

Betriebssysteme

WS 2014/15

Hans-Georg Eßer
Dipl.-Math., Dipl.-Inform.

Foliensatz B:

v1.0, 2014/09/05

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

BS Praxis und BS Theorie (3)



Einführung
Shell
Dateiverwaltung
Filter
C-Compiler
Prozesse / Jobs
Threads
Interrupts

System Calls
Scheduler / Prioritäten
Synchronisation
Speicherverwaltung
Software-Verwaltung
Partitionen
Auskunft
User, Gruppen, Rechte

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

- 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
...
```

- 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
```

- 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

- 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:/usr/share/info"
declare -x LOGNAME="esser"
...
```

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

- 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
```

- 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`

- 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

- `cat` steht für **concatenate** (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 kann spaltenweise Text ausschneiden – Spalten sind wahlweise definierbar über
 - Zeichenpositionen
 - Trennzeichen (die logische Spalten voneinander trennen)

```

$ 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
    
```

c: character; zeichenbasiert
 d: delimiter; Trennzeichen
 f: field (Feld)

- 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
$ _
    
```

- fmt (format) bricht Textdateien um

```

$ 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.
    
```

keine Umbrüche

Zeilen-umbrüche

- Parameter -w75: Breite 75 (width)

- 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 Fahrradeder
2 Baeume
Wohnung
Haus
Hotel
Strasse
Allee

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

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

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

- sed (**Stream Editor**) 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/.../.../gi)
i: ignore case (s/.../.../gi)

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

- 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-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

- 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

<pre>\$ cat test5.txt Haus Die Hotels Hotels am Wasser Bau-Haus-Objekt Diese Zeile nicht</pre>	<pre>\$ grep 'H.*s' test5.txt Haus Die Hotels Hotels am Wasser Bau-Haus-Objekt</pre>	<pre>\$ sed 's/H.*s/HAUS/g' test5.txt HAUS Die HAUS HAUSer Bau-HAUS-Objekt Diese Zeile nicht</pre>
--	--	--

C-Compiler

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

- 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

- 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

- 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)

- 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

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;
```

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
```

• 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

- Struct und Pointer kombiniert
- Oft bei verketteten 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);
}
```

• 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
```


- 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
```

- 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;`

- 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)

- Funktionsprototypen (in Header-Dateien)

- erlauben die Verwendung von Funktionen, deren Implementierung weiter unten im Programm (oder in einer anderen Datei) steht
 - Prototyp enthält nur Rückgabotyp, Name und Argumente, z. B.
`int summe (int x, int y);`

- 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...

- 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)