Zeilen zu filtern, wobei ein abschließender Pfeil hinzugefügt wird, wenn es das gleiche Wort wie auf der vorhergehenden Zeile sieht. Dieser Pfeil kennzeichnet dann deutlich Instanzen, bei denen Wörter unterschiedlich ausgezeichnet worden sind und die deshalb einer näheren Untersuchung durch die Autoren, die Lektoren oder die Produzenten des Buches bedürfen:

```
... | awk '{
    print ($2 == Last) ? ($0 " <----") : $0
    Last = $2
}'
```

Das vollständige Programm sehen Sie in Beispiel 5-6.

#### *Beispiel 5-6: Erstellen einer SGML-Tag-Liste*

```
#!/bin/sh -
# Liest eine HTML/SGML/XML-Datei, die auf der Kommandozeile angegeben wird und
# Auszeichnungen wie <tag>Wort</tag> enthält, und gibt auf der
# Standardausgabe eine durch Tabulatoren getrennte Liste mit
#
#       Zähler Wort Tag Dateiname
#
# aus, sortiert nach aufsteigendem Wort und Tag.
#
# Verwendung:
#       taglist xml-file

cat "$1" |
sed -e 's#systemitem *role="url"#URL#g' -e 's#/systemitem#/URL#' |
tr '(){}[]' '\n\n\n\n\n\n\n\n\n' |
egrep '>[^<>]+</' |
awk -F'[<>]' -v FILE="$1" \
    '{ printf("%-31s\t%-15s\t%s\n", $3, $2, FILE) }' |
sort |
uniq -c |
sort -k2,2 -k3,3 |
awk '{
```

Beispiel 5-6: Erstellen einer SGML-Tag-Liste (Fortsetzung)

```
    print ($2 == Last) ? ($0 " <----") : $0
    Last = $2
}'
```

In »Funktionen« [6.5] zeigen wir Ihnen, wie Sie die Tag-Listen-Operation auf mehrere Dateien anwenden.

## 5.6 Zusammenfassung

In diesem Kapitel haben wir Ihnen gezeigt, wie Sie verschiedene Textverarbeitungsprobleme lösen, von denen keines einfach so in einer herkömmlichen Programmiersprache lösbar wäre. Die wichtigen Lehren dieses Kapitels sind:

- Datenauszeichnungen sind außerordentlich wertvoll, auch wenn sie nicht unbedingt komplex sein müssen. Ein eindeutiges einzelnes Zeichen, wie ein Tabulator, ein Doppelpunkt oder ein Komma, ist oft ausreichend.
- Pipelines aus einfachen Unix-Werkzeugen und kurze, oft einzeilige Programme in einer passenden Textverarbeitungssprache wie `awk` können Datenauszeichnungen ausnutzen, um mehrere Datenteile an eine Reihe von Verarbeitungsstufen zu übergeben und einen nützlichen Bericht zu liefern.
- Indem die Datenauszeichnungen einfach gehalten werden, kann die Ausgabe unserer Werkzeuge leicht wieder zur Eingabe neuer Werkzeuge werden, wie unsere kleine Analyse der Ausgabe des Worthäufigkeitsfilters `wf`, angewandt auf Shakespeares Texte, gezeigt hat.
- Wenn in der Ausgabe minimale Auszeichnungen verbleiben, können wir später noch einmal zurückkehren und diese Daten weiter auswerten. Dies geschah beispielsweise bei der Umwandlung eines einfachen ASCII-Büroverzeichnisses in eine Webseite. Tatsächlich ist es klug, niemals davon auszugehen, dass irgendeine Form von elektronischen Daten schon die endgültige Form ist: In manchen Bereichen gibt es einen wachsenden Bedarf, bei Seitenbeschreibungssprachen wie PCL, PDF und PostScript die ursprünglichen Auszeichnungen beizubehalten, die zur vorhandenen Seitenformatierung geführt haben. Dokumenten aus Textverarbeitungen fehlen solche nützlichen logischen Auszeichnungen momentan fast völlig, aber das könnte sich in der Zukunft ändern. Zum Zeitpunkt des Verfassens dieser Zeilen wurde von einem bekannten Hersteller von Textverarbeitungsprogrammen berichtet, der es in Betracht zieht, eine XML-Repräsentation für die Dokumentenspeicherung zu verwenden. Die Tabellenkalkulation `gnnumeric` des GNU-Projekts, das Linux Documentation Project<sup>9</sup> und die Office-Suite `OpenOffice.org`<sup>10</sup> tun es bereits.

---

<sup>9</sup> Siehe <http://www.tldp.org/>.

<sup>10</sup> Siehe <http://www.openoffice.org/>.

- Zeilen mit Feldern, die durch Trennzeichen getrennt sind, stellen ein bequemes Format für den Datenaustausch mit komplexeren Programmen dar, wie etwa Tabellenkalkulationen und Datenbanken. Solche Systeme bieten zwar manchmal eine Art von Berichtserzeugung an, aber meist ist es einfacher, die Daten als Strom aus Zeilen mit Feldern zu extrahieren und dann Filter, die in einer passenden Programmiersprache geschrieben wurden, darauf anzuwenden, um die Daten weiterzuverarbeiten. Beispielsweise wird die Katalog- und Verzeichnisveröffentlichung oft am besten auf diese Weise erledigt.