Betriebssystem-Entwicklung mit Literate Programming

Foliensatz 2:
Crash-Kurse C / bash
zum Selbststudium

Wintersemester 2015/16

Dr. Hans-Georg Eßer

h.g.esser@gmx.de
http://ohm.hgesser.de/

v1.0, 06.10.2013

Selbststudium

- Diese Folien habe ich im Kurs „Systemprogrammierung Unix/Linux“ im Sommersemester 2013 eingesetzt
- Die Videos dazu finden sich unter

  http://ohm.hgesser.de/sp-ss2013/

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

Einführung in C (3)

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

Einführung in C (4)

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;
```


Einführung in C (5)

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
```


Einführung in C (6)

- 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);
}
```

Mehr zu C

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

Mehr zu C

- Funktionsprotoypen
  - 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);
```


Mehr zu C

- 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...

Mehr zu C

- 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)

Pointer

- Pointer-Typen
  - `typ *ptr;`
    → `ptr` ist ein Zeiger auf etwas von 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

Pointer

- 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
```


Pointer

- 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
```


Pointer

- 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;`

Einführung in die Bash

- Für Unix/Linux sind zahlreiche Shells (Kommandozeileninterpreter) verfügbar (C-Shell csh, Korn Shell ksh, Bash, tcsh)
- Linux-Standard-Shell: Bash („Bourne Again Shell“)
- im Vergleich mit Windows-Shells:
  - deutlich mächtiger als CMD.EXE (Command.com)
  - ganz anders zu bedienen als PowerShell

Shell-Prompt (1)

- Shell zeigt durch **Prompt** an, dass sie bereit ist, einen Befehl entgegen zu nehmen
- Prompts können verschieden aussehen:
  - … `$` _
    … `>` _ : Anwender-Prompt, nicht-privilegiert
  - … `#` _ : Root-Prompt, für den Administrator

Shell-Prompt (2)

- Vor dem `$`, `>`, `#` meist Hinweise auf Benutzer, Rechner, Arbeitsverzeichnis

```
[esser@macbookpro:SysPro]$
root@quad:~#
```

  - `esser`, `root`: Benutzername; individuell
  - `macbookpro`, `quad`: Rechnername
  - `SysPro`, `~`: Arbeitsverzeichnis, je nach Prompt-Einstellung auch in voller Länge (z. B. /home/esser/Daten/Ohm/SS2012/SysPro)
  - `~` = „Home-Verzeichnis“ des Benutzers

Befehlseingabe (1)

- Am Prompt Befehl eingeben und mit [Eingabe] abschicken
- Shell versucht, (in der Regel) erstes Wort als Kommandoname zu interpretieren:
  - Alias? (→ später)
  - Shell-interne Funktion? (→ später)
  - eingebautes Shell-Kommando? (z. B. `cd`)
  - externes Programm? (Suche in Pfad)

Befehlseingabe (2)

- Beispiel: Aktuelles **Arbeitsverzeichnis** anzeigen (`pwd` = **p**rint **w**orking **d**irectory)

```
[esser@quad:~]$ pwd
/home/esser
[esser@quad:~]$ _
```

- Nach Abarbeiten des Befehls (oft: mit einer „Antwort“) erscheint wieder der Prompt – Shell ist bereit für nächstes Kommando

Befehlseingabe (3)

- Mehrere Befehle auf einmal abschicken: mit Semikolon `;` voneinander trennen

```
[esser@quad:~]$ pwd; pwd
/home/esser
/home/esser
[esser@quad:~]$ _
```


Befehlseingabe (4)

- **Inhaltsverzeichnis** anzeigen: `ls` (**ls**t)
- bezieht sich immer auf das aktuelle Arbeitsverzeichnis (Alternative: Ort als Parameter angeben)

```
[esser@quad:~]$ ls
bahn-2011-02-22.pdf    bh-win-04-kret.pdf
buch_kap08.pdf         bv-anleitung.pdf
bz2.pdf
[esser@quad:~]$ ls /tmp
cvcd  kde-esser  ksocket-esser  orbit-esser
ssh-vrUNLb1418  virt_1111
[esser@quad:~]$ _
```


Befehlseingabe (5)

- Inhalt mit mehr Informationen: `ls -l`

```
[esser@quad:~]$ ls -l
-rw------- 1 esser users  29525 Feb 21  2011 bahn-2011-02-22.pdf
-rw-r--r-- 1 esser users 745520 Apr 10  2004 bh-win-04-kret.pdf
-rw-r--r-- 1 esser users 856657 Oct 21  2005 buch_kap08.pdf
-rw-r--r-- 1 esser esser 738570 Mar 17 20:29 bv-anleitung.pdf
-rw-r--r-- 1 esser users 123032 Sep 22  2003 bz2.pdf
[esser@quad:~]$ _
```

- Ausgabe enthält zusätzlich:
  - Zugriffsrechte (`-rw-r--r--` etc.) → später
  - Dateibesitzer und Gruppe (`esser`, `users`) → später
  - Größe und Datum/Zeit der letzten Änderung

Befehlseingabe (6)

- Leere Datei erzeugen (für Experimente): `touch`

```
[esser@quad:~]$ touch Testdatei
[esser@quad:~]$ ls -l Testdatei
-rw-r--r-- 1 esser esser 0 Apr  7 13:58 Testdatei
[esser@quad:~]$ _
```

- Datei hat Größe 0

Befehlseingabe (7)

- Fehlermeldungen: Unbekanntes Kommando

```
[esser@quad:~]$ fom
No command 'fom' found, did you mean:
 Command 'fim' from package 'fim' (universe)
 Command 'gom' from package 'gom' (universe)
 Command 'fop' from package 'fop' (universe)
 Command 'fdm' from package 'fdm' (universe)
 Command 'fpm' from package 'fpm2' (universe)
 [...]
fom: command not found
[esser@quad:~]$ _
```

- Meldung kann auch deutschsprachig sein

Dateiverwaltung (1)

Grundlagen (1)

- Linux kennt keine „Laufwerksbuchstaben“ (`C:`, `D:` etc.)
- Wurzelverzeichnis heißt `/`
- Pfadtrenner: auch `/` – d. h.: `/usr/local/bin` ist das Verzeichnis `bin` im Verzeichnis `local` im Verzeichnis `usr`. (wie bei Webadressen)

Dateiverwaltung (2)

Grundlagen (2)

- Weitere Datenträger erscheinen in Unterordnern
  - Beispiel: DVD mit Dateien hat Volume-Name `SYSPRO`
  - Datei `test.txt` auf oberster DVD-Verzeichnisebene ist als `/media/SYSPRO/test.txt` erreichbar (Windows: `e:\test.txt`)
  - Datei `Software/index.html` der DVD entsprechend als `/media/SYSPRO/Software/index.html` (Windows: `e:\Software\index.html`)

Dateiverwaltung (3)

Grundlagen (3)

- Für private Nutzerdaten hat jeder Anwender ein eigenes **Home-Verzeichnis**, das i. d. R. unterhalb von `/home` liegt, z. B. `/home/esser`.
- Die Tilde `~` ist immer eine Abkürzung für das Home-Verzeichnis
  - funktioniert auch in zusammengesetzen Pfaden
  - `~/Daten/brief.txt` statt `/home/esser/Daten/brief.txt`

Dateiverwaltung (4)

Grundlagen (4)

- Ausnahme: Das Home-Verzeichnis des Systemadministrators `root` ist nicht `/home/root`, sondern `/root`
- Der Trick mit der Tilde `~` funktioniert aber auch für `root`
- Warum? `/home` könnte auf einer separaten Partition liegen und bei einem Fehlstart nicht verfügbar sein

Dateiverwaltung (5)

Grundlagen (5)

- Zwei Spezialverzeichnisse in jedem Ordner
  - `..` ist das Verzeichnis eine Ebene tiefer (von `/usr/local/bin` aus ist `..` also `/usr/local`)
  - `.` ist das aktuelle Verzeichnis
- Pfade kann man **absolut** und **relativ** zusammen bauen
  - absoluter Pfad beginnt mit `/`
  - relativer Pfad nicht; er gilt immer ab dem aktuellen Arbeitsverzeichnis

Dateiverwaltung (6)

Verzeichnisnavigation

- Kommando `cd` (**c**hange **d**irectory) wechselt in ein anderes Verzeichnis
- Zielverzeichnis als Argument von `cd` angeben – wahlweis mit relativem oder absolutem Pfad

```
[esser@quad:~]$ pwd
/home/esser
[esser@quad:~]$ cd /home ; pwd
/home
[esser@quad:home]$ cd .. ; pwd
/
[esser@quad:/]$ _
```

Dateiverwaltung (7)

Datei kopieren

- Kommando `cp` (**cp**y) kopiert eine Datei
- Reihenfolge: `cp` *`Original Kopie`*

```
[esser@quad:tmp]$ ls -l
-rw-r--r--  1 esser  wheel  1501 Apr  5 11:37 test.dat
[esser@quad:tmp]$ cp test.dat kopie.dat
[esser@quad:tmp]$ ls -l
-rw-r--r--  1 esser  wheel  1501 Apr  8 12:17 kopie.dat
-rw-r--r--  1 esser  wheel  1501 Apr  5 11:37 test.dat
[esser@quad:tmp]$ _
```

! Kopie erhält aktuelles Datum/Zeit

Dateiverwaltung (8)

Datei umbenennen

- Kommando `mv` (**mv**e) benennt eine Datei um
- Reihenfolge: `mv` *`AltName NeuName`*

```
[esser@quad:tmp]$ ls -l
-rw-r--r--  1 esser  wheel  1501 Apr  5 11:37 test.dat
[esser@quad:tmp]$ mv test.dat neu.dat
[esser@quad:tmp]$ ls -l
-rw-r--r--  1 esser  wheel  1501 Apr  5 11:37 neu.dat
[esser@quad:tmp]$ _
```

! Umbenennen ändert Datum/Zeit nicht

Dateiverwaltung (9)

Datei verschieben

- Kommando `mv` (move) verschiebt eine Datei
- Reihenfolge: `mv` *`AltName NeuerOrdner/`*

```
[esser@quad:tmp]$ ls -l
-rw-r--r--  1 esser  wheel  1501 Apr  5 11:37 test.dat
[esser@quad:tmp]$ mv test.dat /home/esser/
[esser@quad:tmp]$ ls -l
[esser@quad:tmp]$ ls -l /home/esser/
-rw-r--r--  1 esser  wheel  1501 Apr  5 11:37 test.dat
   [...]
[esser@quad:tmp]$ _
```

! Verschieben ändert Datum/Zeit nicht

Dateiverwaltung (10)

Datei löschen

- Kommando `rm` (**rm**ove) löscht eine Datei

```
[esser@quad:tmp]$ ls -l
-rw-r--r--  1 esser  wheel  1501 Apr  5 11:37 test.dat
[esser@quad:tmp]$ rm test.dat
[esser@quad:tmp]$ ls -l
[esser@quad:tmp]$ _
```

Dateiverwaltung (11)

Mehrere Dateien

- Einige Befehle akzeptieren mehrere Argumente, z. B.
  - `mv` (beim Verschieben in anderen Ordner)
  - `rm`
- Beispiele:

```
[esser@quad:tmp]$ mv datei1.txt datei2.txt Ordner/
[esser@quad:tmp]$ rm datei3.txt datei4.txt datei5.txt
[esser@quad:tmp]$ _
```


Dateiverwaltung (12)

Wildcards (*, ?)

- Bei Befehlen, die mehrere Argumente akzeptieren, können Sie auch Wildcards verwenden:
  - `*` steht für beliebig viele (auch 0) beliebige Zeichen
  - `?` steht für genau ein beliebiges Zeichen
- Beispiele:

```
[esser@quad:~]$ ls -l ?????.pdf
-rw-r--r--  1 esser  staff   79737 Apr  2 01:18 RegA4.pdf
-rw-r--r--  1 esser  staff  132246 Apr  4 18:02 paper.pdf
[esser@quad:~]$ rm /tmp/*
[esser@quad:~]$ _
```


Befehle testen

- Löschbefehl mit Wildcards zu gewagt?
  → vorher mit `echo` testen:

```
[esser@quad:Downloads]$ echo rm *.zip
rm Logo_a5_tif.zip Uebung1.zip c32dwenu.zip
ct.90.01.200-209.zip ct.90.12.130-141.zip
ct.91.02.285-293.zip ct.91.12.024-025-1.zip
ct.91.12.024-025.zip ct.92.08.052-061.zip
ix.94.03.010-011.zip ix.94.07.068-071.zip
[esser@quad:Downloads]$ rm *.zip
[esser@quad:Downloads]$ _
```


Wildcard-Auflösung

- Das letzte Beispiel verrät etwas über das Auflösen der Wildcards
  - Wenn Sie `rm *.zip` eingeben, startet die Shell *nicht* `rm` mit dem Argument „`*.zip`“
  - Die Shell sucht im aktuellen Verzeichnis alle passenden Dateien und macht jeden Dateinamen zu einem Argument für den `rm`-Aufruf.
  - Es wird also
    `rm Logo_a5_tif.zip Uebung1.zip c32dwenu.zip ct.90.01.200-209.zip ...`
    aufgerufen.

Verzeichnisse (1)

Mit Verzeichnissen können Sie ähnliche Dinge tun wie mit Dateien

- Verzeichnis erstellen
- (leeres!) Verzeichnis löschen
- Verzeichnis umbenennen oder verschieben
- Verzeichnis rekursiv (mit allen enthaltenen Dateien und Unterordnern) löschen

Verzeichnisse (2)

Verzeichnis erstellen

- Kommando `mkdir` (**mk**e **dir**ectory) erzeugt ein neues (leeres) Unterverzeichnis

```
[esser@quad:tmp]$ ls -l
[esser@quad:tmp]$ mkdir unter
[esser@quad:tmp]$ ls -l
drwxr-xr-x  2 esser  wheel  68 Apr  8 14:28 unter
[esser@quad:tmp]$ cd unter
[esser@quad:unter]$ ls -l
[esser@quad:unter]$ cd ..
[esser@quad:tmp]$ _
```

! Kurzform `md` für `mkdir` nicht immer vorhanden → vermeiden

Verzeichnisse (3)

Verzeichnis löschen

- Kommando `rmdir` (**rm**ove **dir**ectory) löscht ein leeres (!) Unterverzeichnis

```
[esser@quad:tmp]$ touch unter/datei
[esser@quad:tmp]$ rmdir unter
rmdir: unter: Verzeichnis nicht leer
[esser@quad:tmp]$ rm unter/datei
[esser@quad:tmp]$ rmdir unter
[esser@quad:tmp]$ _
```

! Kurzform `rd` für `rmdir` nicht immer vorhanden → vermeiden

Verzeichnisse (4)

Verzeichnis umbenennen / verschieben

- funktioniert wie das Umbenennen / Verschieben von Dateien
- gleicher Befehl: `mv`, wieder zwei Varianten:
  - `mv Verzeichnis NeuerName`
  - `mv Verzeichnis AndererOrdner/`

Verzeichnisse (5)

Verzeichnis rekursiv löschen

- Kommando `rm` (**rem**ove) hat eine Option `-r` zum **r**ekursiven Löschen:

```
[esser@quad:tmp]$ mkdir a; mkdir a/b; mkdir a/b/c
[esser@quad:tmp]$ touch a/b/c/datei
[esser@quad:tmp]$ rmdir a
rmdir: a: Verzeichnis nicht leer
[esser@quad:tmp]$ rm -r a
[esser@quad:tmp]$ _
```

! Vorsicht beim rekursiven Löschen: „Was weg ist, ist weg“

~~Undelete~~

- Undelete = Löschen rückgängig machen
  - gibt es unter Linux nicht
  - Wiederherstellung von gelöschten Dateien mit Profi-Tools möglich, wenn Computer nach dem Löschen sofort ausgeschaltet wurde
  - solche Tools stellen aber sehr viele Dateien wieder her → enormer Aufwand, anschließend die gesuchte Datei zu finden; u. a. sind die Dateinamen dauerhaft verloren

→ vor `rm -r` … mehrfach prüfen …

Optionen und Argumente

- **Argumente:** z. B. Dateinamen; beziehen sich oft auf Objekte, die manipuliert werden sollen
- **Optionen:** verändern das Verhalten eines Befehls
  - bei den meisten Befehlen zwei Varianten:
  - kurze Optionen: `-a`, `-b`, `-c`, …
    → lassen sich kombinieren: `-abc` = `-a -b -c`
  - lange Optionen: `--ignore`, `--force`, `--all` etc.
  - Beispiel: `-r` bei `rm`

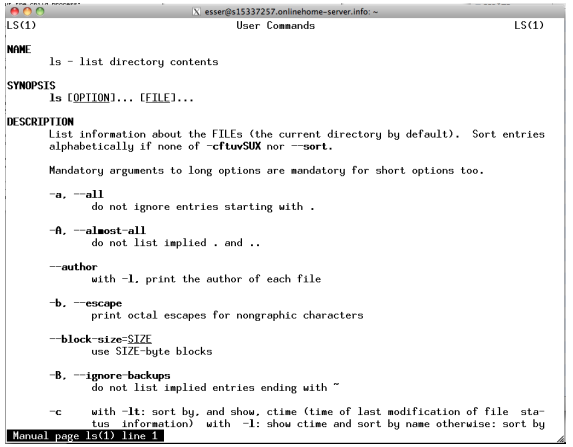
Hilfe: Handbuch

- Zu den meisten Kommandos gibt es eine sog. Manpage, die Sie über

  `man kommando`

  abrufen
- Beispiel:

  `man ls`



Der Editor vi (1)

- Standard-Editor auf allen Unix-Systemen (und damit auch Linux): `vi` (**vi**sual editor)
- gewöhnungsbedürftige Bedienung
- zwei Betriebsarten
  - **Befehlsmodus** (nach Start aktiviert; Normalmodus)
  - **Bearbeitungsmodus**
- `vi` aus Versehen gestartet? Verlassen ohne Speichern von Änderungen mit [Esc] `:q!`

Der Editor vi (2)

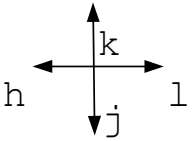
- Warum Umgang mit `vi` lernen?
  - auf jedem – noch so minimalistischen – Unix-System ist ein `vi` installiert (kleines Programm):

```
[esser@quad:~]$ ls -l /usr/bin/vi /usr/bin/emacs
-rwxr-xr-x 1 root root 5502096 Nov  9  2008 /usr/bin/emacs
-rwxr-xr-x 1 root root  630340 Oct 17  2008 /usr/bin/vi
```

  - läuft im Terminal → hilfreich bei Remote-Zugriff
  - Bei Problemen (Plattenfehler, nicht alle Datei-systeme verfügbar) sind andere Editoren evtl. nicht erreichbar, `vi` vielleicht doch → gilt leider nicht mehr für aktuelle Linux-Versionen

Der Editor vi (3)

- Wechseln in den Bearbeitungsmodus: `i`, `I`, `a`, `A`
  - `i`: Text vor dem Cursor einfügen
  - `a`: Text nach dem Cursor einfügen
  - `I`: Text am Zeilenanfang einfügen
  - `A`: Text am Zeilenende einfügen
- Bearbeitungsmodus verlassen: [Esc]
- Navigieren im Text: Cursortasten oder:



Der Editor vi (4)

- Zeichen / Text löschen:
  - im Bearbeitungsmodus mit [Rückschritt] und [Entf], wie aus anderen Editoren bekannt
  - im Befehlsmodus mehrere Möglichkeiten:
  - `x` löscht Zeichen unter Cursor
  - `X` löscht Zeichen links von Cursor
  - `dw` löscht ab Cursor-Position bis Anfang des nächstens Wortes
  - `dd` löscht aktuelle Zeile
  - vorab Zahl: Mehrfachausführung (`15dd`: 15 Zeilen)

Der Editor vi (5)

- Speichern und beenden
  - Immer zuerst in den Befehlsmodus
    → im Zweifelsfall einmal [Esc] drücken
  - Speichern: `:w`
  - Speichern (erzwingen): `:w!`
  - Beenden (klappt nur, wenn Text seit letztem Speichern nicht verändert wurde): `:q`
  - Beenden erzwingen (ohne speichern): `:q!`
  - Speichern und beenden: `:wq` (oder: `ZZ` ohne „:“)

Der Editor vi (6)

- Suche im Text
  - Vorwärtssuche: `/` und Suchbegriff, dann [Eingabe]
  - Sprung zum nächsten Treffer: `n` (next)
  - Rückwärtssuche: `?` und Suchbegriff, dann [Eingabe]
  - Sprung zum nächsten Treffer: `n`
  - Wechsel zwischen Vorwärts- und Rückwärtssuche: einfach `/` bzw. `?` , dann Eingabe und mit `n` weiter (in neuer Richtung) suchen

Der Editor vi (7)

- Rückgängig machen / wiederherstellen
  - Letzte Änderung rückgängig machen: `u` (undo)
  - geht auch mehrfach: `u`, `u`, `u`, …
  - … und mit Mehrfachausführung: `3u` macht die letzten drei Änderungen rückgängig
  - Einen Undo-Schritt aufheben: [Strg]+`r` `:redo`
  - mehrfaches Redo: z. B. `3` [Strg]+`r`

Der Editor vi (8)

- Copy & Paste: Kopieren …
  - `yw` (ab Cursorposition bis Wortende)
  - `y$` (ab Cursorposition bis Zeilenende)
  - `yy` (ganze Zeile)
  - `3yy` (drei Zeilen ab der aktuellen)
- … und Einfügen
  - `P` (fügt Inhalt des Puffers an Cursorposition ein)
- Cut & Paste
  - Löschen mit `dd`, `dw` etc.; dann einfügen mit `P`

Der Editor vi (9)

- Copy & Paste mit der Maus
  - Wenn Sie die grafische Oberfläche verwenden, geht es auch mit der Maus:
  - Kopieren: Mauszeiger auf 1. Zeichen, klicken (und gedrückt halten), zum letzten Zeichen ziehen, loslassen
  - Einfügen: Cursor zu Ziel bewegen, dann (im Einfügemodus!) die mittlere Maustaste drücken
  - Bei beiden Schritten muss man je nach `vi`-Version evtl. die [Umschalt]-Taste drücken

Der Editor vi (10)

- Datei im Editor öffnen:

  ```
  [esser@quad:~] vi Dateiname
  ```

- zweite Datei an Cursorposition hinzuladen:

  ```
  :read Dateiname
  ```

  (im Befehlsmodus!)

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**m**a**t) 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/OHM/g' test4.txt
Das OHM ist ein OHM, und mehrere
Woerter sind der Plural von OHM.
Ohne Woerter oder OHMe gibt es
keinen Satz - wir sind wortlos.
```

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

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
```