Voraussetzungen

Ziele

Inhalt

- Korn-Shell, Kommandoausführung
- Standarddateien
- Umlenken
- Shellvariable
- Maskierung
- Wildcards
- Aliasing
- Redo
- Voreinstellungen
- Shell-Skripts

Definition Shell

- Eine Shell ist ein **textueller Kommandointerpreter**.
- Standard: Bourne Shell
- Erweiterung: C Shell (UC Berkeley)
- zu SVR4 (System V Release 4) neu: **Korn Shell**
  - enthält die Erweiterungen der C Shell, ist aber Bourne-kompatibel
  - erlaubt ein Editieren von Kommandozeilen mit vi oder emacs
  - zusätzliche Erweiterungen

M3 Shells (2)

- hier: Korn Shell
- Aufruf

  `$ sh` (Bourne)

  `$ ksh` (Korn)

  `$ exit` (Ende)

M3 Kommandoausführung (1)

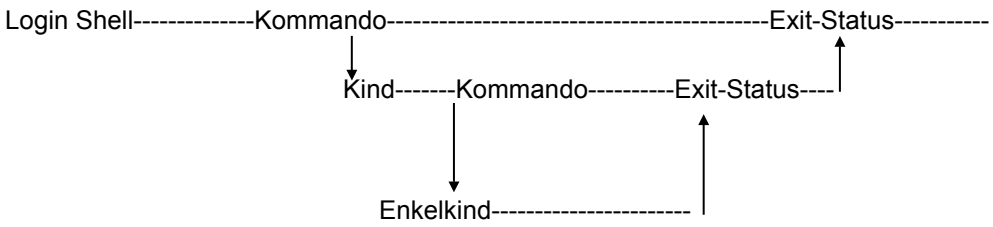
- Syntax: `$ kommando [argumente] <CR>`
- Das Abschicken eines Shell-Kommandos bewirkt die Erzeugung eines Kind-Prozesses (mittels **`fork`**).
- Prozesse können sich in unterschiedlichen **Zuständen** befinden.

  Die wichtigsten sind:

  - laufend (running)
  - bereit (ready)
  - blockiert (blocked)

M3 Kommandoausführung (2)

- Ein über ein Kommando erzeugter Kindprozess kann weitere Kindprozesse erzeugen.
- Der Vater kann parallel zum Kindprozess weiterarbeiten oder auf das Ende des Kindprozesses warten.
- Der Kindprozess meldet dem Vater einen Exit-Status = 0 im Normalfall, oder ≠ 0 im Fehlerfall.
- Ein Kindprozess, dessen Vater nicht mehr existiert, ist im Zombie-Zustand (Z).



M3 Auswertungsmechanismus

Reihenfolge:

Kommandosubstitution	‘ls -l‘
Variablensubstitution	$name
Trennzeichen-Interpretation	Leerstellen, Tabulator und Neue Zeile
Umleitungen	>, >>, <, |
Dateinamengenerierung	?, *, [ ]
Kommandoausführung	

M3 Kommandotypen

- Es gibt 3 Kommandotypen:
  - **eingebaute Kommandos** (Shell-intern, kein Kindprozess, ca. 30 wichtige Kommandos)
  - **externe Kommandos** (Programme in /usr/bin)
  - **benutzererstellte** Prozeduren und Programme
- Dokumentation zu den eingebauten Kommandos:

```
$ man ksh
```

- Dokumentation zu allen Kommandos:

```
$ man kommandoname
```

- Übersicht über externe Kommandos:

```
$ ls  /usr/bin
```


M3 Standarddateien

- Jeder UNIX-Prozeß besitzt eine **Standard-Umgebung**, bestehend aus 3 Dateien („Datei“ im erweiterten Sinn!); ihnen entsprechen die Filedeskriptoren 0, 1, 2.
- Diese 3 Kanäle sind standardmäßig der Tastatur und dem Bildschirm zugeordnet:

0	**Standard Input**	(Tastatur)
1	**Standard Output**	(Bildschirm)
2	**Standard Error**	(Bildschirm)

- Über eine Umdefinition der Filedeskriptoren können diese Kanäle **umgelenkt** werden.

M3 Umlenken mittels Shell-Kommando (1)

- Mittels <, > und 2> wird auf beliebige Dateien umgelenkt:

  $ kommando [*argumente*] <*infile* >*outfile* 2>*errorfile*

- **> - Zeichen**

  $ cat *datei1* (Ausgabe → std out, also Bildschirm)

  $ cat *datei1* >*datei_xyz* (Ausgabe → ***datei_xyz***. Wenn ***datei_xyz*** existiert, wird sie ersetzt.)

- **>>-Zeichen**: Standard-Ausgabe wird an die existierende ***datei_xyz*** angehängt.

  $ cat *datei1* >>*datei_xyz*

M3 Umlenken mittels Shell-Kommando (2)

- **< -Zeichen**

  $ cat >*datei2* <*datei1* identisch mit

  $ cat datei1 >datei2

  $ cat >datei2 (Input von Tastatur, Ausgabe in ***datei2*** bis EoF-Zeichen CTRL D)

M3 Umlenken mittels Shell-Kommando (3)

- **2> -Zeichen**

  lenkt Standard Error um, z.B.

  ```
  $ cat <datei1 >/dev/null 2>datei3
  ```

  (falls datei1 nicht existiert, Fehlermeldung → ***datei3***)

- nützlich für Hintergrundprozesse:

  std out und std error werden umgelenkt in Dateien statt auf Bildschirm.

M3 Shellvariable (1)

- Definition von Variablen mittels Zuweisung. Variablen sind vom Typ String!

  ```
  $ e=2.718282          ohne Blank vor und nach „=„

  $ p=/usr/mike/ftn/prog1
  ```

- Ansprechen mit vorangestelltem **$**

- Interpretation:

  Shell ersetzt **$variable** durch den Wert von **variable** und führt dann das Kommando aus.

M3 Shellvariable (2)

- Verkettung durch Nacheinanderstellen
- Beispiele:
  e=2.718282
  p=/usr/mike/ftn/prog1

Kommando	Interpretation	
`e`	`e`	
`$e`	`2.718282`	
`$e5`	`(leer)`	(da `e5` nicht definiert ist)
`5$e`	`52.718282`	
`$p$e`	`/usr/mike/ftn/prog12.718282`	
`$p/$e`	`/usr/mike/ftn/prog1/2.718282`	
`$p5$e`	`2.718282`	(da `p5` nicht definiert ist)
`${p}5$e`	`/usr/mike/ftn/prog152.718282`	

M3 Shellvariable (3)

- Löschen

```
$ unset variable
```

- Wert ausgeben

```
(1)$ $e
   2.718282: not found
(2)$ echo $e
   2.718282
```

- `$ echo $!`

```
12905
```

(In der Variablen „!“ wird vom System die pid des zuletzt in den Hintergrund geschickten Prozesse gespeichert.)

M3 Sondervariable (1)

- Vom System sind einige Variable vordefiniert (Großbuchstaben!)

$HOME Das Verzeichnis, das mit dem Kommando **cd** (ohne Argument) eingestellt wird. Die Voreinstellung (gleichzeitig das anfängliche Working-Directory) wird im Benutzereintrag systemseitig festgelegt.

$PATH Die Kommandosuchfolge: Bei Eingabe eines Kommandos, dessen Name kein / enthält, wird immer zuerst geprüft, ob es ein eingebautes (Shell-internes) Kommando ist. Danach wird entsprechend der in **$PATH** vorgegebenen Suchfolge gesucht. Voreingestellt ist z.B.:

PATH = /bin:/usr/bin:.

wonach zuerst **/bin**, dann **/usr/bin** und zuletzt das aktuelle Verzeichnis (.) durchsucht wird. Verzeichnisse sind mit : zu trennen. Das zuerst gefundene, ausführbare Kommando wird ausgeführt. Diese Voreinstellung kann je nach System unterschiedlich sein.

M3 Sondervariable (2)

$TERM Die Terminaldefinition (z.B.: **vt100** oder **ansi).**

$PS1 Der Login-Prompt. Voreinstellung i.a.: **$.**

$PS2 Der Fortsetzungsprompt. Voreinstellung**: >.**

$IFS der Satz von Trennungszeichen. Voreingestellt sind das Leerzeichen, das Tabulatorzeichen und **CR.**

M3 Sondervariable (3)

- Beispiel

```
$ echo $HOME
/usr/mike
$ HOME=$HOME/prog
$ echo $HOME
/usr/mike/prog
```

- Anzeige aller Umgebungsvariablen (Benutzer- und Sonder-Variable) mit
  `$ set`
- Anzeige der Sondervariablen mit
  `$ env`

M3 Maskierung (Quoting) (1)

- Zeichen mit Sonderbedeutung für die Shell:

  `> < ; & ( ) | \ <CR> $`

  müssen maskiert werden, wenn sie in einem String vorkommen. Ihre Sonderbedeutung muß also unterbunden werden.

- 3 Methoden

  `\` maskiert das folgende Einzelzeichen

  `´.....´` maskiert alle eingeschlossenen Zeichen (außer ´)

  `"....."` maskiert alle eingeschlossenen Zeichen (außer `$ \ ´`)

- Beispiel 1

```
$ a=cd\ /usr/mike/ftn
$ b=´cd /usr/mike/ftn´
$ c="cd /usr/mike/ftn"
```

M3 Maskierung (Quoting) (2)

- Beispiel 2

```
$ d=$HOME e=\$HOME f=´$HOME´ g="$HOME"
$ echo $d $e $f $g
/usr/mike $HOME $HOME /usr/mike
```


M3 Kommandoersetzung (1)

- Wird ein Kommandoname in **umgekehrte Hochkommas** (`` `.....` ``) eingeschlossen, dann wird die Standardausgabe des Kommandos der Variablen zugewiesen (nicht zu verwechseln mit ´... ´ !)
-
```
$ k=`pwd`
$ echo $k
/usr/mike/ftn
```
- Umgekehrte Hochkommas quoten nicht.
- Umgekehrte Hochkommas werden durch \ und zwischen ´..... ´ gequotet, nicht aber zwischen "`....`"

- Beispiel 1

```
$ d=/usr/mike/ftn
$ m=`ls $d | wc -w`
$ echo $m
3
```

- Beispiel 2

```
$ n="Das aktuelle Verzeichnis ist `pwd`."
$ echo $n
Das aktuelle Verzeichnis ist /usr/mike/ftn/prog.
```


- Ein Wort, das *, ? oder [.....] enthält, ist ein **Namensmuster**. Es wird **vor** der Kommandoausführung in eine (alphabetisch sortierte) Liste **expandiert**.
- Bedeutung
  - * ein beliebiger String, auch der Nullstring,
  - ? ein einzelnes Zeichen,
  - [...] ein Satz einzelner Zeichen (ohne Trennzeichen!),
  - - (zwischen zwei Zeichen innerhalb [....] definiert einen Bereich,
  - ! (als erstes Zeichen nach []): alle Zeichen außer den gelisteten.

M3 Namensexpansion mit Wildcards (2)

- Beispiel 1

  aktuelles Verzeichnis enthält

  prog1, p1.1, p1.2, p1.3, p1.4, p1.5, p1.6. Übergeordnetes Verzeichnis enthält

  .profile, prog.a, prog.b

Muster	expandierte Dateien
prog*	prog1
p???	p1.1, p1.2, p1.3, p1.4, p1.5, p1.6
2	p1.2
p1.[136]	p1.1, p1.3, p1.6

M3 Namensexpansion mit Wildcards (3)

- Beispiel 1 Fortsetzung

```
p1.[1-36]          p1.1, P1.2, p1.3, p1.6

p1.[!1-36]         p1.4, p1.5

../*               ../prog.a, ../prog.b

../.*              ../.profile
```

M3 Wildcards

- Beispiel 2

`rm p1.*`	löscht `p1.1, p1.2,... p1.6`. Aber: **Vorsicht bei der Anwendung der Metazeichen in Löschkommandos!** Metazeichen in Löschkommandos sollten nur mit der `-i` (interaktiv)-Option verwendet werden; also besser:
`rm -i p1.*`	
`ls -l ?????`	Findet alle Dateien und Directories, deren Namen aus fünf Zeichen bestehen.
`cat p1.* >progs`	verkettet `p1.1, p1.2, ..., p1.6` in eine Sammeldatei `progs`.

M3 Aliasing

- Mit dieser Methode können Kommandostrings ersetzt werden, z. B. zur Abkürzung.
- Beispiel

```
$ alias h=history
$ alias l='ls -al | more'
$ alias rm='rm -i'
```

- `$ alias` ohne Argumente liefert die aktuellen Aliases.

M3 Redo (1)

- **Wiederholung** von Kommandos ohne neues Eintippen, ohne Modifikation

  `$ r` repeat (wiederholt das zuletzt eingegebene Kommando)

  `$ r -2` (wiederholt das vorletzte Kommando)

  `$ r x` (wiederholt das letzte Kommando, das mit x anfängt.)

- Wiederholung mit Modifikation

  a) mit vi

  b) mit emacs (s. u.)

M3 Redo mit emacs

- Navigieren in der Kommandohistorie

  CTRL p **p**revious: hole vorheriges Kommando

  (oder ↑)

  CTRL n **n**ext: hole nächstes Kommando

  (oder ↓)

- Cursor-Bewegung

  CTRL b **b**ack: Cursor nach links

  (oder ←)

  CTRL f **f**orward: Cursor nach rechts

  (oder →)

- Liste der vorherigen Befehle

  `$ history`

M3 Shell Voreinstellungen

- Set

  Beim Aufruf einer Shell können **Optionen** angegeben werden, z.B.

  ```
  $ ksh -o emacs
  ```

  Diese Option kann auch nachträglich mit `set` eingestellt werden:

  ```
  $ set -o emacs     oder
  $ set -o vi
  ```

- **Ausgabe** aller Shelloptionen mit

  ```
  $ set -o [Wert]
  ```

- Rücksetzen auf Default-Wert

  ```
  $ set +o Wert
  ```


M3 Start-up

- Beim Sessionstart (login) werden folgende Dateien ausgeführt:

  /home/<usr>/.login

  /etc/profile

  /home/<usr>/.profile

Wenn die Variable ENV auf .kshrc gesetzt ist, wird auch ~/.kshrc ausgeführt.

M3 Shellscripts

- Definition Shellscript:

  Ein Shellscript ist eine in eine Datei geschriebene, zusammenhängende Kommandofolge.

- Informationen:
  RRZN-Handbuch, Seiten 113 -129
  man ksh

- Sonderzeichen im Shellscript:
  # am Anfang der Zeile bedeutet Kommentar.
  \ als letztes Zeichen einer Zeile bedeutet, dass die Zeile fortgesetzt wird.
  ; Trennzeichen, wenn mehr als ein Kommando in einer Zeile steht.

- Ein Shellscript **`script`** kann ausgeführt werden, indem man die Datei **`script`**, die es enthält als ausführbar deklariert (x-Bit), oder indem man es mit dem Aufruf **`ksh script`** startet. Natürlich muss der Ausführende Leseberechtigung für die Datei **`script`** besitzen.

M3 Shellscripts: Parameter

- Beispiel skript02:

```
#!/bin/ksh
echo Hier ist die Korn-Shell,
echo in diesem Verzeichnis befinden sich folgende Einträge:
ls
```

- Parameter
  Nach dem Dateinamen können Parameter stehen. Sie werden durch Blanks getrennt und als Strings interpretiert. Im Shellscript werden sie automatisch bei 1 beginnend durchnummeriert und können als $i (mit i= 1, 2, ..., Anzahl der Parameter) angesprochen werden.

- Beispiel skript02a:

```
#! /bin/ksh
      echo Hier ist die Korn-Shell
      echo Das ist der erste Parameter: $1 ...
      echo und das ist der zweite Parameter: $2
```

- Es werden nur die ersten 9 Parameter über $1 .. $9 erreicht.
- Falls > 9 Parameter:
  `shift n` löscht die ersten n Argumente
- Beispiel

```
shift 8
echo $1 $3
```

Gibt den Wert des ursprünglich 9. und 11. Arguments zurück.

- Bedingte Ersetzung von Parametern
  $ aufzufassen als Ersetzungsoperator

Ausdruck	Ersetzung
`$parameter`	Wert, falls **`parameter`** definiert, sonst Leerstring
`${parameter-wort}`	Wert, falls **`parameter`** definiert, sonst **`wort`**
`${parameter?}`	Abbruch, falls **`parameter`** nicht definiert
`${parameter?meldung}`	Abbruch mit **`meldung`**, falls **`parameter`** nicht definiert

M3 Shellscripts: Leerkommando :

- Leerkommando :
  : bewirkt die Auswertung einer Folge von Argumenten.

- Beispiel 1:
```
: ${1?fehlt} ${2?fehlt}
```

- Beispiel 2:
```
echo Die Anzahl Dateien in ${1-$HOME} ist
ls ${1-$HOME}  |  wc -w
```

- Variablendefinition (Zuweisung), falls **variable** nicht definiert ist:
```
${variable=wort}
```


M3 Shellscripts: Shellvariable im Skript

- Verwendung von Shellvariablen im Skript wie von der Shelloberfläche

- Im Skript erzeugte oder umdefinierte Variable haben keinen Einfluss auf (gleichnamige) Variable außerhalb!

- Ausnahme: Mittels
```
$ export variablenliste
```
  definierte Variable werden aufgerufenen Prozeduren (und Kindprozessen) übergeben.

M3 Shellscripts: Punktbefehl .

- Wie kann man Prozeduren (= Shellskripte) benutzen, um die Variablen der aktuellen Shell zu ändern?

  ➡ Punktbefehl .

  Führt eine Prozedur aus, welche auf die Variablen der übergeordneten Shell wirkt.

- Beispiel

```
$ e=5
$ . testpunkt
echo $e
 6
```

wobei testpunkt:

e = 6

M3 Shellscripts: Sonderparameter

- Sonderparameter:

`$0`	der Name der Prozedur
`$#`	die Anzahl angegebener Argumente
`$*`	alle Argumente als eine Zeichenkette
`$$`	die PID-Nummer des aktuellen Prozesses

skript03

```
#! /bin/ksh
echo Hier ist die Korn-Shell
echo Name der Prozedur: $0
echo Anzahl Argumente: $#
echo alle Argumente als Zeichenkette: $*
echo meine pid: $$
```

- Beispiel: Ausgabe mehrerer Dateien am Bildschirm

skript03a

```
 #! /bin/ksh
echo Hier ist die Korn-Shell
cat $*
echo $# Dateien wurden ausgegeben.
```

M3 Shellscripts: read

- Daten einlesen:
  Die Standardeingabe eines Shellscripts ist per Voreinstellung **`/dev/null`**. Das Kommando **`read`** stellt dies um.

skript04
```
#! /bin/ksh
echo Hier ist die Korn-Shell
echo 'Eingabe von 3 Werten in 1 Zeile:'
read a b c
echo Parameter 1: $a
echo Parameter 2: $b
echo Parameter 3: $c
```

- Beispiel:
  `read a b c`
  erwartet die Eingabe einer Zeile von der Tastatur. Das erste Wort wird der Variable **`a`**, das zweite Wort der Variable **`b`** und der Rest der Zeile der Variable **`c`** zugewiesen. Analog bedeutet

  `read a b c <input`

  dass die erste Zeile der Datei **`input`** gelesen wird.

M3 Shellscripts: for

- Ablaufsteuerung: Die for Anweisung
  `for variable` **`//variable`** wird durch alle Argumente ersetzt
  `do`
  `  kommandos`
  `done`

  oder:
  `for variable; do kommandos; done`

skript05
```
#! /bin/ksh
echo Hier ist die Korn-Shell
for variable
do
        echo $variable
done
```

- Beispiel: Berechnen der Längen von mehreren Dateien

skript05a
```
#! /bin/ksh
echo Hier ist die Korn-Shell
for file
do
        Anzahl=`cat $file | wc -w`
        echo Die Anzahl der Wörter in $file ist $Anzahl
done
```

- Ablaufsteuerung: Die for Anweisung (2)

```
for variable in wörterliste          // variable wird durch alle
do                                   // Worte in wörterliste
  kommandos                          // ersetzt
done
```

skript06

```
#! /bin/ksh
echo Hier ist die Korn-Shell
for file in skript02 skript02a
do
        Anzahl=`cat $file | wc -w`
        echo Die Anzahl der Wörter in $file ist $Anzahl
done
```


- Bedingungen:
  Zum Testen einer Bedingung dient das Kommando [ ] mit Leerstellen nach [und vor]. Das Kommando [] hat zwar keine Ausgabe, jedoch einen Exit-Status (0 für wahr, ansonsten falsch).

Kommando	ist wahr falls:
`[ -d name ]`	Verzeichnis **name** existiert
`[ -f name ]`	Datei **name** existiert
`[ -s name ]`	Datei **name** existiert und ist nicht leer
`[ -r name ]`	Datei **name** existiert und ist lesbar

M3 Shellscripts: Bedingungen

- Bedingungen (Fortsetzung):

Kommando	ist wahr falls:
`[ -w name ]`	Datei **name** existiert und ist beschreibbar
`[ -x name ]`	Datei **name** existiert und ist ausführbar
`[ string ]`	**string** nicht der Nullstring ist
`[ name1 = name2 ]`	**name1** und **name2** identisch sind
`[ zahl1 -eq zahl2]`	**zahl1** und **zahl2** gleich sind
	(analog mit **-ne, -gt, -ge, -lt, -le**)

Verknüpfen von Bedingungen:

`-a` AND, `-o` OR, ! NOT

Kommando	ist wahr falls:
`[ -d name1 -a -f name2 ]`	Verzeichnis **name1** und Datei **name2** existieren
`[ ! -r name ]`	Datei **name** nicht existiert oder nicht lesbar ist

M3 Shellscripts: if

- Ablaufsteuerung: Die if Anweisung

```
if bedingung
then
  kommandos
fi
```

oder:

```
if bedingung
then
  kommandos_1
else
  kommandos_2
fi
```

- Die else if Anweisung

```
if bedingung
then
  kommandos_1
elif
  kommandos_2
else
  kommandos_3
fi
```

M3 Shellscripts: Beispiel

- Beispiel: Verfeinerung der Berechnung der Längen von mehreren Dateien

skript09

```
#! /bin/ksh
echo Hier ist die Korn-Shell
for file
do
        if [ -d $file ]
        then
                echo $file ist ein Verzeichnis
        elif [ ! -s $file ]
        then
                echo $file ist leer
        else
                Anzahl=`cat $file | wc -w`
                echo Die Anzahl der Wörter in $file ist $Anzahl
        fi
done
```


M3 Shellscripts: while und until

- Ablaufsteuerung: Die **`while`** Anweisung

```
while bedingung
do
  kommandos
done
```

skript10

```
#! /bin/ksh
echo Hier ist die Korn-Shell
while [ ! -f $1 ]
do
        echo Wo ist die Datei $1 ????
        sleep 3
done
echo Endlich ist die Datei $1 da!!!
```

Die **`kommandos`** werden wiederholt ausgeführt, solange **`bedingung`** wahr ist

- Ablaufsteuerung: Die **`until`** Anweisung

```
until bedingung
do
  kommandos
done
```

Die **`kommandos`** werden wiederholt ausgeführt, solange **`bedingung`** falsch ist

M3 Shellscripts: Arithmetik

- Arithmetik im Shellskript ist umständlich!

```
e=1
e=$(($e + 3))
echo $e
   4
```

- Alternative

```
typeset -i e
e=1
e=e+3
echo $e
   4
```


M3 Shellscripts: Steuerkommandos

- Einige Steuerkommandos:

`exit`	beendet die aktuelle Shell mit dem Exit-Status des zuletzt ausgeführten Kommandos
`break`	beendet die aktuelle Schleife und setzt nach dem nächsten **`done`** fort
`continue`	bricht den aktuellen Iterationsschritt ab und startet den nächsten Schleifendurchlauf
`true`	immer wahr (Exit-Status 0)
`false`	immer falsch (Exit-Status ungleich 0)