Betriebssysteme

WS 2015/16

Hans-Georg Eßer

Foliensatz B: v1.2, 2015/08/20

- Shell: Variablen, History, Filter-Programme, Pipelines
- C: Der GNU C Compiler (gcc)
- C: Crashkurs zur Programmiersprache C

Übersicht

A	Einführung Shell Dateiverwaltung	System Calls	D
		Software-Verwaltung	E
		Scheduler / Prioritäten	F
Folien B	Filter C-Compiler	Synchronisation	G
C	Prozesse / Jobs Threads		
D	Interrupts	Zusammenfassung	H

Shell: Variablen, History und Filter

Arbeiten in der Shell

Aus Einführung und Übungsblatt 1 bereits bekannt:

- pwd: aktuelles (Arbeits-) Verzeichnis anzeigen
- cd: Verzeichniswechsel
- .. : nächst höheres Verzeichnis
- ls: Verzeichnisinhalt anzeigen
- cp: Datei kopieren
- vi: Text-Editor
- mkdir: Verzeichnis erzeugen
- rmdir: Verzeichnis löschen
- rm: Datei löschen
- rm -r: Verzeichnis rekursiv löschen
- touch: Datei (leer) erzeugen; Zugriffsdatum aktualisieren
- less: Datei anzeigen
- grep: Suchen in Datei
- head, tail: Anfang und Ende einer Datei
- man: Hilfe anzeigen
- dmesg: Systemmeldungen ausgeben
- wc: word count
- shutdown: System runter fahren

Shell-Variablen (1)

- Die Shell (und auch andere Programme) nutzen **Umgebungsvariablen** (für Optionen, Einstellungen etc.)
- „set“ gibt eine Liste aller in dieser Shell gesetzten Variablen aus

```
$ set
BASH=/bin/bash
BASH_VERSION='3.2.48(1)-release'
COLUMNS=156
COMMAND_MODE=unix2003
DIRSTACK=()
DISPLAY=/tmp/launch-Lujw2L/org.x:0
EUID=501
GROUPS=()
HISTFILE=/home/esser/.bash_history
HISTFILESIZE=500
HISTSIZE=500
HOME=/home/esser
HOSTNAME=macbookpro.fritz.box
...
```

Shell-Variablen (2)

- Einzelne Variablen geben Sie mit „echo“ und einem Dollar-Zeichen ($) vor dem Variablennamen aus

```
$ echo $SHELL
/bin/bash
$ _
```

- zum Ändern / Setzen schreiben Sie „var=wert“:

```
$ TESTVAR=fom
$ echo $TESTVAR
fom
$ set | grep TEST
TESTVAR=fom
$ _
```

- Sie können Variablen auch **exportieren**:

```
$ export TESTVAR
$ _
```

→ nächste Folie

Shell-Variablen (3)

- Exportieren?

  Wert einer Variablen gilt nur lokal in der laufenden Shell.

- Exportierte Variablen gelten auch in aus der Shell heraus gestarteten Programmen

```
$ A=eins; B=zwei; export A
$ echo "A=$A B=$B"
A=eins B=zwei
$ bash              # neue Shell starten; das ist ein neues Programm!
$ echo "A=$A B=$B"
A=eins B=
$ exit              # diese zweite Shell verlassen, zurück zur ersten
$ echo "A=$A B=$B"
A=eins B=zwei
```

!

Shell-Variablen (4)

- Liste aller exportierten Variablen gibt „export“ ohne Argument aus – allerdings in ungewöhnlicher Syntax

```
$ export
declare -x A="1"
declare -x Apple_PubSub_Socket_Render="/tmp/launch-CYfDhh/Render"
declare -x COMMAND_MODE="unix2003"
declare -x DISPLAY="/tmp/launch-Lujw2L/org.x:0"
declare -x HOME="/Users/esser"
declare -x INFOPATH="/sw/share/info:/sw/info:/usr/share/info"
declare -x LOGNAME="esser"
...
```

- (Hintergrund: „declare -x VAR“ exportiert ebenfalls die Variable VAR, ist also dasselbe wie „export VAR“)

History (1)

- Shell merkt sich die eingegebenen Befehle („History“)
- Komplette Ausgabe mit „history“:

```
$ history
1  df -h
2  ll
3  /opt/seamonkey/seamonkey
4  dmesg|tail
5  ping hgesser.de
6  google-chrome
7  killall kded4
```

- Wie viele Einträge? Normal 500:

```
$ echo $HISTSIZE
500
```

History (2)

- Neben Ausgabe der kompletten History gibt es auch eine intelligente Suche nach alten Kommandos: [Strg-R]

```
$  # Suche nach dem letzten echo-Aufruf
$ ^R
(reverse-i-search)`ech': echo $HISTFILESIZE
```

- mit [Eingabe] ausführen
- weitere [Strg-R] liefern ältere Treffer
- Außerdem: Mit [Pfeil hoch], [Pfeil runter] durch alte Befehle blättern
- gefundenes Kommando kann übernommen und überarbeitet werden

Filter für Text-Streams

- Idee beim Filter:
  - Standardeingabe in Standardausgabe verwandeln
  - Ketten aus Filtern zusammen bauen:
  - `prog1 | filter1 | filter2 | filter3 ...`
  - mit Eingabedatei:
    `prog1 < eingabe | filter1 | ...`
- cat, cut, expand, fmt, head, od, join, nl, paste, pr, sed, sort, split, tail, tr, unexpand, uniq, wc

cat

- cat steht für con**cat**enate (aneinanderfügen)
- gibt mehrere Dateien unmittelbar hintereinander aus
- auf Wunsch auch nur eine Datei
  → Mini-Dateibetrachter
- Spezialoptionen:
  - -n (Zeilennummern)
  - -T (Tabs als ^I anzeigen)
  - … und einige weitere (siehe: man cat)

cut

- cut kann spaltenweise Text ausschneiden – Spalten sind wahlweise definierbar über
  - Zeichenpositionen
  - Trennzeichen (die logische Spalten voneinander trennen)

c: character; zeichenbasiert

d: delimiter; Trennzeichen

```
$ cat test.txt
1234 678901 234
abc def ghijklmn
r2d2 12 99
1 2 3
Langer Testeintrag
```

```
$ cut -c3-8 test.txt
34 678
c def
d2 12
2 3
nger T
```

```
$ cut -d" " -f2,3 test.txt
678901 234
def ghijklmn
12 99
2 3
Testeintrag
```

f: field (Feld)

fmt

- fmt (**for**mat) bricht Textdateien um

keine Umbrüche

```
$ cat test.txt
Das ist mal ein Beispiel fuer einen Satz. Das ist mal ein Beispiel fue
r einen Satz. Das ist mal ein Beispiel fuer einen Satz. Das ist mal ei
n Beispiel fuer einen Satz. Das ist mal ein Beispiel fuer einen Satz.
Das ist mal ein Beispiel fuer einen Satz. Das ist mal ein Beispiel fue
r einen Satz. Das ist mal ein Beispiel fuer einen Satz. Das ist mal ei
n Beispiel fuer einen Satz. Das ist mal ein Beispiel fuer einen Satz.

$ fmt test.txt
Das ist mal ein Beispiel fuer einen Satz. Das ist mal ein Beispiel
fuer einen Satz. Das ist mal ein Beispiel fuer einen Satz. Das ist
mal ein Beispiel fuer einen Satz. Das ist mal ein Beispiel fuer
einen Satz. Das ist mal ein Beispiel fuer einen Satz. Das ist mal
ein Beispiel fuer einen Satz. Das ist mal ein Beispiel fuer einen
Satz. Das ist mal ein Beispiel fuer einen Satz. Das ist mal ein
Beispiel fuer einen Satz.
```

Zeilen-umbrüche

- Parameter -w75: Breite 75 (width)

split

- split kann große Dateien in mehrere Dateien mit angegebener Maximalgröße aufteilen
- (cat fügt diese anschließend wieder zusammen)

```
$ split ZM_ePaper_18_11.pdf -b1440k ZM_ePaper_18_11.pdf.
$ ls -l ZM*
-rw-r--r-- 1 esser esser 10551293 2011-04-29 06:58 ZM_ePaper_18_11.pdf
-rw-r--r-- 1 esser esser  1474560 2011-04-29 14:46 ZM_ePaper_18_11.pdf.aa
-rw-r--r-- 1 esser esser  1474560 2011-04-29 14:46 ZM_ePaper_18_11.pdf.ab
-rw-r--r-- 1 esser esser  1474560 2011-04-29 14:46 ZM_ePaper_18_11.pdf.ac
-rw-r--r-- 1 esser esser  1474560 2011-04-29 14:46 ZM_ePaper_18_11.pdf.ad
-rw-r--r-- 1 esser esser  1474560 2011-04-29 14:46 ZM_ePaper_18_11.pdf.ae
-rw-r--r-- 1 esser esser  1474560 2011-04-29 14:46 ZM_ePaper_18_11.pdf.af
-rw-r--r-- 1 esser esser  1474560 2011-04-29 14:46 ZM_ePaper_18_11.pdf.ag
-rw-r--r-- 1 esser esser   229373 2011-04-29 14:46 ZM_ePaper_18_11.pdf.ah
$ cat ZM_ePaper_18_11.pdf.* > ZM_Kopie.pdf
$ ls -l ZM_Kopie.pdf
-rw-r--r-- 1 esser esser 10551293 2011-04-29 14:48 ZM_Kopie.pdf
$ diff ZM_ePaper_18_11.pdf ZM_Kopie.pdf
$ _
```

sort

- sort ist ein komplexes Sortier-Tool, das
  - Sortierung nach *n*-ter Spalte
  - alphabetische und numerische Sortierung

  unterstützt
- Einfache Beispiele:

```
$ cat test3.txt
13 Autos
5 LKW
24 Fahrraeder
2 Baeume
Wohnung
Haus
Hotel
Strasse
Allee
```

```
$ sort test3.txt
13 Autos
2 Baeume
24 Fahrraeder
5 LKW
Allee
Haus
Hotel
Strasse
Wohnung
```

```
$ sort -n test3.txt
Allee
Haus
Hotel
Strasse
Wohnung
2 Baeume
5 LKW
13 Autos
24 Fahrraeder
```

uniq

- uniq (**uniq**ue, einmalig) fasst mehrere identische (aufeinander folgende) Zeilen zu einer zusammen; entfernt also Doppler
- Alternative: Beim Sortieren mit sort kann man über die Option -u (**u**nique) direkt Doppler entfernen;
  - statt `sort datei | uniq` also besser `sort -u datei`

grep

- grep (**g**lobal/**r**egular **e**xpression/**p**rint) zeigt nur die Zeilen einer Datei, die einen Suchbegriff enthalten – oder nicht enthalten (Option -v)

```
$ wc -l /etc/passwd
57 /etc/passwd
$ grep esser /etc/passwd
esser:x:1000:1000:Hans-Georg Esser,,,:/home/esser:/bin/bash
$ grep /bin/bash /etc/passwd
root:x:0:0:root:/root:/bin/bash
esser:x:1000:1000:Hans-Georg Esser,,,:/home/esser:/bin/bash
$ grep -v /bin/bash /etc/passwd | head -n5
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
```

sed (1/2)

- sed (**S**tream **Ed**itor) führt (u. a.) Suchen-/Ersetzen-Funktionen in einem Text durch

```
$ cat test4.txt
Das Wort ist ein Wort, und mehrere
Woerter sind der Plural von Wort.
Ohne Woerter oder Worte gibt es
keinen Satz - wir sind wortlos.

$ sed 's/Wort/Bild/' test4.txt
Das Bild ist ein Wort, und mehrere
Woerter sind der Plural von Bild.
Ohne Woerter oder Bilde gibt es
keinen Satz - wir sind wortlos.

$ sed 's/Wort/FOM/g' test4.txt
Das FOM ist ein FOM, und mehrere
Woerter sind der Plural von FOM.
Ohne Woerter oder FOMe gibt es
keinen Satz - wir sind wortlos.
```

```
$ sed 's/Wort/FOM/gi' test4.txt
Das FOM ist ein FOM, und mehrere
Woerter sind der Plural von FOM.
Ohne Woerter oder FOMe gibt es
keinen Satz - wir sind FOMlos.
```

s: substitute (**s**/.../.../gi)
g: global (s/.../.../**g**i)
i: ignore case (s/.../.../g**i**)

Die i-Option gibt es nicht in jeder sed-Version!

sed (2/2)

- sed-Optionen:
  - -i: in-place-editing, verändert die angegebene Datei; am besten mit Angabe eines Suffix für eine Backup-Datei:
    z. B. `sed -i.bak 's/Wort/Bild/g' test4.txt`
    legt erst Sicherheitskopie test4.txt.bak an und verändert dann test4.txt
  - -e: zum Kombinieren mehrerer Ersetzungen; z. B.
    `sed -e 's/1/eins/g' -e 's/2/zwei/g' test.txt`
  - weitere Optionen → Manpage

Reguläre Ausdrücke

- Idee: Allgemeinere Suchbegriffe, vergleichbar mit Wildcards (*, ?) bei Dateinamen
- Muster:
  - . – ein beliebiges Zeichen
  - [abcd] – eines der Zeichen a, b, c, d
  - [2-8] – eines der Zeichen 2, 3, 4, 5, 6, 7, 8
  - ^ – Zeilenanfang
  - $ – Zeilenende
  - ? – vorheriger Ausdruck darf vorkommen, muss aber nicht
  - * – vorheriger Ausdruck kann beliebig oft (auch 0 mal) vorkommen

```
$ cat test5.txt
Haus
Die Hotels
Hotels am Wasser
Bau-Haus-Objekt
Diese Zeile nicht
```

```
$ grep 'H.*s' test5.txt
Haus
Die Hotels
Hotels am Wasser
Bau-Haus-Objekt
```

```
$ sed 's/H.*s/HAUS/g' test5.txt
HAUS
Die HAUS
HAUSer
Bau-HAUS-Objekt
Diese Zeile nicht
```

Reguläre Ausdrücke

- Beispiele für reguläre Ausdrücke
  (live, in der Shell...)

C-Compiler

Benutzung des C-Compilers gcc

- Quellcode-Datei `prog.c` im Editor erstellen
- In der Shell

  ```
  gcc -o prog prog.c
  ```

  eingeben, um zu kompilieren
- In der Shell

  ```
  ./prog
  ```

  eingeben (wichtig: „`./`“ am Anfang), um das übersetzte Programm zu starten

Einführung in C (1)

- Vorab das wichtigste:
  - keine Klassen / Objekte
    → statt Objekten:
    „structs“ (zusammengesetzte Datentypen)
    → statt Methoden nur Funktionen
    → zu bearbeitende Variablen immer als Argument übergeben
  - kein String-Datentyp (sondern Zeichen-Arrays)
  - häufiger Einsatz von Zeigern
  - `int main () {}` ist immer Hauptprogramm

Einführung in C (2)

- Ausführlichere Informationen fürs Selbststudium: http://www.c-howto.de/

  → auf der Webseite: ausführlichere Version mit erklärenden Kommentaren
  (Download: Zip-Archiv)
- auch als Buch für ca. 20 € erhältlich
- in der Vorlesung/Übung: Fokus auf Unterschiede zu C++/C#/Java
- keine komplexen C-Programme im Kurs

Einführung in C (3)

- Im Anschluss an diese Vorlesung: erstes Übungsblatt mit C-Aufgaben
- Vorbereitend ein paar Informationen zu
  - **Structs** (Strukturen, zusammengesetzte Typen)
  - **Pointern**
  - **Quellcode- und Header-Dateien** (Prototypen)

Strukturen und Pointer (1)

Structs

- Mehrere Möglichkeiten der Deklaration

```
struct {
  int i;
  char c;
  float f;
} variable;

variable.i = 9;
variable.c = 'a';
variable.f = 0.123;
```

```
struct mystruct {
  int i;
  char c;
  float f;
};

struct mystruct variable;

variable.i = 9;
variable.c = 'a';
variable.f = 0.123;
```

```
typedef struct {
  int i;
  char c;
  float f;
} mystruct;

mystruct variable;

variable.i = 9;
variable.c = 'a';
variable.f = 0.123;
```

Strukturen und Pointer (2)

Pointer

- Deklaration mit *: `char *ch_ptr;`
- verwalten Speicheradressen (an welchem Ort befindet sich die Variable?)
- Operatoren
  - & (Adresse von)
  - * (Dereferenzieren)

```
char ch, ch2;
char *ch_ptr;  char *ch_ptr2;

ch_ptr = &ch;  // Adresse von ch?
ch2 = *ch_ptr; // Inhalt

ch_ptr2 = ch_ptr;
   // kopiert nur Adresse
```

Strukturen und Pointer (3)

- Struct und Pointer kombiniert
- Oft bei verkettenen Listen

```
struct liste {
  struct liste *next;
  struct liste *prev;
  int inhalt;
};

struct liste *anfang;
struct liste *p;

for (p=anfang; p != NULL; p=p->next) {
  use (p->inhalt);
}
```

Strukturen und Pointer (4)

- Pointer-Typen
  - `typ *ptr;`
    → `ptr` ist ein Zeiger auf etwas vom Typ `typ`
  - `typ **pptr;`
    → `pptr` ist ein Zeiger auf einen Zeiger vom Typ `typ`
  - `ptr` bzw. `pptr` sind Speicheradressen
  - `*ptr` gibt den Wert zurück, der an der Speicherstelle abgelegt ist, auf die `ptr` zeigt
  - analog: `**pptr` ist ein Wert, aber `*pptr` ein Zeiger

Strukturen und Pointer (5)

- Pointer-Typen
  - &-Operator erzeugt zu Variable einen Pointer
  - Beispiele:

```
int i;
int *ip;
int **ipp;

i = 42;
ip = &i;          // ip = Adresse von i
ipp = &ip;         // ipp = Adresse von ip

printf (*ip);  // -> 42
printf (**ipp); // -> auch 42
```

Strukturen und Pointer (6)

- Nicht-initialisierte Pointer: schlecht
  - Beispiel:

```
int *ip;
int **ipp;

printf (ip);    // nicht-init. Adresse (0)
printf (*ip);   // illegal -> Abbruch

*ip = 42;       // auch illegal, schreibt an
                // nicht def. Adresse
```

Strukturen und Pointer (7)

- Vorsicht bei `char* a,b,c;` etc.

```
[esser@macbookpro:tmp]$ cat t2.c
int main () {
  char* a,b;
  printf ("|a| = %d \n", sizeof(a));
  printf ("|b| = %d \n", sizeof(b));
}

[esser@macbookpro:tmp]$ gcc t2.c; ./a.out
|a| = 8
|b| = 1
```

- besser: `char *a, *b, *c;`

Prototypen (1)

- Programm- und Header-Dateien
  - Header-Dateien (*.h) enthalten Funktionsprototypen und Makrodefinitionen (aber keinen normalen Code)
  - Programmdateien (*.c) enthalten den Code, können aber ebenfalls Prototypen und Makros enthalten (kein Zwang, eine .h-Datei zu erzeugen)

Prototypen (2)

- Funktionsprotoypen (in Header-Dateien)
  - erlauben die Verwendung von Funktionen, deren Implementierung weiter unten im Programm (oder in einer anderen Datei) steht
  - Prototyp enthält nur Rückgabetyp, Name und Argumente, z. B.

```
int summe (int x, int y);
```

Prototypen (3)

- Wie findet der Compiler die Header-Dateien?
  → Zwei Varianten:
  - `#include "pfad/zu/datei.h"`

    Dateiname ist Pfad (relativ zu Verzeichnis mit der .c-Datei)
  - `#include <name.h>`

    name.h wird in den Standard-Include-Verzeichnissen gesucht.

    Welche sind das? Beim Bauen des gcc festgelegt...

Include-Verzeichnisse

- Standard-Include-Verzeichnisse

```
[esser@s15337257:~]$ cpp -v
Using built-in specs.
Target: i486-linux-gnu
[...]
#include "..." search starts here:
#include <...> search starts here:
 /usr/local/include
 /usr/lib/gcc/i486-linux-gnu/4.4.5/include
 /usr/lib/gcc/i486-linux-gnu/4.4.5/include-fixed
 /usr/include
End of search list.
```

(und deren Unterordner)