– Lösung zur Praktikumsaufgabe 2 –

Thema: *Textarbeit unter Unix, einfache Skripte*

3. Das Kommando `man` sollte Ihnen weiterhelfen.

Kommando	*Funktion*
`find`	Finden von Dateien
`grep`	Suchen von Zeichenketten *innerhalb* von Dateien
`sort`	Sortieren von Textzeilen
`wc`	Zählen von Zeichen, Worten, Zeilen

4.

```
~> find / -name test
```

liefert eine Menge von Ergebnissen, jedoch leider auch Fehlermeldungen, da man als gewöhnliche Nutzer nicht jedes Verzeichnis betreten darf.

Wie wir wissen, werden Fehlermeldungen jedoch nach `stderr` geschrieben, während gewöhnliche Ausgaben nach `stdout` landen. Dies liefert uns einen Ansatz, die Fehlermeldungen auszuschalten.

```
~> find / -name test 2>/dev/null
```

Da wir noch wissen wollen, *wieviele* Dateien namens `test` existieren, müssen wir nur noch die Zeilen zählen:

```
~> find / -name test 2>/dev/null | wc -l
```

5. Die Datei `bibel.txt` enthalte im folgenden den gesuchten Text. Dieser ist im Original Latin1-kodiert und muss gegebenenfalls mit dem Kommando

```
recode l1..u8 <datei.txt>
```

in UTF-8-Kodierung umgewandelt werden, sonst gibt es Ärger mit den Umlauten. Bitte beachten Sie auch, dass das Kommando *in-place* arbeitet, der Text wird also *nicht* kopiert.

a)
- Jede Zeile enthält einen Vers.
- Die ersten 3 Buchstaben kodieren das Buch („Gen“, „Exo“ usw.)
- Danach folgt die Kapitel- und Versnumerierung, getrennt durch einen Doppelpunkt.

- Danach folgt der eigentliche Bibelvers, eingeleitet durch einen Großbuchstaben und abgeschlossen mit einem Satzzeichen.
- Die Worte des Verses sind durch Leerzeichen voneinander getrennt.
- Umlaute kommen vor.
- In einigen wenigen Versen folgt nach der Kapitel-/Versnumerierung eine zweite Numerierung in eckigen Klammern.

b)

```
~> cat bibel.txt | wc -l -w
  31102  764512
```

c)

```
~> grep Schlange | wc -l
49
```

Da bereits das einmalige Vorkommen des Suchbegriffs dazu führt, dass die entsprechende Zeile ausgegeben wird, unterschlagen wir damit mehrfaches Auftreten in einer Zeile.

Ohne bereits auf Details regulärer Ausdrücke einzugehen (vgl. Praktikum 04), versuchen wir diesen Sonderfall zu analysieren.

```
~> grep "Schlange.*Schlange" bibel.txt
```

zeigt, dass ein Vers der Bibel zweimal den Begriff „Schlange“ enthält, ein Vers sogar dreimal. Der Schalter `'-o'` sorgt dafür, dass `grep` wirklich jedes Auftreten des Suchbegriffes in einer Zeile meldet (überzeugen Sie sich!). Interessanterweise sparen ihn verschiedene `man`-Pages aus.

Die korrekte Lösung der Aufgabe lautet also:

```
~> grep -o Schlange bibel.txt | wc -l
52
~> grep -o Maus bibel.txt | wc -l
1
~> grep -o Löwe bibel.txt | wc -l
142
```

d) Wie bereits in der Aufgabenstellung vermerkt, führt das folgende Kommando zum Ergebnis.

```
~> cut -d ' ' -f 1 bibel.txt
```

Wollen Sie stattdessen wissen, wieviele *verschiedene* Bücherkürzel es gibt, dann hilft:

```
~> cut -d ' ' -f 1 bibel.txt | uniq
```

e) Wie in der Aufgabenstellung vermerkt müssen zunächst die Verstexte ausgeschnitten werden, bevor sie alphabetisch mittels `sort` sortiert werden können.

```
~> cut -d ' ' -f 3- bibel.txt | sort
```

6. Ein Shellskript ist nichts weiter als eine Ansammlung von Shell-Kommandos (am besten eines auf jeder Zeile). Damit die Shell selbst erkennt, dass es sich um ein Shellskript handelt, muss die erste Zeile den sogenannten *Shebang* enthalten, was in unserem Falle die Zeichenkette `#!/bin/sh` ist.

   Da Textdateien (und nichts anderes ist ein Shellskript) standardmäßig ohne das Recht zur Ausführung angelegt werden, müssen wir dieses Recht dem Skript mittels `chmod +x <skriptname>` verleihen.

```
#!/bin/bash
echo 'Tiere in der Bibel'
echo '------------------'
echo Schlangen:
grep -o Schlange bibel.txt | wc -l
echo Mäuse:
grep -o Maus bibel.txt | wc -l
echo Löwen:
grep -o Löwe bibel.txt | wc -l
```

   Die Ausgabe kann auch in einer einzigen (langen) Zeile erfolgen:

```
#!/bin/bash
echo -n Es gibt `grep -o Schlange bibel.txt | wc -l` \
echo -n Schlangen, `grep -o Maus bibel.txt | wc -l`\
echo -n Mäuse und `grep -o Löwe bibel.txt | wc -l` \
echo Löwen in der Bibel
```

   Noch unübersichtlicher wird es, wenn wir alles in einer Programmzeile verstauen (nicht zur Nachahmung empfohlen):

```
#!/bin/bash
echo Es gibt `grep -o Schlange bibel.txt | wc -l` Schlangen, `grep ↩
    -o Maus bibel.txt | wc -l` Mäuse und `grep -o Löwe bibel.txt | ↩
    wc -l` Löwen in der Bibel
```

7. a)

```
#!/bin/bash
echo 'Tiere in der Bibel'
```

```
echo  ------------------
echo $1: `grep -o -c $1 bibel.txt.utf8`
```

b)

```
#!/bin/bash
echo 'Tiere in der Bibel'
echo  ------------------
echo $1: `grep -o -c $1 $2`
```

Wir haben bewußt noch keine Fehleranalyse der Kommandozeilenparameter vorgenommen, dies erfolgt in Praktikum 3.

8.* Das Kommando `dd` ist äußerst flexibel zum Einlesen und Generieren von Daten in allgemeinster Form zu gebrauchen. So kann es z. B. zum Sichern und Restaurieren des Masterbootrecords (MBR) einer Festplatte, zur bitgenauen Kopie eines beliebigen Datenträgers oder, wie im vorliegenden Beispiel, zur Generierung von Dateien spezifischer Länge genutzt werden.

`dd` übernimmt stets eine Eingabedatei (`if`) und eine Ausgabedatei (`of`), aus denen die Daten gelesen bzw. in die die Daten geschrieben werden.

Darüberhinaus ist die Angabe einer Blockgröße (`bs`, *block size*) sowie der Anzahl zu lesender/schreibender Blöcke (`count`) erforderlich. `bs`·`count` ergibt die Größe der Zieldatei.

```
~> dd if=/dev/urandom of=rnd-1mib.bin bs=512 count=2048
```

schreibt eine 1 MiB große Datei mit Zufallszahlen und

```
~> dd if=/dev/zero of=zero-1mib.bin bs=1024 count=1024
```

generiert eine ebensogroße Datei mit Nullbytes (Achtung: `/dev/zero` nicht mit `/dev/null` verwechseln).

Überzeugen Sie sich mittels `hexdump` vom Inhalt der generierten Datei:

```
~> hexdump -C zero-1mib.bin
```

9. Es handelt sich um eine Scherzaufgabe. Sie sollen mitnichten die gewünschte Funktionalität als Shellskript nachprogrammieren (so weit sind wir im Stoff ohnehin noch nicht), sondern stattdessen das System einsetzen.

   Es gibt bereits ein Kommando, was exakt für die Bewältigung der Aufgabenstellung entwickelt wurde, `du` (*disk usage*). Wir müssen nur noch erreichen, dass nur die Gesamtsumme und zwar in Bytes ausgegeben wird. Fehlermeldungen über gesperrte Ver-

zeichnisse sollten möglicherweise ebenfalls verschwinden. Das folgende Skript leistet das Gewünschte:

```
#!/bin/bash
du -b -s $1 2>/dev/null | cut -f1
```

10.*

```
#include <stdio.h>
#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>
#include <sys/stat.h>
#include <dirent.h>
#include <libgen.h>
#include <errno.h>

#define MAXPATHLGTH 8192

#define DEBUG

double rd_dir(const char*);

double rd_dir(const char *name)
{
  DIR *stream;
  struct dirent *cur_entry;
  struct stat file_stat;
  char pathname[MAXPATHLGTH];
  double sumlength = 0;

#ifdef DEBUG
  printf("Opening %s\n", name);
#endif
  stream = opendir(name);
  if (stream == NULL) {
    perror(name);
    return 0;  /* ignoriere unlesbare Verzeichnisse */
  }

  while(1) {
    errno = 0;
    if (cur_entry=readdir(stream)) {
      strncpy(pathname, name, MAXPATHLGTH-2);
      strcat(pathname, "/");
      if (strlen(pathname)+strlen(cur_entry->d_name)>=MAXPATHLGTH) ↩
          {
        printf("Pathname exceeding %d Bytes. Exiting.\n",  ↩
            MAXPATHLGTH);
        exit(EXIT_FAILURE);
      }
      strcat(pathname, cur_entry->d_name);
#ifdef DEBUG
```

```
        printf("-- Testing %s \n", pathname);
#endif
      if (!lstat(pathname, &file_stat)) {
  if (S_ISLNK(file_stat.st_mode)) {
    /* symbolischen Link ueberspringen */
    continue;
  }
  else if ((S_ISDIR(file_stat.st_mode))
     && ((strcmp(cur_entry->d_name, "."))!=0)
     && ((strcmp(cur_entry->d_name, ".."))!=0)) {
    /* Verzeichniseintrag ist Verzeichnis, <> "..", "." */
    sumlength += rd_dir(pathname);
  }
  else if (S_ISREG(file_stat.st_mode)) {
    /* es handelt sich um eine regulaere Datei */
    sumlength += (double) file_stat.st_size;
  }
      }
      else {  /* lstat() fehlgeschlagen */
  perror(pathname);
      }

    }
    else { /* cur_entry == NULL */
      if (errno != 0) { /* Test, ob Fehler oder EOF */
  perror(name);
      }
      closedir(stream);
      return sumlength;
    }
  }
}


int main(int argc, char* argv[])
{
  if (argc!=2) {
    printf("Usage: %s <path>\n", basename(argv[0]));
    exit(EXIT_FAILURE);
  }

  printf("du: %0.0lf Bytes\n", rd_dir(argv[1]));

  return EXIT_SUCCESS;
}
```