

Die Shell

Die Shell ist der Kommandointerpreter des Unix Systems. Die unterschiedlichen Shells unterscheiden sich in der Handhabung, nicht aber wesentlich in den Konzepten. Hier wird auf die Korn Shell eingegangen.

Inhalt

1. Eingabe- und Ausgabeumlenkung	2
2. Zusammenfassen von Kommandos	6
3. Verarbeitung im Hintergrund.....	9
4. Wildcards	15
5. Quoting.....	20
6. Pipes	25
7. Ausführen von Kommandos zu vorgegebener Zeit.....	28
8. Shell Scripts	32
9. Hörsaalübung	34

1. Eingabe- und Ausgabeumlenkung

Normalerweise erwartet ein Unix Dienstprogramm Eingaben von der Tastatur (stdin) und gibt Resultate auf den Bildschirm (stdout) aus.

Die Shell erlaubt das Umlenken von Standardein- und -ausgabe.

```
$ ls
$ ls
C_Programmierung      ShellProgrammierung
Shell                  tmp
$
$ ls > dat1
$ cat > dat1
$ ls
C_Programmierung      ShellProgrammierung
Shell                  tmp
$
```

Hier wurde die Ausgabe von ls auf die Datei date1 umgelenkt.

```

$ echo ls -l > proc
$
$ ls -l
total 10
drwxr-xr-x  6 as      staff   512 Jan  4 16:31 C_Programmierung
drwxr-xr-x  2 as      staff   512 Jan  7 17:17 Shell
drwxr-xr-x  3 as      staff   512 Dec 11 11:41 ShellProgrammierung
-rw-r--r--  1 as      staff    6 Jan  7 17:21 proc
drwxr-xr-x  3 as      staff   512 Jan  7 17:16 tmp
$
$ cat proc
ls -l
$
$ sh < proc
total 10
drwxr-xr-x  6 as      staff   512 Jan  4 16:31 C_Programmierung
drwxr-xr-x  2 as      staff   512 Jan  7 17:17 Shell
drwxr-xr-x  3 as      staff   512 Dec 11 11:41 ShellProgrammierung
-rw-r--r--  1 as      staff    6 Jan  7 17:21 proc
drwxr-xr-x  3 as      staff   512 Jan  7 17:16 tmp
$

```

Hier wurde die Datei proc mit der Zeichenkette "ls -l" erzeugt. Dann wurde die Eingabe für das Kommando "sh" umgelenkt, so dass sh aus der Datei proc gelesen hat.

Beides zusammen ist auch möglich: Eingabe- und Ausgabeumlenkung.

```

$ sh < proc > dat2
$ cat dat2
total 10
drwxr-xr-x   6 as      staff   512 Jan  4 16:31 C_Programmierung
drwxr-xr-x   2 as      staff   512 Jan  7 17:17 Shell
drwxr-xr-x   3 as      staff   512 Dec 11 11:41 ShellProgrammierung
-rw-r--r--   1 as      staff     0 Jan  7 17:26 dat2
-rw-r--r--   1 as      staff     6 Jan  7 17:21 proc
drwxr-xr-x   3 as      staff   512 Jan  7 17:16 tmp
$

```

Soll die Ausgabe so umgelenkt werden, dass an eine bestehende Datei am Ende angefügt wird, so kann man die Zeichenfolge ">>" verwenden.

```

$ echo Noch eine Zeile an das Ende >> dat2
$ cat dat2
total 10
drwxr-xr-x   6 as      staff   512 Jan  4 16:31 C_Programmierung
drwxr-xr-x   2 as      staff   512 Jan  7 17:17 Shell
drwxr-xr-x   3 as      staff   512 Dec 11 11:41 ShellProgrammierung
-rw-r--r--   1 as      staff     0 Jan  7 17:26 dat2
-rw-r--r--   1 as      staff     6 Jan  7 17:21 proc
drwxr-xr-x   3 as      staff   512 Jan  7 17:16 tmp
Noch eine Zeile an das Ende
$

```

```
$ ls xyz > dat3
xyz: No such file or directory
$ ls -l
total 16
drwxr-xr-x  6 as      staff  512 Jan  7 17:33 .
drwxr-xr-x  5 as      staff  512 Dec  8 21:58 ..
drwxr-xr-x  6 as      staff  512 Jan  4 16:31 C_Programmierung
drwxr-xr-x  2 as      staff  512 Jan  7 17:17 Shell
drwxr-xr-x  3 as      staff  512 Dec 11 11:41 ShellProgrammierung
-rw-r--r--  1 as      staff  418 Jan  7 17:29 dat2
-rw-r--r--  1 as      staff    0 Jan  7 17:33 dat3
-rw-r--r--  1 as      staff   6 Jan  7 17:21 proc
drwxr-xr-x  3 as      staff  512 Jan  7 17:16 tmp
$
```

Fehlerausgaben werden nicht umgelenkt.

Zusammenfassung:

Ein- (stdin) und Ausgabe (stdout) von Kommandos (Programmen) werden durch "<", ">" und ">>" umgelenkt. Fehlermeldungen (stderr) werden dabei nicht umgelenkt.

2. Zusammenfassen von Kommandos

In einer Kommandozeile lassen sich Kommandos durch ";" trennen und mit "(" und ")" gruppieren.

```
$ date; ls -l
Mon Jan  7 17:42:47 MET 2002
total 12
drwxr-xr-x  6 as      staff  512 Jan  4 16:31 C_Programmierung
drwxr-xr-x  2 as      staff  512 Jan  7 17:17 Shell
drwxr-xr-x  3 as      staff  512 Dec 11 11:41 ShellProgrammierung
-rw-r--r--  1 as      staff  418 Jan  7 17:29 dat2
-rw-r--r--  1 as      staff   0 Jan  7 17:33 dat3
-rw-r--r--  1 as      staff   6 Jan  7 17:21 proc
drwxr-xr-x  3 as      staff  512 Jan  7 17:16 tmp
$
```

```
$ (date; ls -l) > dat
$ cat dat
Mon Jan  7 17:45:09 MET 2002
total 14
drwxr-xr-x  6 as      staff   512 Jan  4 16:31 C_Programmierung
drwxr-xr-x  2 as      staff   512 Jan  7 17:17 Shell
drwxr-xr-x  3 as      staff   512 Dec 11 11:41 ShellProgrammierung
-rw-r--r--  1 as      staff    29 Jan  7 17:45 dat
-rw-r--r--  1 as      staff   418 Jan  7 17:29 dat2
-rw-r--r--  1 as      staff     0 Jan  7 17:33 dat3
-rw-r--r--  1 as      staff     6 Jan  7 17:21 proc
drwxr-xr-x  3 as      staff   512 Jan  7 17:16 tmp
$
```

```
$ date; ls -l > dat
Mon Jan  7 17:46:17 MET 2002
$ cat dat
total 12
drwxr-xr-x  6 as      staff   512 Jan  4 16:31 C_Programmierung
drwxr-xr-x  2 as      staff   512 Jan  7 17:17 Shell
drwxr-xr-x  3 as      staff   512 Dec 11 11:41 ShellProgrammierung
-rw-r--r--  1 as      staff    0 Jan  7 17:46 dat
-rw-r--r--  1 as      staff  418 Jan  7 17:29 dat2
-rw-r--r--  1 as      staff    0 Jan  7 17:33 dat3
-rw-r--r--  1 as      staff    6 Jan  7 17:21 proc
drwxr-xr-x  3 as      staff   512 Jan  7 17:16 tmp
$
```

3. Verarbeitung im Hintergrund

Normalerweise führt die Shell ein Kommando aus und gibt erst nach der Ausführung die Kontrolle an den Benutzer zurück (Prompt).

Soll ein Programm im Hintergrund arbeiten, so kann dies der Shell mitgeteilt werden, indem man das Zeichen "&" am Kommandoende anfügt.

```
$ find / -name "*.c" -print > find.txt &  
[1] 4106  
$
```

Die Prozessnummer (PID) wird dabei von der Shell zurück gegeben. Die PID identifiziert den Prozess eindeutig.

Die aktuell ablaufenden Prozesse kann man mit dem Kommando "ps" anzeigen.

```
$ find / -name "*.c" -print > find.txt &  
[1] 4109  
$ ps  
  PID TTY          TIME CMD  
 4026 pts/5        0:00 ksh  
 4109 pts/5        0:01 find  
 4037 pts/5        0:00 bash  
 4111 pts/5        0:00 ps  
$
```

Einen im Hintergrund ablaufenden Prozess kann man durch das Kommando "kill" abbrechen.

```
$ sleep 10000 &
[2] 4116
$ ps
  PID TTY          TIME CMD
 4026 pts/5        0:00 ksh
 4109 pts/5        0:13 find
 4117 pts/5        0:00 ps
 4116 pts/5        0:00 sleep
 4037 pts/5        0:00 bash
$ kill 4116
Terminated                sleep 10000
$ ps
  PID TTY          TIME CMD
 4026 pts/5        0:00 ksh
 4109 pts/5        0:14 find
 4118 pts/5        0:00 ps
 4037 pts/5        0:00 bash
$
```

Wenn ein Benutzer sich abmeldet, so werden normalerweise seine im Hintergrund ablaufenden Prozesse vom System "gekillt".

Möchte man dies verhindern, so muss man das Kommando mit "nohup" starten.

```
$ nohup grep "<math.h>" *.c > use_mathlib &
[1] 4122
$
$ exit
login: _
```

Wenn ein Programm im Hintergrund abläuft, kann es in der ksh bzw. bash durch die Kommandos fg (foreground) und bg (background) und sogenannter Suspension Charakter verwaltet werden.

Zunächst das Verarbeiten im Hintergrund und das wieder in den Vordergrund bringen:

```
$ sleep 60&
[1] 7163
as@trex:
$ fg
sleep 60
```

Sind mehrere Hintergrundprozesse vorhanden, so kann man sich diese Jobs mit dem Kommando "jobs" ansehen:

```
$ sleep 70&
[1] 7166
$ sleep 80&
[2] 7167
$ jobs
[1]-  Running          sleep 70 &
[2]+  Running          sleep 80 &
$
```

fg kann als Parameter die Jobnummer haben, die in den Vordergrund gebracht werden soll:

```
$ jobs
[1]-  Running          sleep 80 &
[2]+  Running          sleep 70 &
as@trex:
$ fg %1
sleep 80
```

fg bewirkt ein Zurückholen auf den Bildschirm, bis zum Programmende. Mit ^Z kann das Programm stoppen und mit bg wieder in die Hintergrundverarbeitung bringen.

```
$ sleep 100&
[1] 7174
$ fg
sleep 100
^Z
[1]+  Stopped                  sleep 100
$ jobs
[1]+  Stopped                  sleep 100
$ bg
[1]+ sleep 100 &
as@trex:
$ jobs
[1]+  Running                  sleep 100 &
$
```

Das Suspend Zeichen ^Z ist mittels stty Kommando (stty susp) definierbar (vgl. Kapitel Terminalverwaltung).

4. Wildcards

Durch die Zeichen "*", "?" und eckige Klammern "[]" können Dateinamen vereinfacht hingeschrieben werden.

Solche Wildcards (Stellvertreterzeichen) werden von der **Shell behandelt**, d.h. sie werden expandiert **bevor** die Shell das Kommando mit den expandierten Strings ausführt.

Gehen wir von folgendem Inhalt des aktuellen Verzeichnisses aus:

```
$ ls
123.x      abc1      xy1      xy2345    xywarten
abc        xy        xy2      xy2345.c
$
```

Ein Stern "*" steht für eine beliebige Zeichenfolge von 0 oder mehreren Zeichen ohne den Punkt an erster Stelle.

Liste alle Dateien, die mit xy beginnen:

```
$ ls xy*
xy          xy1        xy2        xy2345    xy2345.c  xywarten
$
```

Liste alle Dateien auf:

```
$ ls *
123.x      abc1      xy1      xy2345    xywarten
abc        xy        xy2      xy2345.c
$
```

Liste alle C-Quellprogramme auf (enden auf .c)

```
$ ls *
123.x      abc1      xy1      xy2345    xywarten
abc        xy        xy2      xy2345.c
$
```

Liste alle Dateien auf, die ein b enthalten.

```
$ ls *b*
abc      abc1
$
```

Ein Fragezeichen "?" steht für ein beliebiges Zeichen.

Liste alle Dateien auf, die mit xy beginnen und deren Namen 3 Zeichen lang ist.

```
$ ls xy?
xy1  xy2
$
```

Das Fragezeichen wird häufig verwendet, um Dateinamen anzusprechen, die nicht druckbare Zeichen enthalten:

Beispiel:

Angenommen man verwendet das mv-Kommando und macht einen Tippfehler; man schreibt versehentlich ein nicht darstellbares Zeichen. Nun gehört das Zeichen zum Dateinamen.

Anlegen der Datei:

```
$ echo Test mit nicht darstellbaren Zeichen > trial
$
```

Beim mv Kommando verschrieben:

```
$ mv trial trial^G
$ ls
123.x      abc1      xy        xy2       xy2345.c
abc        trial    xy1      xy2345    xywarten
$
```

? verwenden, um Dateinamen zu korrigieren.

```
$ mv trial? trial1
as@trex:
$ ls
123.x      abc1      xy        xy2       xy2345.c
abc        trial1    xy1       xy2345    xywarten
$
```

Wenn man nicht weiß, wo man sich verschrieben hat, kann man sich die Namen mit dem `od`-Kommando ansehen:

```
$ mv trial trial^G
$ ls tr* > l
$ od -c l
0000000  t   r   i   a   l 007  \n
0000007
$
```

Eine in eckige Klammern eingeschlossene Zeichenfolge bedeutet ein beliebiges der eingeklammerten Zeichen (Auswahl).

Liste alle Dateien, die mit `xy` beginnen, gefolgt von einer Ziffer und sonst nichts.

```
$ ls xy[0123456789]
xy1  xy2
$
```

Liste alle Dateien, die mit xy beginnen, gefolgt von einer Ziffer und dann beliebige (ev. leere) Zeichenfolge.

```
$ ls xy[0-9]*
xy1          xy2          xy2345      xy2345.c
$
```

Achtung:

Der * kann im Zusammenhang mit einigen Kommandos, z.B. rm unliebsame Folgen haben:

```
$ pwd
/home/as/Vorlesungen/Unix/Shell
$ rm -r *
$
```

Hier werden alle Dateien ab dem aktuellen Verzeichnis inklusive der enthalten Verzeichnisse gelöscht!

5. Quoting

Die Zeichen [,], *, ?, <, >, \ und & sind so genannte Metazeichen. Sollen uninterpretiert von der Shell verwendet werden, so ist das Zeichen "\" voranzustellen. Das Zeichen "\" hebt also die Bedeutung des ihm folgenden Zeichens für die Shell auf.

```
$ echo Test > \>
$ ls \>
>
$ cat \>
Test
$
```

Dieser Mechanismus ist lediglich zu empfehlen, wenn einzelne Zeichen zu quoten sind.

Soll die Bedeutung einer Zeichenfolge erhalten bleiben, also nicht von der Shell interpretiert werden, so kann die Zeichenfolge in einfache oder doppelte Hochkommata eingeklammert werden.

```
$ echo Test > '>>>>>>abc>>>>>>'
$ echo Test > ">>>>>>123>>>>>>"
$ ls \> *
>
>>>>>>123>>>>>> >>>>>>abc>>>>>>
$
```

Dabei behält eine in einfache Hochkommata eingeschlossene Zeichenkette stets ihre Bedeutung, d.h. die Shell liest vom ersten '-Zeichen bis zum nächsten '-Zeichen.

In eine in doppelte Hochkommata eingeschlossene Zeichenfolge interpretiert die Shell die Zeichen \$, ' und ". Ein Beispiel wird später gezeigt.

Mit dem Kommando

grep *Muster Datei*

wird die *Datei* nach Zeilen durchsucht, die *Muster* enthalten.

```
$ cat trial
Diese Zeile enthaelt ein * Zeichen
diese Zeile nicht
$
$ grep Zeile trial
Diese Zeile enthaelt ein * Zeichen
diese Zeile nicht
$
$
$ grep "*" trial
Diese Zeile enthaelt ein * Zeichen
$
```

Gehen wir von folgender Situation aus:

```
$ ls
trial  zeile
$ cat trial
Diese Zeile enthaelt ein * Zeichen
diese Zeile nicht
$
```

Was passiert dann bei

```
$ grep * trial
```

grep findet hier keine Zeile in der Datei, da ja zuerst die Shell den * expandiert zu

```
trial zeile
```

danach wird also das Kommando

```
$ grep trial zeile trial
```

ausgeführt.

```
$ ls
trial  zeile
$ grep * trial
$ mv zeile Zeile
$ ls
Zeile  trial
$ grep * trial
trial:Diese Zeile enthaelt ein * Zeichen
trial:diese Zeile nicht
trial:Diese Zeile enthaelt ein * Zeichen
trial:diese Zeile nicht
$
```

Das Quoten hebt auch die Bedeutung der Leerzeichen auf:

```
$ grep "Diese Zeile" trial
Diese Zeile enthaelt ein * Zeichen
$
```

Wird der Akzent "`" zum Quoten verwendet, so bedeutet dies, dass die Zeichenkette als Kommando interpretiert wird, also die Shell zuerst den gequoteten Text als Kommando ausführt und das Ergebnis dann weiter verarbeitet wird.

```
$ pwd
/home/as/Vorlesungen/Unix/Shell
$ echo pwd
pwd
$ echo `pwd`
/home/as/Vorlesungen/Unix/Shell
$
```

Hier wird die Ausgabe des Kommando "pwd" zum Parameter des Kommando "echo".

6. Pipes

Unix unterstützt die Verwendung von Filtern: Programm, die von stdin lesen und auf stdout schreiben.

Falls bei grep keine zu durchsuchende Datei angegeben ist, so durchsucht grep die Eingabe zeilenweise nach Eingaben, die das Muster enthalten:

```
$ grep abc
Zeile ohne Muster          EINBABE
Zeile mit Muster abc      AUSGABE VON GREP
Zeile mit Muster abc      EINGABE
^D                          EINGABE
$
```

Sollen mehrere Dateien auf stdout angezeigt werden, kann man das cat Kommando mit mehreren Dateien als Parameter aufrufen:

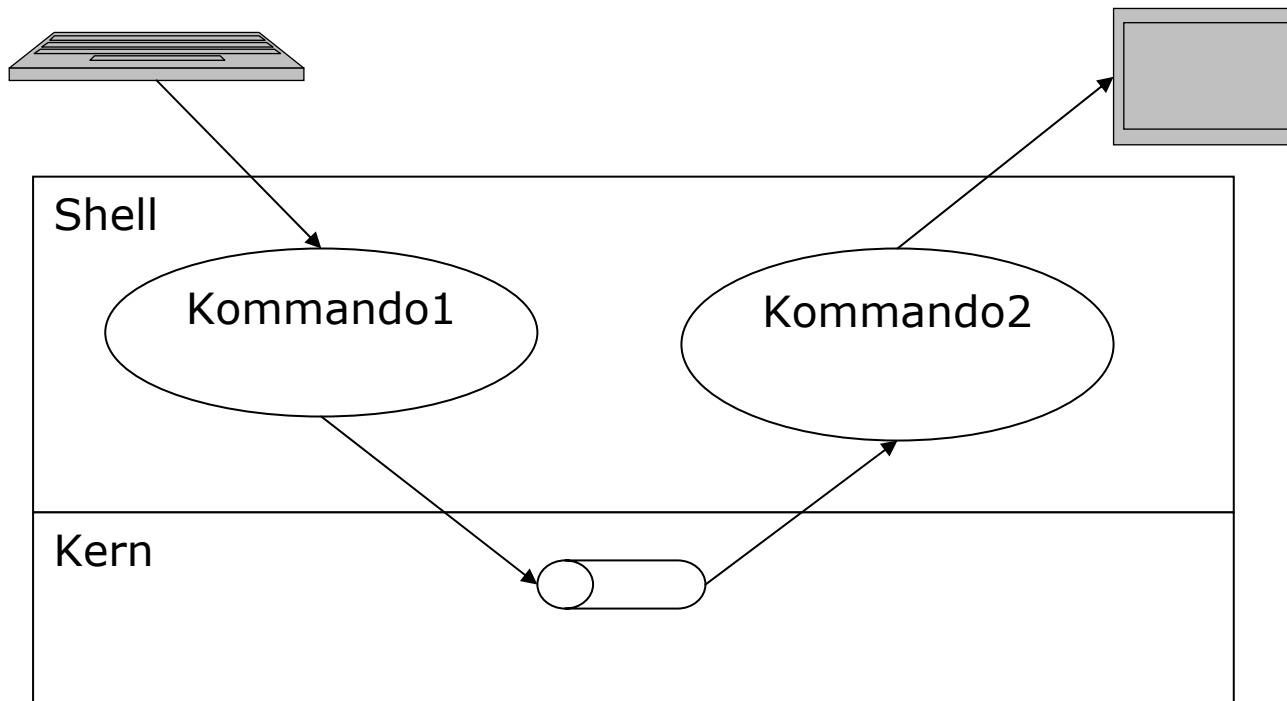
```
$ cat tel_handy tel_handy
Alois  0177 7919531
Philipp 0172 1234567
Alois  0177 7919531
Philipp 0172 1234567
$
```

Um die Ausgabe eines Programms mit der Eingabe eines anderen Programms zu verbinden, das parallel läuft, gibt es den Pipe Operator "|".

Durch

Kommando1 | Kommando2

Wird die Ausgabe von Kommando1 die Eingabe von Kommando2, dabei laufen beide Kommandos als eigener Prozess.



Beispiel mit cat und grep (Suchen in großen Dateien)

```
$ cat tel_handy tel_handy | grep Alois
Alois 0177 7919531
Alois 0177 7919531
$
```

Beispiel cat und pg (Ausgabe großer Dateien seitenweise)

```
$ cat kapitel1 kapitel2 kapitel3 | pg
...
$
```

Beispiel: Ausdruck aller C-Quellen auf Bildschirm mit Kopfzeile:

```
$ cat *.c | pr -F | pg
...
$
```

Beispiel: date und cut Kommando

```
$ date
Tue Jan 8 17:18:26 MET 2002
$ date | cut -c12-19
17:18:28
$
```

7. Ausführen von Kommandos zu vorgegebener Zeit

Durch das at-Kommando können Kommandos zu einer fest vorgegebenen Zeit ausgeführt werden.

Durch

```
$ at Zeit
    Kommando 1
...
Kommando n
^D
$
```

wird festgelegt, dass die n Kommandos zur angegebenen Zeit ausgeführt werden.

```
$ tty
/dev/pts/10
$ date > /dev/pts/10
Tue Jan  8 17:25:46 MET 2002
as@trex:
$ at 17:27
at> date > /dev/pts/10
at> <EOT>
commands will be executed using /bin/ksh
job 1010507220.a at Tue Jan  8 17:27:00 2002
$
```

Die option -l zeigt die eingeplanten Jobs.

```
$ at -l
1010507220.a      Tue Jan  8 17:27:00 2002
as@trex:
$ Tue Jan  8 17:27:00 MET 2002
$
```

Sollen Jobs periodisch ausgeführt werden, so kann der Systemchronometer „cron“ verwendet werden. Dies ist ein Dämon, der im Hintergrund arbeitet und jede Minute in benutzerspezifischen Tabellen, den crontabs, nachsieht, ob ein Kommando auszuführen ist.

Eine solche crontab hat folgenden Aufbau:

```
$ cat crontab.bsp
as@linux> cat crontab.bsp
#       minute (0-59),
#       hour   (0-23),
#       day of the month (1-31),
#       month of the year (1-12),
#       day of the week (0-6 with 0=Sunday).
#       command

* * * * 1-5 /bin/date > /users/as/tmp/cron
as@linux>
```

Das Kommando `crontab` dient dem Erzeugen, Editieren und Löschen von crontabs.

Aufruf:

```
crontab [ -u user ] file
```

```
crontab [ -u user ] { -l | -r | -e }
```

Optionen:

-l Anzeige

-e Editieren

-r Löschen

8. Shell Scripts

Die Shell nimmt Anweisungen nicht nur von stdin entgegen. Die Shell kann auch Kommandos aus Textdateien lesen.

Das folgende Script produziert die Anzeige, die die nächsten Inhalte der Vorlesung beschreibt.

```
$ cat script
date
banner gleich gehts weiter
sleep 20
echo das naechste Kapitel behandelt Shell Programmierung
$
```

```
$ sh script
```

```
Tue Jan 8 17:32:38 MET 2002
```

```
##### # ##### # ##### # #  
# # # # # # # # # #  
# # ##### # # #####  
# ### # # # # # # # #  
# # # # # # # # # #  
##### ##### # # # # # #
```

```
##### ##### # # ##### #  
# # # # # # # # # #  
# ##### ##### # # # # #  
# ### # # # # # # # #  
# # # # # # # # # #  
##### ##### # # # # # #
```

```
# # ##### # ##### # # # # #  
# # # # # # # # # #  
# # ##### # # # # # # # #  
# # # # # # # # # # # # # #  
## ## # # # # # # # # # #  
# # ##### # # ##### # # #
```

```
das naechste Kapitel behandelt Shell Programmierung
```

```
$
```

9. Hörsaalübung

Ein Systemverwalter soll täglich alle Dateien löschen in denen das Wort „Sex“ vorkommt und die die Endung .HTML oder .html haben.

Was muss der Systemverwalter tun, damit die Aufgabe automatisiert wird?
