Programmiermethoden und -werkzeuge 1

Woche 4 - Bash

Jochen Hosenfeld

jochen.hosenfeld@informatik.hs-fulda.de

Fachbereich Angewandte Informatik

November 14, 2025

Bash in WSL

Bash in WSL

```
johos@AI-HOSENFELD-NB:~$
johos@AI-HOSENFELD-NB:~$ cd ..
johos@AI-HOSENFELD-NB:/home$ ls
johos
johos@AI-HOSENFELD-NB:/home$
```

Bash in WSL

```
johos@AI-HOSENFELD-NB:~$
johos@AI-HOSENFELD-NB:~$ cd ..
johos@AI-HOSENFELD-NB:/home$ ls
johos
johos@AI-HOSENFELD-NB:/home$ cd ..
johos@AI-HOSENFELD-NB:/$ ls
Docker  boot  etc   init  lib32  libx32      media  opt   root  sbin  srv  tmp  var         wslfhnlfe  wsljf
bin     dev   home  lib   lib64  lost+found  mnt    proc  run   snap  sys  usr  wslNiPghe  wslhhBbfe  wsllE
johos@AI-HOSENFELD-NB:/$
```

Bash in WSL

```
johos@AI-HOSENFELD-NB:~$
johos@AI-HOSENFELD-NB:~$ cd ..
johos@AI-HOSENFELD-NB:/home$ ls
johos
johos@AI-HOSENFELD-NB:/home$ cd ..
johos@AI-HOSENFELD-NB:/$ ls
Docker  boot  etc   init  lib32  libx32      media  opt   root  sbin  srv  tmp  var        wslfhnlfe  wsljf
bin     dev   home  lib   lib64  lost+found  mnt    proc  run   snap  sys  usr  wslNiPghe  wslhhBbfe  wsllE
johos@AI-HOSENFELD-NB:/$ cd mnt
johos@AI-HOSENFELD-NB:/mnt$ ls
c  wsl  wslg
johos@AI-HOSENFELD-NB:/mnt$ cd c
johos@AI-HOSENFELD-NB:/mnt/c$
```

Bash in traditionellen Unix-Systemen

Layer	Beschreibung
Bash	Kommandozeilen-Schnittstelle
Unix-Kernel	Direkt auf physischer Hardware laufender Kernel
Hardware	Physische Maschine mit CPU, Speicher, Festplatte, Netzwerk usw.

Bash in Linux

Layer	Beschreibung
Bash	Kommandozeilen-Schnittstelle
Linux Distribution	Sammlung von Software und Tools, die als Benutzerumgebung dienen (z. B. Ubuntu)
Linux-Kernel	Kontrolle über Hardware, Prozesse, Speicher und Treiber
Hardware-Abstraktionsschicht (HAL)	Treiber, die die Hardware ansprechen und verwalten
Hardware	Physische Maschine mit CPU, Speicher, Festplatte, Netzwerk usw.

Bash in WSL

Layer	Beschreibung
Bash	Kommandozeilen-Schnittstelle
Linux Distribution	Benutzerumgebung und Tools, die in WSL laufen (z. B. Ubuntu)
Linux-Kernel (VM)	Echter Linux-Kernel, der in einer virtuellen Maschine (VM) läuft
Virtualisierung (Hyper-V)	Virtualisierungstechnologie von Windows; verwaltet Linux-VM
Windows	Betriebssystem; kontrolliert Hardware, HAL und Virtualisierung
Hardware-Abstraktionsschicht (HAL)	Treiber, die die Hardware ansprechen und verwalten
Hardware	Physische Maschine mit CPU, Speicher, Festplatte, Netzwerk usw.

Virtuelle Maschine

Partneraufgabe: *Was versteht man unter einer virtuellen Maschine?*

Virtuelle Maschine

- Softwarebasierter, isolierter Computer auf Host-System
- Simuliert CPU, RAM, Speicher & Netzwerk
- Verwaltet durch Hypervisor
- Ermöglicht parallele Betriebssysteme auf einem Rechner

Commands

Aufbau von Commands

Arguments

- CommandName

  → ls

- CommandName Argument

  → mkdir folder1

- CommandName Argument Argument

  → cp folder1 folder2

=> CommandName [Argument]…

Aufbau von Commands

Options

- CommandName

  → ls

- CommandName Option

  → ls -a

- CommandName Option Option

  → ls -a -l

  → ls -al

=> CommandName [Option]…

Beispiel: Make Directory

```
johos@AI-HOSENFELD-NB:~/Studium$ mkdir Erstes Semester
```

Beispiel: Make Directory

```
johos@AI-HOSENFELD-NB:~/Studium$ mkdir Erstes Semester
johos@AI-HOSENFELD-NB:~/Studium$ mkdir Zweites Semester
mkdir: cannot create directory ‘Semester’: File exists
```

Beispiel: Make Directory

```
johos@AI-HOSENFELD-NB:~/Studium$ mkdir Erstes Semester
johos@AI-HOSENFELD-NB:~/Studium$ mkdir Zweites Semester
mkdir: cannot create directory ‘Semester’: File exists
johos@AI-HOSENFELD-NB:~/Studium$ ls
Erstes  Semester  Zweites
```

Quoting und *Escaping*

- Quoting
  - → Single Quotes: ’
  - → Double Quotes: ”
- Escaping
  - → Backslash: \

Beispiel: Make Directory

```
johos@AI-HOSENFELD-NB:~/Studium$ mkdir "Erstes Semester"
johos@AI-HOSENFELD-NB:~/Studium$ mkdir 'Zweites Semester'
johos@AI-HOSENFELD-NB:~/Studium$ mkdir Drittes\ Semester
johos@AI-HOSENFELD-NB:~/Studium$ ls
'Drittes Semester'  'Erstes Semester'  'Zweites Semester'
```

Tipp

- Quoting statt Escaping
- Code ist besser lesbar
- Erleichtert spätere Bearbeitung
- bessere Unterscheidung zwischen wörtlichem und nicht-wörtlichem Code

Ablauf eines Commands

- Bash bekommt einen *Input*
- *Input* wird gelesen und aufgeteilt in *Words*

  → **“word splitting”**

- Geteilt wird nach einem *Whitespace* (*Spaces* oder *Tabs*)

Words und Strings

- In Bash ist ein *Word* mehr als ein Wort
- *String*: Reihe von Buchstaben, die eine Einheit bilden
  - → Ein Wort
  - → Mehrere Wörter
- In Bash ist fast alles ein String:
  - → Namen von Commands
  - → Argumente
  - → Variablen
  - → Namen von Files
  - → …

Strings

```
1  Einkaufsliste
2  -------------
3  Mehl
4  Zucker
5  Milch (3,5%)
6  10 Eier
7  Kakao, am besten Fair Trade
8  Schokostreusel oder -flocken
```

Beispiel: Euro und Dollar

```
johos@AI-HOSENFELD-NB:~$ echo How much is $10 in €?
```

Beispiel: Euro und Dollar

```
johos@AI-HOSENFELD-NB:~$ echo How much is $10 in €?
How much is 0 in €?
```

Beispiel: Euro und Dollar

```
1  johos@AI-HOSENFELD-NB:~$ echo How much is $10 in €?
2  How much is 0 in €?
3  johos@AI-HOSENFELD-NB:~$ echo "How much is $10 in €?"
4  How much is 0 in €?
```

Beispiel: Euro und Dollar

```
1 johos@AI-HOSENFELD-NB:~$ echo How much is $10 in €?
2 How much is 0 in €?
3 johos@AI-HOSENFELD-NB:~$ echo "How much is $10 in €?"
4 How much is 0 in €?
5 johos@AI-HOSENFELD-NB:~$ echo 'How much is $10 in €?'
6 How much is $10 in €?
```

Beispiel: Peter Parker

```
johos@AI-HOSENFELD-NB:~/Variables$ ls
Peter  'Peter Griffin'  'Peter Jackson'  'Peter Parker'
johos@AI-HOSENFELD-NB:~/Variables$ name='Peter Parker'
johos@AI-HOSENFELD-NB:~/Variables$ echo $name
Peter Parker
johos@AI-HOSENFELD-NB:~/Variables$ rmdir $name
rmdir: failed to remove 'Parker': No such file or directory
johos@AI-HOSENFELD-NB:~/Variables$ ls
'Peter Griffin'  'Peter Jackson'  'Peter Parker'
johos@AI-HOSENFELD-NB:~/Variables$ rmdir "$name"
johos@AI-HOSENFELD-NB:~/Variables$ ls
'Peter Griffin'  'Peter Jackson'
```

Redirections

```
1 echo "hello" > hello.txt
2 cat < hello.txt > hello_world.txt
3 echo "World" >> hello_world.txt
```

Zeichen	Name	Beschreibung
<	Input Redirection	Leitet die Standardeingabe eines Befehls um.
>	Output Redirection	Leitet die Standardausgabe in eine Datei und überschreibt sie dabei.
>>	Appending Output Redirection	Leitet die Standardausgabe in eine Datei und hängt sie an.

Streams

- Fluss von Daten, der entweder innerhalb eines Prozesses oder zwischen Prozessen stattfindet.

- Streams ermöglichen es Bash-Skripten und -Befehlen, Daten sequenziell zu verarbeiten, ohne die Gesamtstruktur oder -größe der Daten im Voraus kennen zu müssen.

File Descriptors

Referenzen für Streams

- **FD 0** - Standard Input (`stdin`): Ermöglicht es Befehlen in Bash, Eingaben zu empfangen, meist von der Tastatur oder durch Daten, die von anderen Befehlen über Pipelines weitergeleitet werden.

- **FD 1** - Standard Output (`stdout`): Beinhaltet die regulären Ausgaben von Bash-Befehlen, die in der Regel im Terminal angezeigt oder zu Dateien bzw. anderen Befehlen umgeleitet werden können.

- **FD 2** - Standard Error (`stderr`): Speziell für Fehlermeldungen genutzt, ermöglicht eine getrennte Behandlung und Umleitung von Fehlerausgaben, um diese leicht von regulären Ausgaben zu unterscheiden.

Arten von Commands

Partneraufgabe: *Woher kennt Bash Befehle wie echo, cd oder mkdir?*

Arten von Commands

- Alias
- Function
- Builtin
- Ausführbares Programm

Alias

- Kurzname oder Ersatztext für einen anderen Befehl oder eine Befehlssequenz
- Definiert mit `alias name='Befehl'`
- Nur in der Shell-Umgebung gültig, keine echte Datei
- Sichtbar mit dem Befehl `alias`
- Vereinfacht häufig genutzte oder lange Befehle

Function

- Benutzerdefinierte Funktionen in der Shell
- Besteht aus einer Folge von Shell-Befehlen, zusammengefasst unter einem Namen
- Definiert z. B. mit `function name { command; }` oder `name() { command; }`
- Erweitert die Shell um eigene Befehle oder Logik
- Sichtbar mit `declare -f` oder `type name`

Function

```
mcd () {
    mkdir -p "$1"
    cd "$1"
}
```

```
sum () {
    echo "$1 + $2 = $(($1 + $2))"
}
```

Builtin

- Interne Shell-Befehle, die direkt von Bash implementiert und ausgeführt werden
- Schneller als externe Programme, da keine Prozess-Erzeugung notwendig
- Beispiele: `cd, echo, history, pwd`
- Sichtbar mit `type` name oder `help`
- Können speziellen Zugriff auf die Shell selbst bieten (z. B. Zugriff auf Umgebungsvariablen)

Externes Programm

- Programme als ausführbare Dateien im Dateisystem (meist in Verzeichnissen wie `/bin, /usr/bin`)
- Werden von Bash durch Suchen in `$PATH` gefunden und als eigener Prozess gestartet
- Beispiele: `mkdir, ls`
- Sichtbar mit Befehlen wie `which, type` oder `command -v`
- Können beliebige Programme oder Skripte sein, nicht nur Shell-Befehle

Glob Patterns

Glob Patterns

- **Glob**: Allgemeiner Name für Bash-Funktionen, die spezifische Muster erkennen oder erweitern.
- “global pattern” -> Globbing
- Synonym: “Pattern Matching”, ~~“Regular Expressions”~~
- Syntax:
  - → * passt auf 0 oder mehr Zeichen.
  - → ? passt genau auf ein Zeichen.
  - → […] passt auf jedes einzelne Zeichen in einer angegebenen Menge (siehe Bereiche).
- Verankerung: Alle Globs sind automatisch am Anfang und am Ende verankert.

Beispiel Globs

Muster	Beschreibung
`*`	Passt auf jede Zeichenkette beliebiger Länge.
`Kapitel*`	Passt auf jede Zeichenkette, die mit **Kapitel** beginnt.
`*a*`	Passt auf jede Zeichenkette, die ein **a** enthält (am Anfang, in der Mitte oder am Ende).
`*.jpg`	Passt auf jede Zeichenkette, die mit **.jpg** endet.
`*[sn]`	Passt auf jede Zeichenkette, die mit **s** oder **n** endet.
`Sei?`	Passt auf **Sein**, **Seid** oder **Seit**, aber nicht auf **Seife**.

Special Characters 1

Zeichen	Name	Beschreibung
	Whitespace	Bestimmen, wo *words* beginnen und enden.
`$`	Expansion	Leitet eine Variable oder einen Parameterersatz ein.
`'`	Single Quotes	Text bleibt wortwörtlich, Sonderzeichen werden ignoriert.
`"`	Double Quotes	Schützt vor *word splitting*, erlaubt aber *Expansions*. Die meisten Sonderzeichen werden ignoriert.
`\`	Escape	Charakter dahinter wird *escaped*, d. h. sie verlieren ihre spezielle Bedeutung.
`#`	Comment	Wird für Kommentare verwendet. Danach wird alles bis zum Zeilenende ignoriert.
`=`	Assignment	Weist einer Variablen einen Wert zu. Whitespace dazwischen sind nicht erlaubt.
`|`	Pipe	Leitet die Ausgabe eines Befehls als Eingabe an einen anderen weiter.

Special Characters 2

Zeichen	Name	Beschreibung
`*`	Wildcard	Steht für null oder mehr Zeichen in Dateinamen.
`?`	Wildcard	Steht für genau ein beliebiges Zeichen in Dateinamen.
`<`	Input Redirection	Leitet die Standardeingabe eines Befehls um.
`>`	Output Redirection	Leitet die Standardausgabe in eine Datei und überschreibt sie dabei.
`>>`	Appending Output Redirection	Leitet die Standardausgabe in eine Datei und hängt sie an.
`2>`	Error Redirection	Leitet die Standardfehlerausgabe in eine Datei.
`2>&1`	Combine Standard and Error Output Redirection	Leitet die Fehlerausgabe zur Standardausgabe, oft verwendet, um alle Ausgaben in eine Datei umzuleiten.
`~`	Home Directory	Verweist auf das Heimverzeichnis des aktuellen Benutzers.

Zusammenfassung

- Bash in WSL
- Commands
  - → Quoting und Escaping
  - → Word Splitting
  - → Redirections
  - → File Descriptors
  - → Arten von Commands
- Glob Patterns