Linux Grundlagen

Workshop im Rahmen des Bachelor-Treffens

Georg Wassen

SoSe 2012

Übersicht

- Übersicht
- Arbeiten auf der Kommandozeile
- Shell-Skripte
- Werkzeuge
- Zusammenfassung

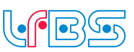
Kommandozeile

- Übersicht
- Arbeiten auf der Kommandozeile
  - Grundlagen
  - vi
  - praktische Übung
- Shell-Skripte
- Werkzeuge
- Zusammenfassung

Kommandozeile

Standard-Shell: `bash` (Bourne-again shell)
Alternativen (hier nicht behandelt):

- `sh` klassische Shell, Vorgänger der Bash
- `csh` C-Shell, C-ähnliche Skriptsprache
- `zsh` Z-Shell
- `ksh` Korn-Shell
- `fish` Friendly interactive Shell
- `dash` Debian/Ubuntu, ähnlich Bash
- ...



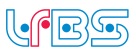
Bash konfigurieren

- /etc/profile
- /etc/profile.d/*.sh
- .bash_profile (Login-Shells)
- .bashrc (interaktive Shells) [1]

Vorlage:

```
cp /global/tmp/wassen2/bashrc ~/.bashrc
```

[1] Möglichkeit: bashrc führt bash_profile aus; dort alle Einstellungen



Kommandozeile – Grundlagen

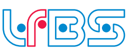
KDE KDE → Anwendungen → System → Terminal (Konsole)

Gnome Aktivitäten → Anwendungen → Systemwerkzeuge → Terminal

Text-Modus Konsolen 2 bis 6
(F1 oder F7 ist meist grafische Oberfläche)

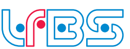
- aus grafischer Oberfläche: Strg + Alt + F2
- aus Text-Modus: Alt + F2
- zurück zum Grafik-Modus: Alt + F1
  (oder Alt + F7)

SSH Secure-Shell: `ssh drogo`



Kommandointerpreter

- allg. Form: `Kommando Parameter Parameter Parameter`
- Leerzeichen trennt
- Zeichenketten: `"..."` oder `'...'`
- Beispiele:
  - **`cd`** `src`
  - **`cd`** `..` beachte Leerzeichen, *nicht*: **`cd..`**
  - **`echo`** `Hello, World`
  - **`echo`** `"Hello,␣World"`

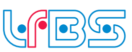


Kommandozeile – Grundlagen

TODO

Ausführen
Hintergrund: &
strg-Z, bg
fg
jobs
nohub

RWTH AACHEN UNIVERSITY



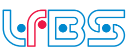
Sonderzeichen

Dollar (Variablen), < und >, Klammern, . . .

maskieren oder Zeichenkette

Unterschied einfache/doppelte Anführungszeichen:

- `"..."` interpretiert `$Variablen`
- `'...'` unverändert übernommen



wichtige Umgebungsvariablen

`$DISPLAY`	wo X-Fenster geöffnet werden sollen
`$EDITOR`	Standard-Editor
`$HOME`	Benutzerverzeichnis
`$HOSTNAME`	Rechnername
`$PATH`	Suchpfade für ausführbare Dateien
`$PS1`	Standard-Prompt, z. B. `"\u@\h:\w$"`
`$USER`	Name des Benutzers
`$LANG`	Sprache/Locale
`$IFS`	Internal Field Separator

```
$ echo $HOSTNAME
$ export PATH=~/bin:$PATH
```

Zugriffsrechte

```
$ ls -l
-rw-r--r-- 1 wassen team  140 10. Mai 15:25 obst
drwxr-xr-x 3 wassen team 4096 10. Mai 15:25 src
-rwxr-xr-x 1 wassen team 6549 10. Mai 16:13 warn
-rw-r--r-- 1 wassen team 1528 10. Mai 15:25 warn
```

r read (lesbar)

w write (schreibbar)

x executable (Dateien: ausführbar, Verzeichnisse: betretbar)

- 1. Zeichen: Typ (d=dir, l=link)
- 1. Dreiergruppe: Rechte des Besitzers (user)
- 2. Dreiergruppe: Rechte der Gruppe (group)
- 3. Dreiergruppe: Rechte für alle anderen (other)

Links

- (hard) Link: `ln obst.txt obst2.txt`
  - nur für Dateien
  - I-Node: nur gleiches Dateisystem
- Symlink: `ln -s obst.txt obst3.txt`
  - auch Verzeichnisse
  - auch andere Partition

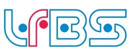
```
$ ls -l obst*
-rw-r--r-- 2 georg users 140 obst2.txt
lrwxrwxrwx 1 georg users   8 obst3.txt -> obst.t
-rw-r--r-- 2 georg users 140 obst.txt
```

Hilfe

- Parameter `--help` oder `-h`
- Manual-Pages: `man` *befehl*
- Suchen nach Befehlen: `apropos` *befehl*
- man-Sektionen: `man` *n* *befehl*
  - 1 : Kommandos und Programme
  - 2 : Systemaufrufe (C)
  - 3 : Standard-Bibliothek (C)
  - 7 : Konventionen (z. B. man 7 signal)

zum Beispiel:

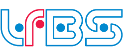
```
$ man 1 printf
$ man 3 printf
```



Kopieren und Einfügen mit der Maus

Auf der Konsole kann man meist mit der Maus kopieren und einfügen:

- Alles, was mit der Maus markiert wird,
  landet automatisch in einem Puffer
- Einfügen mit mittlerer Maustaste oder Rad
  (bzw. Emulation durch rechte und linke Taste gleichzeitig)

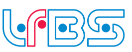


TODO

wichtige Befehle

- find
- ssh, scp
- ...

RWTHAACHEN UNIVERSITY

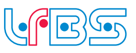


VIsual editor

- auf *jedem* UNIX-Rechner nutzbar
- auch über langsame Internetverbindungen (ssh)

die wichtigsten Modi:

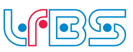
- Kommandomodus [Esc]
- Eingabemodus [i]
- visuell markieren: [v] oder [V]
- ex-Modus: [Q] , beenden mit: `vi` [Enter]



Kommandozeile – vi

Kommandomodus

- Cursor bewegen (Pfeiltasten oder [h] [j] [k] [l])
- Eingabe an Cursorposition: [i] (vor Cursor: [a]
  Zeilenanfang: [I] Zeilenende: [A])
  - Eingeben oder löschen
  - Beenden mit [Esc]
- Zeichen löschen: [x]
- Zeile löschen: [d] [d]
- Zeile kopieren: [y] [y]
- einfügen: [p] *oder* [P]
- letzte Änderung zurücknehmen: [u]
- Zahl vor Befehl: Wiederholen. Beispiele:
  3dd: drei Zeilen löschen
  2cw*neu* [Esc] : zwei Wörter durch „neu“ ersetzen

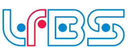


Tricks

- Wort überschreiben: `cw`
- Wort löschen: `dw`
- Zahl ++/– –: Strg + A / Strg + X
- letzten Befehl wiederholen: .
- ex-Modus: Suchen&Ersetzen mit RegEx
- Konfigurieren der CAPS-lock-Taste als zusätzliche ESC-Taste

Speichern/Beenden

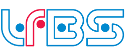
- Beenden: `:q` Enter
- Speichern: `:w` Enter
- Speichern und Beenden: `:wq` Enter
- Beenden ohne Speichern: `:q!` Enter
- Speichern unter anderem Namen: `:saveas` *`dateiname`* Enter
- Laden einer Datei: `:edit` *`dateiname`* Enter



Hilfe

`:help` Enter
Links folgen: Strg -]
beenden mit `:q` Enter

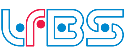
RWTH AACHEN UNIVERSITY



Arbeiten mit mehreren Dateien

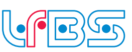
öffnen im neuen Tab: `:tabe` *`dateiname`* [Enter]
wechseln zwischen Tabs: [g] [t] / [g] [T]

RWTH AACHEN UNIVERSITY



Einstellungen

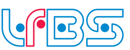
- `:set numbers` – Zeilennummern anzeigen
- `:set fileencoding?` – zeige Encoding an
- `:set fileencoding=utf-8` – ändere Encoding



Zusammenfassung

1. `vi` *dateiname*
2. Cursor positionieren: [↓], [→], ...
3. [i]
4. *Eingabe*
5. [Esc]
6. if not_done() goto 2
7. `:wq` [Enter]

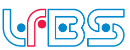
(praktische Übung folgt...)



Kommandozeile – praktische Übung

Nun seid Ihr dran. . .

RWTH AACHEN UNIVERSITY



Anleitung

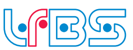
einzugebende Befehle:

```
$ ls
```

- $ ist Prompt-Zeichen,
- Befehl dahinter eingeben,
- Enter drücken

Ausgaben (Beispiel, i. d. R. gekürzt):

```
land.csv    obst.txt   sprachen.txt   src
stadt.csv   tiere
```



Kommandozeile – praktische Übung

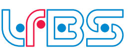
- öffne Terminalfenster

KDE	Anwendungen System Konsole
Gnome	Aktivitäten Anwendungen Systemwerkzeuge Terminal
Mac OS	Programme Systemprogramme Terminal
Windows	Cygwin

- SSH-Verbindung (z. B. `drogo`)
  (bei Bedarf VPN-Verbindung starten)

Linux	`ssh drogo`
Mac OS	`ssh drogo`
Windows	PuTTY

RWTH AACHEN UNIVERSITY



Vorlagendatei

```
$ wget http://www/users/global/workshop.tar.gz
```

- lädt Datei von Webserver[2]

```
$ tar xzf workshop.tar.gz
```

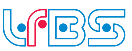
- TAR-Archiv entpacken:

x Dateien aus Archiv auspacken (extract)

z dekomprimieren (gzip-Format)

f lese Datei `workshop.tar.gz`

[2] außerhalb des Lehrstuhlnetzes:
`http://www.lfbs.rwth-aachen.de/users/global/workshop.tar.gz`



Eingaben auf der Kommandozeile

Die Bash verwendet **Readline**, so dass Eingaben bearbeitet werden können.

- Eingabe kann mit Cursortasten und [Del] [⟵] bearbeitet werden
- [Enter] – führt aktuelle Zeile aus
- \ – Befehl in folgender Zeile fortführen
- [⟵⟶] – vervollständigt Befehle, Programmnamen und viele Parameter
- [↑] [↓] – zurückrufen der letzten Eingaben (History)
- [Strg] + [R] – Rückwärtssuche in History
- [Strg] + [D] – EOF (End of File): Beenden
- [Strg] + [U] [Strg] + [K] [Strg] + [W] – Löschen

Emacs-Mode: die meisten Emacs-Befehle können in der Bash verwendet werden.

Man kann auch in den `vi`-Mode umschalten: `set -o vi`

Einstellungen

Wenn bisher keine `.bashrc`:

```
$ ls -a ~
$ cp bashrc.vorlage ~/.bashrc
```

öffne `~/.bashrc` in einem Editor (z. B.: `nano ~/.bashrc`)

Vorschläge:

- Prompt: `export PS1="\u@\h:\w\$ "`
- Bash-Mode vi: `set -o vi`
- Standard-Editor: `export EDITOR=vim`
- max. Core-Dump-Größe: `ulimit -c 400000`
- Alias für farbiges `ls`: `alias ls="ls --color=auto"`
- Alias für `ll`: `alias ll="ls -l"`
- Alias für farbiges `grep`: `alias grep="grep --color=auto"`

Hilfe

- Kommandozeilenparameter `-h` oder `--help`

```
$ ls --help
Aufruf: ls [OPTION]... [DATEI]...
Auflistung von Informationen der DATEIen (Standardvor
Verzeichnis). Alphabetisches Sortieren der Einträge,
noch --sort angegeben wurden.

Erforderliche Argumente für lange Optionen sind auch
  -a, --all                   Einträge, die mit . begi
  -A, --almost-all            implizierte . und .. nic
...
```

- man-Page

Kommandozeile – praktische Übung

Hilfe

- Kommandozeilenparameter `-h` oder `--help`
- `man`-Page

```
$ man ls
```

öffnet Manual in Pager (`less`)

- scrollen: ↑ ↓ (meist auch mit Maus-Rad)
- Beenden: q

Verzeichnisse

```
$ cd workshop
```

- wechsele in das Verzeichnis `workshop`

```
$ pwd
/home/wassen/workshop
$ ls
land.csv    obst.txt    src    stadt.csv
```

- Zeige Name und Inhalt des aktuellen Verzeichnisses

Erzeugen und Löschen

mkdir	erzeugt ein Verzeichnis:	`mkdir tmp`
rmdir	löscht ein leeres Verzeichnis:	`rmdir tmp`
touch	erzeugt eine leere Datei:	`touch testdatei`
file	analysiert Typ einer Datei:	`file testdatei`
rm	löscht eine Datei:	`rm testdatei`

Kommandozeile – praktische Übung

```
$ mkdir testverzeichnis
$ rmdir testverzeichnis
$ mkdir tmp
$ touch tmp/testdatei
$ file tmp/testdatei
tmp/testdatei: empty
$ file src/example.py
src/example.py: a /usr/bin/python script, ASCII
$ ls tmp/
$ rmdir tmp
rmdir: konnte "tmp" nicht entfernen:
Das Verzeichnis ist nicht leer
$ rm tmp/testdatei
```

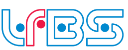
Kommandozeile – praktische Übung

Kopieren, Verschieben und Linken

Befehl	Beschreibung	Beispiel
`cp`	kopiert Dateien	`cp quelle ziel`
`mv`	verschiebt Dateien und Verzeichnisse	`mv datei dir/`
`ln`	erzeugt (hard) Links	`ln quelle ziel`
`ln -s`	erzeugt Symlinks	`ln -s quelle ziel`
`dirname`	Verzeichnisteil	`basename /usr/bin/bash`
`basename`	Dateiname	`basename /usr/bin/bash`

Kommandozeile – praktische Übung

```
$ cp obst.txt kopie.lst
$ ls
kopie.lst   obst.txt    tmp
$ mv kopie.lst kopie.txt            # umbenennen
$ ls
kopie.txt   obst.txt    tmp
$ mv kopie.txt tmp/                 # verschieben
$ ln obst.txt obst2.txt             # (hard) Link
$ ln -s obst.txt obst3.txt          # Symlink
$ ls -l
-rw-r--r-- 1 georg u. 140 kopie.txt
-rw-r--r-- 2 georg u. 140 obst2.txt
lrwxrwxrwx 1 georg u.   8 obst3.txt -> obst.txt
-rw-r--r-- 2 georg u. 140 obst.txt
```



Muster und Platzhalter

`*`	steht für beliebig viele Zeichen
`?`	steht für genau ein Zeichen
`[a-z]`	steht für eines der Zeichen
`{txt,csv}`	steht für eine der Varianten

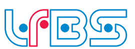
```
$ ls [a-m]*
$ ls *.{txt,csv}
$ cp *.[ch] tmp/
$ echo *.txt
```

- Auflösung der Dateimuster vor Ausführung des Befehls

Kommandozeile – praktische Übung

Arbeiten mit Textdateien

cat	Dateien (hintereinander) ausgeben: `cat datei1 datei2`
less	Datei im Pager ausgeben
head	Anfang einer Datei ausgeben
tail	Ende einer Datei ausgeben
sort	Sortieren
cut	Spalten ausschneiden
paste	Dateien nebeneinander (als Spalten) ausgeben
column	Dateien in Spalten ausgeben
join	Dateien nach Schlüsseln kombinieren
grep	Suchen nach Zeilen
sed	Stream Editor: systematische Änderungen durchführen
awk	Programmiersprache für Datei-/Datenbearbeitung



Kommandozeile – praktische Übung

```
$ cat sprachen.txt
...
$ less sprachen.txt
```

Bedienung des Pagers `less` (auch für Manual-Pages)

- scrollen: [↑] [↓]
- Anfang: [g] Ende: [G]
- Suche: /*begriff* [Enter]
- nächste Fundstelle: [n] vorige: [N]
- Beenden: [q]

Kommandozeile – praktische Übung

```
$ head sprachen.txt
$ tail -n 5 sprachen.txt
$ sort obst.txt
$ cut -d, -f 1,3 stadt.csv
$ paste obst.txt stadt.csv
$ join -t, stadt.csv land.csv
$ grep Aachen stadt.csv
$ sed 's/,/:/g' stadt.csv
$ awk -F, '{print $2, $1}' stadt.csv
```

Kommandozeile – praktische Übung

```
$ grep -n beere obst.txt
```

Kommandozeile – praktische Übung

```
$ grep -n beere obst.txt
```

Reguläre Ausdrücke:

```
$ grep '^B' obst.tex
$ grep 'ne$' obst.tex
$ grep '\([aeiou]\).*\1' obst.txt
```

siehe: `man 7 regex`, [3]

Kommandozeile – praktische Übung

Ausgabeumleitung

- `>` Umleitung der Standardausgabe in Datei
  `sort obst.txt > obst_sorted.txt`
- `>>` Anhängen an Datei
- `<` Lesen der Standardeingabe aus Datei
  `sort < obst.txt`
- `2>` Umleitung der Standard-Fehlerausgabe
- `2>&1` Fehlerausgabe in Standardausgabe umleiten
  `svn export http://svn/svn/texte > svn.log 2>&1`
- `/dev/null` „Papierkorb“
- `|` Pipe: Ausgabe in Eingabe weiterleiten
- `tee` Schreibt (Zwischenergebnis) in Datei und in Standardausgabe

Kommandozeile – praktische Übung

```
$ ls -l > dateien.lst
$ gcc -c src/warning.c 2> /dev/null
$ echo Melone >> obst.txt
$ join -t, stadt.csv land.csv
$ ... | grep NRW
$ ... | sort -t, -k2 -nr
$ ... | cut -d, -f1,2
$ ... | column -s, -t
$ grep -r alles tiere/ | tee allesfresser.txt
$ cat allesfresser.txt
```

Hier: ... bedeutet eine Fortsetzung der vorigen Zeile;
↑, dann Erweiterung anhängen.

Kommandozeile – praktische Übung

Arbeiten mit Dateien

`type`	Ist es Kommando/Alias/Funktion/...?
`which`	Welche Datei wird ausgeführt
`file`	Dateityp anzeigen
`find`	Suchen von Dateien
`tree`	Verzeichnisstruktur als Baum anzeigen
`stat`	Dateiattribute anzeigen

Kommandozeile – praktische Übung

```
$ type test
$ which test
$ file /usr/bin/test
$ file src/warning.c
$ file warning.o
$ find . -name tiger
$ tree tiere
$ tree -d tiere
$ stat obst.txt
```

Kommandozeile – praktische Übung

Rechte ändern: `chmod <wer><operation><was> <Datei>`

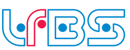
- Wer: ugoa – user/group/other/all
- Operation: +-= – add/remove/set right to
- Was: rwx – read/write/execute

```
$ ls -l obst.txt
-rw-r--r-- 1 wassen team 140 10. Mai 15:25 obst.
$ chmod g+w obst.txt
$ ls -l obst.txt
-rw-rw-r-- 1 wassen team 140 10. Mai 15:25 obst.
$ chmod go-rw obst.txt
$ ls -l obst.txt
-rw------- 1 wassen team 140 10. Mai 15:25 obst.
```

Verzeichnisse: Schreibrecht für bestimmte Dateioperationen nötig

Besitz ändern: `chown user.group dateiname`

Standard für neue Dateien ändern: **`umask`**



Komprimieren

Kompression von Dateien

- gz: gzip, gunzip, zcat
- bz2: bzip2, bunzip2, bzcat
- xz: xz, unxz, xzcat

Zusammenfügen von Dateien in ein Archiv: tar

```
$ tar czf archiv.tar.gz <dateien>
$ tar xzf archiv.tgz
$ tar xjf archiv.tar.bz2
```

Verarbeiten weiterer Formate

- zip: Verarbeiten von ZIP-Archiven
- unrar: Entpacken von RAR-Archiven

Kommandozeile – praktische Übung

Administration

Befehl	Beschreibung	Beispiel
su	Substitute User (benötigt `root`-Passwort)	
du	Disk Usage	`du -sh`
df	Disk Free	`df -h`
mount	Blockgerät einhängen (nur mit `root`-Rechten)	
free	freien Speicher anzeigen	
uname	Kernel-Info	`uname -a`
uptime	Laufzeit und Auslastung	
ps	Liste der Prozesse	`ps aux`
top	Prozessmanager	
kill	Signal senden (Standard: SigTerm, also Prozess beenden)	
ifconfig	Netzwerk-Infos	

fortgeschrittene Tricks

Kann man eine Datei mit Namen `-l` (Minus L) oder einem Leerzeichen anlegen und wieder löschen?

- in Anführungszeichen ist fast jeder Name möglich (kein /)
- `--` markiert Ende der Parameter

```
$ touch ' '
$ touch -- -l
$ ll
$ rm ' '
$ rm -- -l
```

Vorsicht bei so was:

```
-rw-r--r-- 1 wassen team   140 obst.txt
-rw-r--r-- 1 wassen team     0 -rf *
-rw-r--r-- 1 wassen team  3741 sprachen.txt
```

Kommandozeile – praktische Übung

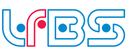
vim

- Öffnet Datei in vim: `vim obst.txt`
- Bewege Cursor ans Dateiende: `G`
- Bewege Cursor drei Zeilen nach oben: `3k`
- Füge Text ein: [i] `Holunderbeere` [Enter] [Esc]
- Suche „Orange“: [/] Orange [Enter]
- Füge Zeile darunter ein: [o] Grapefruit [Esc]
- Speichern und Beenden: `:wq` [Enter]
- alternativ Speichern und Beenden: `ZZ`
- Hilfe: `:help` [Enter]

Emacs

- Öffne Datei in Emacs: `emacs obst.txt`
- Bewegen mit Cursortasten: ↓ → ...
- Eingabe/Änderung
- Speichern: Strg + x , Strg + s
- Beenden Strg + x , Strg + c

Übliche Notation für Emacs-Shortcuts: `C-x C-s`



Nano

Sehr einfacher Editor, Shortcuts werden unten im Programm angezeigt.

Literatur

- Shelley Powers, Jerry Peek, Tim O'Reilley, Mike Loukides.
  Unix Power Tools.
  O'Reilley, 2003.
- Nicholas Petreley, Jono Bacon.
  Linux Desktop Hacks.
  O'Reilley, 2005.

Literatur

Martin Dietze.
Praxiskurs Unix-Shell.
O'Reilley, 2011.

Michael Kofler.
Linux Kommando-Referenz: Shellbefehle von a2ps bis zypper.
Addison-Wesley, 2010.

Literatur

GNU Project.
Bash Reference Manual.
http://www.gnu.org/software/bash/manual/

Simon Myers.
Power Shell Usage: Bash Tips & Tricks.
http://www.ukuug.org/events/linux2003/papers/bash_tips/

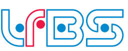
Zed A. Shaw.
Learn Regex The Hard Way.
http://regex.learncodethehardway.org/

Literatur

- Giles Orr.
  Learn Regex The Hard Way.
  http://www.tldp.org/HOWTO/Bash-Prompt-HOWTO/
- Pixelbeat.
  Linux Command Line Reference.
  http://www.pixelbeat.org/cmdline.html
  http://www.pixelbeat.org/docs/linux_commands.html
- C. J. Rennie.
  A very brief guide to Linux.
  http://www.cheat-sheets.org/saved-copy/A.very.brief.guide.to.Linux.htm

Shell-Skripte

- Übersicht
- Arbeiten auf der Kommandozeile
- Shell-Skripte
  - Grundlegende Techniken
  - SSH mit Schlüsselpaar
  - praktische Übung
- Werkzeuge
- Zusammenfassung



Shell-Skripte

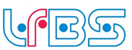
Skript: eine Datei mit (Shell-)Befehlen

- komplexe Befehle mit vielen Parametern
- ausgeklügelte Pipes
- ganze Programme

Skripte ausführen

Aufruf Interpreter mit Eingabedatei:

```
$ bash my_first_script.sh
```



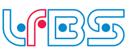
Skripte ausführen

Aufruf Interpreter mit Eingabedatei:

```
$ bash my_first_script.sh
```

Als ausführbare Datei:

```
$ cat my_first_script.sh
#!/bin/bash
echo "Hello world!"
$ chmod ug+x my_first_script.sh
$ ls -l my_first_script.sh
-rwxr-xr-- 1 wassen team 32  my_first_script.sh
$ ./my_first_script.sh
```



She-Bang

- x-Flag in Berechtigung: ausführbar
- 1. Zeile: `#!` „She-Bang“
- dahinter: Interpreter für dieses Skript
  - /bin/bash
  - /usr/bin/python
  - /usr/bin/perl
  - /usr/bin/gnuplot

Umgebung

- Skript wird in Sub-Shell ausgeführt
  - Umgebung wird vererbt
  - export: Variable wird Teil der Umgebung
  - in Skript geänderte Variablen nachher für Aufrufer nicht sichtbar
- Skript „sourcen“:

```
$ source my_first_script.sh
```

Befehle der Datei werden in aktueller Shell ausgeführt

  - nötig, wenn Umgebungsvariablen gesetzt werden sollen!

Variablen

- Zuweisung mit =, keine Leerzeichen!
- Referenzierung mit `$VARIABLE` oder `${VARIABLE}`
- Konvention: Großbuchstaben (aber nicht nötig)
- Zeichenketten mit `"..."` (interpretiert Variablen)
- oder `'...'` (interpretiert nicht)
- interpretiert bei Zuweisung

Kommandozeilenparameter

`$0`	Programmname
`$1 ... $9`	die ersten 9 Parameter
`$#`	Anzahl der Parameter
`shift`	Verwirft $1 und schiebt Parameter nach links

Standardausgabe

Standardausgabe eines Befehls in Variable speichern:

```
FILES=`ls`
OUTPUT=$(benchmark -s 128)
```

Die Backticks sind klassische Syntax,
$(...) lässt sich auch schachteln

Rechnen

```
A=3
B=4
C=$(( A*A + B*B ))
```

Achtung: nur Integer-Rechnung.

Rechnen

```
A=3
B=4
C=$(( A*A + B*B ))
```

Achtung: nur Integer-Rechnung.
`bc` (basic calculator)

```
C=$(echo "sqrt␣(␣$A*$A␣+␣$B*$B␣)" | bc -l)
pi=$(echo "scale=10;␣4*a(1)" | bc -l)
```

`scale` Anzahl Nachkommastallen

`-l` nutze Mathe-Bibliothek

`a()` Arcus Tangens

Hinweis: Shell-Variable `pi` ist Zeichenkette

Tastatureingabe

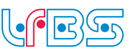
`echo -n` unterdrückt Zeilenumbruch

`read V` lese Eingabe in Variable `V`

`read` warte nur auf Enter

`read A B C` lese drei Wörter in die Variablen

- Trennzeichen kann mit `$IFS` geändert werden
- weitere Wörter in letzter Variable

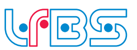


Kontrollstrukturen

- if/then/else
- case-Fallunterscheidung
- while-Schleife
- for-Schleife

if

```
if [ $# -lt 2 ]
then
    echo "zu␣wenige␣Parameter"
fi
```

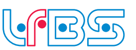


if

```
if [ $# -lt 2 ]
then
    echo "zu␣wenige␣Parameter"
fi
```

Mit ; können mehrere Befehle in einer Zeile geschrieben werden:

```
if [ $# -lt 2 ]; then
    echo "zu␣wenige␣Parameter"
fi

if [ $# -lt 2]; then echo ja; fi
```



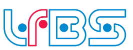
test

```
[ $COUNT -lt 10 ]
```

ist abgekürzte Schreibweise für

```
test $COUNT -lt 2
```

Rückgabewert 0=true 1=false
Achtung: Teile der Bedingung sind Parameter. Leerzeichen!
Auf Kommandozeile: $? enthält Rückgabe des letzten Befehls



Möglichkeiten von test

-lt -gt -le	numerisch: less-than, greater-than, less-or-equal
= !=	Zeichenketten: gleich, ungleich
-z	String ist leer (Länge 0)
-a -o	and, or
-r -w	Datei ist lesbar, schreibbar

viele weitere; siehe: `man test`

if, elif, else

```
if [ $# -lt 2 ]; then
    echo "zu wenige Parameter"
    exit
elif [ $# -gt 2 ]; then
    echo "zu viele Parameter"
    exit
else
    INPUT=$1
fi
```

neue Sytax für if

```
if [[ $# -ge 1 ]]; then echo ja; fi
```

Auswertung von Bash selber; kein Subprozess für test gestartet.
Regex:

```
if [[ :$PATH: =~ :([a-z/]*script[a-z/]*): ]]
then
    echo ja: ${BASH_REMATCH[1]};
fi
```

Regex nicht in Anführungszeichen! (seit Bash 3.2)

case

```
case $INPUT in
    start)
        echo starting process
        ;;
    stop)
        echo killing process
        ;;
    *)
        echo invalid command: $INPUT
        ;;
esac
```

Wie `fi` ist esac Rückwärts für case

for

```
for I in *.txt; do echo $I; done
for I in 1 2 3 4; do echo $I; done
for I in $(seq 1 10); do echo $I; done
for I in {1..10}; do echo $I; done
```

for

```
for I in *.txt; do echo $I; done
for I in 1 2 3 4; do echo $I; done
for I in $(seq 1 10); do echo $I; done
for I in {1..10}; do echo $I; done
```

es gibt auch eine an C angelehnte Variante:

```
for (( I=1; I<=256; I*=2)); do echo $I; done
```

while

```
while read F; do
    echo $F
done
```

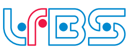
while

```
while read F; do
    echo $F
done
```

Außerdem:

`break` verlässt Schleife

`continue` beginnt nächsten Durchlauf



Funktionen

```
times_two () {
    echo $(( $1 * 2 ))
}
```

- Klammern nur Zierde (keine Parameter dort)
- Parameter in $1 etc.
- Aufruf wie jedes Programm

```
$ time_two 5
10
```

nützliche Helfer

`seq`	erzeugt Sequenzen	`seq 1 10`
`printf`	wie in C	
`pushd`/`popd`	Verzeichnisstack	**`pushd`** `/tmp` **`popd`**
`select`	Auswahlmenü	

Variablensubstitution

```
${#VAR}                      # length of $VAR
${VAR:start:length}          # substr
${VAR/pattern/string}        # replace
${VAR[1]}                    # Array-Zugriff
```

alle Möglichkeiten: siehe [2]

ssh

Erzeuge Schlüsselpaar:

```
$ ssh-keygen
```

öffentlichen Schlüssel auf Systemen installieren:

```
local$ scp ~/.ssh/id_dsa.pub remote:.ssh/tmp.pub
local$ ssh remote
remote$ cd .ssh
remote$ cat tmp.pub >> authorized_keys
remote$ rm tmp.pub
remote$ exit
local$
```

Im Lehrstuhlnetz (gleiches Home-Verzeichnis):

```
cat .ssh/id_dsa.pub >> .ssh/authorized_keys
```

ssh ohne Passwort

Wenn beim Erzeugen Passphrase angegeben:
starte ssh-agent

```
$ ssh-agent bash
$ ssh-add
Enter passphrase:
Identity added: /home/wassen/.ssh/id_dsa (/home/
```

nun SSH-Login ohne weitere Frage möglich.

ssh ohne Passwort

Wenn beim Erzeugen Passphrase angegeben:
starte ssh-agent

```
$ ssh-agent bash
$ ssh-add
Enter passphrase:
Identity added: /home/wassen/.ssh/id_dsa (/home/
```

nun SSH-Login ohne weitere Frage möglich.

```
local$ ssh -A remote
remote$ ssh server
```

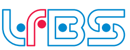
leite Agent weiter für weiteren Login von dort

Shell-Skripte – praktische Übung

Anleitung

- die meisten Skripte in Vorlagendatei
  - Name mit Verzeichnis angegeben
- viele Übungen auch direkt auf Kommandozeile
  - Befehle hinter $ eingeben

RWTH AACHEN UNIVERSITY



Shell-Skripte – praktische Übung

Vorlagendatei

```
$ wget http://www/users/global/workshop.tar.gz
```

- lädt Datei von Webserver[3]

```
$ tar xzf workshop.tar.gz
$ cd workshop/
```

- TAR-Archiv entpacken und ins Verzeichnis wechseln

[3] außerhalb des Lehrstuhlnetzes:
http://www.lfbs.rwth-aachen.de/users/global/workshop.tar.gz

Skript ausführen

my_first_script.sh:

```
#!/bin/bash
echo "Hello,␣world!"
```

Aufrufe:

```
$ bash my_first_script.sh
Hello, world!
$ ls -l my_first_script.sh
-rwxr-xr-- 1 wassen team 32  my_first_script.sh
$ ./my_first_script.sh
Hello, world!
```

weitere Ausgaben

(erweitere my_first_script.sh)

```
#!/bin/bash
echo "Hello, world!"
echo $PATH
echo -n Du bist im Verzeichnis
pwd
```

- PATH ist Umgebungsvariable
- **echo** -n unterdrückt Zeilenumbruch
- Programm pwd wird einfach ausgeführt

Verwendung von Variablen

```
#!/bin/bash

GRUSSFORMEL="Mit freundlichen Grüßen, $USER"

echo $GRUSSFORMEL
echo "$GRUSSFORMEL"
echo '$GRUSSFORMEL'
```

Shell-Skripte – praktische Übung

Standardausgabe in Variable speichern:

```
#!/bin/bash

DATUM=`date "+%d.%m.%Y"`                # Backti
TEMP=$(mktemp $(date +%Y%m%d).XXXXXX)   # so: sc

echo Heute ist der $DATUM
echo $TEMP
```

Gültigkeitsbereich von Variablen

betrachte `src/scope.sh` und `src/subscope.sh`

```
$ cd src/ ; ./scope.sh
scope before calling subscope: LOCAL=1 EXP=1
subscope before changing:      LOCAL= EXP=1
subscope after changing:       LOCAL=2 EXP=2
scope after calling subscope:  LOCAL=1 EXP=1
```

- nicht exportierte Variable nicht an Unterprogramm
- exportierte Variable an Unterprogramm
- beide nicht zurück

Gültigkeitsbereich von Variablen

betrachte `src/scope.sh` und `src/subscope.sh`

```
$ cd src/ ; ./scope.sh
scope before calling subscope: LOCAL=1 EXP=1
subscope before changing:      LOCAL= EXP=1
subscope after changing:       LOCAL=2 EXP=2
scope after calling subscope:  LOCAL=1 EXP=1
```

- nicht exportierte Variable nicht an Unterprogramm
- exportierte Variable an Unterprogramm
- beide nicht zurück

```
$ echo $LOCAL $EXP
$ source ./scope.sh
$ echo $LOCAL $EXP
```

Shell-Skripte – praktische Übung

Kontrollstrukturen

`if`- und `for`-Konstrukte können einfach in der Kommandozeile getestet (und schrittweise erweitert) werden:

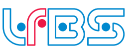
```
$ A=5
$ if [ $A -gt 3 ]; then echo ja; else echo nein; fi
$ for I in *.txt; do echo $I; done
$ for (( I=1; I<=10; I++)); do
    printf 'Datei%02u.txt\n' $I
  done
```

test

[ist ein Synonym für **test**:

```
$ echo $UID
1000
$ [ $UID -ge 1000 ]
$ echo $?
0
$ test $UID -ge 1000
$ echo $?
0
```

Erinnerung: **return** 0; bedeutet SUCCESS.



test

[ist ein Synonym für **test**:

```
$ echo $UID
1000
$ [ $UID -ge 1000 ]
$ echo $?
0
$ test $UID -ge 1000
$ echo $?
0
```

Erinnerung: **return** 0; bedeutet SUCCESS.

if auch mit anderen Programmen:

```
$ if grep 'Apfel' obst.txt; then echo ja; fi
```

Tipp: oft ist es sinnvoll, die Ausgabe umzuleiten, z. B. nach /dev/null

Parameter

src/parameter.sh

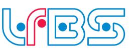
```
#!/bin/bash
FIRST=$1
shift
while [ $1 ]; do
    echo $FIRST: $1
    shift
done
```

Aufruf z. B. `src/parameter.sh abc d e f`

Parameter 2: getopts

src/getopt.sh

```
#!/bin/bash
while getopts ":ac:" opt; do
  case $opt in
    a)
      echo "-a␣was␣triggered!"
      ;;
    c)
      echo "-c␣was␣set␣to␣$OPTARG"
      ;;
    \?)
      echo "Invalid␣option:␣-$OPTARG"
      ;;
  esac
done
```



Funktionen

src/fakultaet.sh (Aufruf mit **source**!)

```
fakultaet() {
  if [[ $1 -lt 1 ]]; then
    echo 1
  else
    echo $(( $1 * $( fakultaet $(( $1-1 )) ) ))
  fi
}
```

Aufruf:

```
$ source src/fakultaet.sh
$ fakultaet 5
```

Rechnen

```
$ echo $(( 23 * 42 ))
$ for I in {1..8}; do echo $(( 2**I )); done
$ bc
-> i=2.5
-> 3*i+1.2
8.7
-> ^D
$ R=5; echo "$R*2*3.14152" | bc
```

Tipp: Die meisten Eingabeaufforderungen können mit Strg + D (End-of-File, EOF) beendet werden.

Tastatureingabe

src/input.sh

```
#!/bin/bash
echo -n "Wie␣heißt␣Du?␣"
read NAME
echo Hallo, $NAME
```

Tastatureingabe

src/input.sh

```
#!/bin/bash
echo -n "Wie␣heißt␣Du?␣"
read NAME
echo Hallo, $NAME
```

Nachfrage: „Sind Sie sicher?“

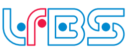
```
echo "Fortsetzen␣mit␣ENTER,␣Abbruch␣Strg-C"
read
echo "fortgesetzt."
```

select Auswahlmenü

src/select.sh

```
#!/bin/bash
PS3='Auswahl: '
select VAR in A B C Exit; do
    echo REPLY=$REPLY VAR=$VAR
    if [ "$VAR" = Exit ]; then
        break
    fi
done
```

Hinweis: auch Strg + D beendet das Menü
Achtung: "$VAR" hier in Anführungszeichen, da andere Auswahlen mehrere Wörter enthalten können.



Shell-Skripte – praktische Übung

Signalhandler

src/signal.sh

```
#!/bin/bash

sig_int_handler() {
    echo "got␣Signal␣SIGINT."
    exit 0
}
trap sig_int_handler INT

while /bin/true; do
    echo -n "."
    sleep 1
done
```

Hinweis: Das Signal kann auch mit **kill** geschickt werden.

Shell-Skripte – praktische Übung

Quellverzeichnis

Ausführung immer im Aufrufverzeichnis.
Wie das Verzeichnis bekommen, wo Skript gespeichert ist?
(z. B. weil dort eine Konfigurationsdatei liegt)
`src/readconfig.sh`:

```
#!/bin/bash
SRCDIR=$(dirname $(which $0))
source $SRCDIR/config.sh
echo $CONFIG
```

Prüfen, ob mit root-Rechten aufgerufen

- $UID enthält User-ID
- 0 ist immer root

```
if [ $UID -ne 0 ]; then
    echo 'Call this script as root'
    exit 1
fi
```

Temporäre Datei erzeugen

```
$ mktemp
/tmp/tmp.gqxopVIxnL
TEMP=$(mktemp dump-XXXXXX)
create_dump > $TEMP
```

- Datei wird leer erzeugt
- garantiert neue Datei mit einzigartigem Namen

Shell-Skripte – praktische Übung

Datei/Verzeichnisnamen aus Uhrzeit erzeugen

date gibt Systemzeit aus.

```
$ date
So 3. Jun 21:43:47 CEST 2012
$ date +%Y-%m-%d_%H-%M-%S
2012-06-03_21-44-24
$ LOGFILE=log.$(date +%Y-%m-%d_%H-%M-%S)
$ date >> $LOGFILE
```

Tipp: Bei langen Benchmarks vorher und hinterher date speichern

Datei zeilenweise lesen

read von stdin
in Schleife Eingabeumleitung verwenden:

```
while read LINE; do
    echo "Hmmm, $LINE mit Sahne"
done < obst.txt
```

Datei zeilenweise lesen

read von stdin
in Schleife Eingabeumleitung verwenden:

```
while read LINE; do
    echo "Hmmm,␣$LINE␣mit␣Sahne"
done < obst.txt
```

Auch für mehrere Token pro Zeile:

```
IFS=,
while read STADT EINW FL; do
    echo $STADT hat $(( EINW / FL )) Einw./km^2
done < stadt.csv | sort -rnk3
```

Shell-Skripte – praktische Übung

TODO

- Filehandles (öffnen, ändern, . . .), auch Sockets
  - siehe: `exec`
- nohup
  - no hangup: führe Programm nach Logout weiter aus
- . . .

Beispiele

- `src/coreinfo.sh`
  - Aufbereitung von Datei aus `/sys/devices/system/cpu`
- ...

Literatur

- Mendel Cooper.
  Advanced Bash-Scripting Guide
  http://www.tldp.org/guides.html
- Patrick Ditchen.
  Shell-Skript-Programmierung.
  mitp, 2011.

Literatur

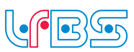
- Steve Parker.
  UNIX/Linux Shell Cheat Sheet.
  http://steve-parker.org/sh/cheatsheet.pdf
- Heiko Schlittermann.
  SSH ohne Passwort.
  http://www.schlittermann.de/doc/ssh
- A.P. Lawrence.
  Getopt and getopts.
  http://aplawrence.com/Unix/getopts.html

Literatur

GNU Project.
Bash Reference Manual.
http://www.gnu.org/software/bash/manual/

Mendel Cooper.
Advanced Bash-Scripting Guide
http://www.tldp.org/guides.html

- Übersicht
- Arbeiten auf der Kommandozeile
- Shell-Skripte
- Werkzeuge
  - Übung 1: Kompilieren und Linken
  - Übung 2: Makefiles
  - Übung 3: weitere Binutils
  - Übung 4: Fehlersuche
  - Übung 5: Subversion
  - Übung 6: GDB
- Zusammenfassung



Werkzeuge

Alle Werkzeuge sind frei verfügbar.
Zur Vertiefung empfehlen sich die Manual-Pages und Tutorials im Internet.

Musterprogramme:

```
$ wget http://www/users/global/workshop.tar.gz
```

- lädt Datei von Webserver[4]

```
$ tar xzf workshop.tar.gz
$ cd workshop/
```

- TAR-Archiv entpacken und ins Verzeichnis wechseln

[4] außerhalb des Lehrstuhlnetzes:
http://www.lfbs.rwth-aachen.de/users/global/workshop.tar.gz

Übung 1:

Kompilieren und Linken

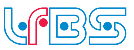
src/hello/main.c

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

static void greetings(void)
{
    printf("Hello,␣world!\n");
}

int main(int argc, char *argv[])
{
    greetings();
    return EXIT_SUCCESS;
}
```

1.1 Grundlagen

- wechsle in Verzeichnis `src/hello`
- Kompilieren einzelner Dateien:

```
$ gcc -o hello main.c
$ ./hello
```



1.1 Grundlagen

- wechsle in Verzeichnis `src/hello`
- Kompilieren einzelner Dateien:

```
$ gcc -o hello main.c
$ ./hello
```

- Untersuchen der erzeugten Datei

```
$ file hello
$ strings hello
$ nm hello
```

BTW: findet jemand printf?

1.2 Optimierung

- übersetze mit verschiedenen Optimierungs-Einstellungen:

```
$ gcc -o hello0 -O0 main.c
$ gcc -o hello2 -O2 main.c
$ gcc -o helloS -Os main.c
```

- vergleiche Größen:

```
$ size hello?
```

Hinweis: sehr primitives Programm, wenig Optimierungsbedarf.

1.2 Optimierung

- übersetze mit verschiedenen Optimierungs-Einstellungen:

```
$ gcc -o hello0 -O0 main.c
$ gcc -o hello2 -O2 main.c
$ gcc -o helloS -Os main.c
```

- vergleiche Größen:

```
$ size hello?
```

Hinweis: sehr primitives Programm, wenig Optimierungsbedarf.

- gebe Symbole aus:

```
$ nm hello0
$ nm hello2
```

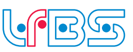
Hinweis: beachte Symbol `greetings`!

1.3 Make

Make kann das Programm mit impliziten Regeln übersetzen:

```
$ make main
```

Hinweis: Make weiß, wie `main` aus `main.c` übersetzt werden kann.



1.3 Make

Make kann das Programm mit impliziten Regeln übersetzen:

```
$ make main
```

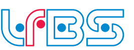
Hinweis: Make weiß, wie `main` aus `main.c` übersetzt werden kann.

einfachstes Makefile:

```
main: main.c
```

damit (erstes Target ist Standard):

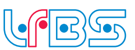
```
$ make
```



1.4 Linken

Objektdateien:

```
$ gcc -c main.c
```



1.4 Linken

Objektdateien:

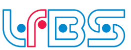
```
$ gcc -c main.c
```

Linken mit GCC:

```
$ gcc -o main main.o
```

GCC ruft den Linker `ld` mit den nötigen Parametern auf.
Standardname für ausführbare Datei: `a.out`

Übung 2:
Makefiles



Werkzeuge – 2: Make

Demo-Programm im Verzeichnis `src/demo`

- **main.c** enthält `main()`
- **output.c** Funktionen zur Datenausgabe
- **list.c** Verwaltung von linearen Listen

Kompilieren und Linken in einem Schritt:

```
$ gcc -o demo main.c output.c list.c
```

Kompilieren und Linken in mehreren Schritten:

```
$ gcc -c main.c
$ gcc -c output.c
$ gcc -c list.c
$ gcc -o demo main.o output.o list.o
```

2.1 Grundlagen

```
demo : main.o output.o list.o
    gcc -o demo main.o output.o list.o

clean :
    rm -rf demo *.o
```

- erste Regel ist Standard-Target
- nutze implizite Regel für `*.c` → `*.o`
- clean-Regel löscht erzeugte Dateien

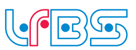
2.2 Phony-Targets

```
$ touch clean
$ make clean
```

- wenn eine Datei `clean` existiert:
  kein Grund, Befehl auszuführen, da Target aktuell

```
.PHONY : clean
```

diese Targets sind nie *aktuell*, werden also immer ausgeführt.

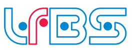


2.3 Variablen

```
CC=gcc
CFLAGS=-O0 -g
LDFLAGS=

demo : main.o output.o list.o
     $(CC) -o demo $(LDFLAGS) main.o output.o lis
```

betrachte Ausgabe: auch implizite Regeln benutzen diese Standardvariablen

CC	Compiler
CFLAGS	Flags für Compiler
LDFLAGS	Flags für Linker



2.4 Muster-Regeln

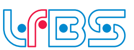
```
demo : main.o output.o list.o
    $(CC) -o $@ $(LDFLAGS) $^

%.o : %.c
    $(CC) -c $(CFLAGS) -o $@ $<
```

- `$@` Platzhalter für Target
- `$<` Platzhalter für erste Abhängigkeit
- `$^` Platzhalter für alle Abhängigkeiten
- `%` Platzhalter für Dateiname in Musterregel

2.5 Abhängigkeiten

```
demo : main.o output.o list.o
    $(CC) -o $@ $(LDFLAGS) $^
main.o : main.c defs.h output.h
output.o : output.c output.h
list.o : list.c list.h
```

Ein Ziel darf mehrmals auftauchen; Make kombiniert alle Angaben.



2.5 Abhängigkeiten

```
demo : main.o output.o list.o
	$(CC) -o $@ $(LDFLAGS) $^
main.o : main.c defs.h output.h
output.o : output.c output.h
list.o : list.c list.h
```

Ein Ziel darf mehrmals auftauchen; Make kombiniert alle Angaben.
GCC kann die Abhängigkeiten ausgeben:

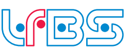
```
$ gcc -MM *.c
```

z. B. in Datei schreiben und diese in Makefile einbinden.

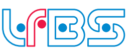
2.6 Dateien automatisch erkennen

```
CFILES=$(shell ls *.c)
OFILES=$(CFILES:.c=.o)

debug :
	@echo CFILES='$(CFILES)'
	@echo OFILES='$(OFILES)'
```

$(shell ...)	führt Shellbefehl aus und speichert stdout
$(VAR:a=b)	ersetzt a durch b in VAR
@...	vor Befehl: unterdrückt Ausgabe des Befehls
-...	vor Befehl: bricht nicht ab bei Fehler



Übung 3:
weitere Binutils

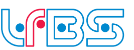


3.1 Disassembler

Mit objdump kann man Objektdateien disassemblieren:

```
$ objdump -d main.o
```

- Objektdateien: Funktionsaufrufe nicht gelinkt
- ausführbare Dateien
- ohne Optimierung besser verständlich
- -S : C-Code anzeigen (etwas unübersichtlich)



3.2 strings

Gibt alle Zeichenketten eines Programms aus:

```
$ strings hello
```

- zusammenhängende Folgen von druckbaren Zeichen
- was tut ein Programm? Welche Ausgaben sind möglich?

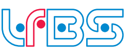
3.3 Symbole

nm zeigt Symbole aus Objektdateien an.

```
$ nm main.o
```

- t lokales Symbol in Text-Segment (Code)
- T exportiertes (globales) Symbol in Text
- d/D initialisierte Daten (Data)
- b/B uninitialisierte Daten (BSS)
- U undefiniertes Symbol

siehe `man nm` (und GI4, Kapitel *Binder und Lader*)



3.4 Strip

`strip` entfernt Symbole aus ausführbarer Datei:

```
$ strip hello
$ nm hello
```

- macht Datei etwas kleiner
- kein Debugging mehr möglich

3.5 Bibliotheken

`ldd` zeigt benötigte Bibliotheken an

```
$ ldd hello
```

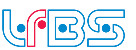
- Pfad und Version benötigter Bibliotheken

3.6 strace

Systemcalls können mit `strace` beobachtet werden:

```
$ strace ./hello
```

- zeigt jeden Syscall mit Parametern und Antwort



3.7 hexdump

Mit hexdump können binäre Daten ausgegeben werden:

```
$ hexdump -C hello
```

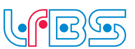
- Parameter -C : Offset, Hex und Ascii-Ausgabe

3.8 Autotools

Die Autotools erzeugen Makefiles:

```
$ ./configure
$ make
$ sudo make install
```

- `configure` erlaubt Einstellungen:
  - Installationspfad: `--prefix=/usr/local/packages/hello-2.1`
  - (De-)Aktivieren von Funktionalitäten
  - Auswahl von nötigen Bibliotheken
  - siehe: `./configure --help`
- prüft, ob alle nötigen Abhängigkeiten erfüllt sind
- `make` übersetzt
- `make install` installiert in `/usr/local`
  (i. d. R. `root`-Rechte nötig: `sudo make install`)



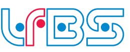
3.9 ctags und cscope

```
$ ctags -R
```

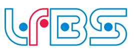
- in VIM: Strg + [(folge Tag) Strg + T (zurück)

```
$ cscope
```

- startet Text-Interface
- erlaubt umfangreiche Suchen (auch Aufrufe einer Funktion)
- öffnet Quellcode in Editor (Standard: VIM)



Übung 4:
Fehlersuche



4.1 Warnungen

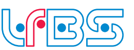
Man sollte die Warnungen immer einschalten und berücksichtigen

```
$ gcc -Wall -Wextra -o hello main.c
```

`-Wall` (fast) alle Warnungen aktivieren

`-Wextra` weitere Warnungen aktivieren

- ungenutzte Variablen und Funktionen sind während der Entwicklung ok
- in Ausnahmefällen können einzelne Warnungen per Pragma deaktivert werden



4.2 clang static analyser

Der C-Compiler `clang` von LLVM hat noch umfangreichere Warnungen.
Weiterhin bietet `clang` eine statitsche Analyse:

```
$ scan-build make
```

- benötigt funktionierendes Makefile
- dauert länger als normales Übersetzen
- erzeugt HTML-Report

4.3 weitere Werkzeuge

gprof Profiler erzeugt Call-Graph von ausgeführten Programmen

```
$ gcc -o hello main.c -pg
$ ./hello
$ gprof hello
```

valgrind debugging und Profiling zur Laufzeit

oprofile siehe [2]

systemtap Kernel profilen

Linux Trace Toolkit Kernel Tracer

Übung 5:
Subversion

5.1 Auschecken

Benötigt Projekt-URL (Repository)

```
$ svn checkout http://url dir
```

Erzeugt Ordner mit Namen `dir`.

Konventionen:

trunk	Entwicklungszweig
tags	gesicherte Versionen (z. B. veröffentlichte Version 1.0)
branches	Zweige für isolierte Entwicklung
subscribers	Datei mit Mailadressen zur Benachrichtigung

Repository Information anzeigen:

```
$ svn info
```

5.2 Update

Änderungen vom Server abholen:

```
$ svn update
```

Ausgabe der (letzten N) Änderungen:

```
$ svn log -l N [-v]
```

Ausgecheckte Version ändern (Repo sollte *clean* sein):

```
$ svn update -r123
```

-r123	Revision 123
-rHEAD	aktuelle Version
-rPREV	vorige Version

5.3 Änderungen kontrollieren

Änderungen anzeigen:

```
$ svn status
```

Neue Dateien hinzufügen:

```
$ svn add <datei>
```

Dateiänderungen anzeigen:

```
$ svn diff <datei>
```

welche Zeilen stammen von wem?

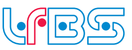
```
$ svn blame <datei>
```

5.4 Änderungen speichern

Hinweis: vorher Update durchführen (wenn fremde Änderungen zu erwarten):

```
$ svn update
$ svn commit -m "Änderungsnachricht"
```

- Beim Update kann es zu Konflikten kommen
  → Hinweise lesen!
- Die Änderungsnachricht sollte sinnvoll sein!
- ohne Parameter `-m`: Editor wird geöffnet.



5.5 Konflikte

Meist werden Konflikte automatisch *gemerged*.
Wenn Änderungen an der gleichen Zeile: manuelle Korrektur nötig.

```
<<<< .r123
printf("Hello,␣world!");
=====
printf("Hello,␣you!");
>>>> .mine
```

- Suche so markierte Stellen.
- Übertrage Änderungen in eine Version
- Entferne andere Version und Marker-Zeilen
- Teile SVN mit, dass Konflikt aufgelöst

5.6 Properties

```
$ svn proplist <datei>
$ svn propget svn:executable <datei>
$ svn propset svn:executable '*' <datei>
$ svn propedit svn:ignore <verzeichnis>
```

svn:ignore	Muster von Dateien, die nicht von `svn st` gemeldet werden sollen
svn:executable	Ausführbare Dateien (Wert: '*')
svn:eol-style	'native' – Beim Auschecken unter Windows Zeilenenden korrigieren
svn:external	Externe SVN-Repositories einbinden

Übung 6:
GDB

RWTH AACHEN UNIVERSITY

Werkzeuge – 6: GDB

1. C-Programm auspacken, kompilieren, Segfault.
2. Core-Dump? → Aktivieren
3. mit GDB öffnen (keine Debug-Infos)
4. mit Debug-Infos neu übersetzen, keine Optimierung!
5. Ausführen, Core-Dump in GDB öffnen
6. im GDB ausführen, Breakpoint setzen
7. Schritt-für-Schritt
8. Variablen untersuchen

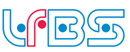
Literatur

LLVM Clang Project
Clang Static Analyzer.
http://clang-analyzer.llvm.org/index.html

IBM Developerworks
Smashing performance with OProfile.
http://www.ibm.com/developerworks/linux/library/l-oprof/index.html

Zusammenfassung

- Übersicht
- Arbeiten auf der Kommandozeile
- Shell-Skripte
- Werkzeuge
- Zusammenfassung



Zusammenfassung

jeder Programmierer muss sein Werkzeug beherrschen!

- lerne mit zehn Fingern zu schreiben
- entscheide Dich für einen Editor und meistere ihn
- automatisiere wiederkehrende Aufgaben (Skripte, Makefiles)

Anleitungen dazu gibt es genug im Netz!