integrata cegos

Linux/UNIX Shellprogrammierung und Tools

Shellskripte mit bash und ksh verstehen, erstellen, erweitern

1

WICHTIGE UNIX-KOMMANDOS

Wichtige UNIX-Kommandos

integrata cegos

Kommando	Funktion
grep, egrep, fgrep	Muster/String-Suche in Dateien
sort	Dateien Zeilen- oder Spaltenweise sortieren Standard-Trennzeichen: Tabulator, Leerzeichen
head	Ausgabe der ersten n Zeilen aus einer Datei Standard: 10
tail	Ausgabe der letzten n Zeilen aus einer Datei Standard: 10
cut	Text Spalten- bzw. Zeichenweise aus einer Datei heraus Schneiden; Standard-Trennzeichen: Tabulator
tr	Konvertieren, Komprimieren oder Löschen von Zeichen/Bytes aus der Standardeingabe
find	rekursives Durchsuchen von Directorybäumen nach Einträgen, die auf entsprechend spezifizierte Auswahlbedingungen passen

Mustersuche in Dateien – grep

- Die 3 Kommandos für die Suche in Textdateien:

fgrep	Einfache Suchtextbeschreibung in Form von String-Konstanten (***fast grep***), am schnellsten
grep	Unterstützt Sonderzeichen für die Beschreibung von regulären Ausdrücken (BRE's) an der Stelle des Suchtextes
egrep	Unterstützt Sonderzeichen für die Beschreibung von regulären Ausdrücken (ERE's) an der Stelle des Suchtextes, mehrere Musterbeschreibungen können mit einem logischen ODER verkettet werden (***extended grep***)

Mustersuche in Dateien – grep

- Aufruf-Syntax: grep [*optionen*] *muster* [*datei ....*]

Option	Bedeutung
-c	nur die Anzahl der gefundenen Zeilen ausgeben
-i	Klein- und Großschreibung ignorieren
-l	nur die Namen der Datei, in denen der Text mindestens einmal gefundenen wurde, ausgeben
-n	Ausgabe der gefundenen Zeilen, mit vorangestellter Zeilennummer
-v	Alle Zeilen ausgeben, die das Suchmuster nicht enthalten
-E	Unterstützung erweiterter regulärer Ausdrücke (ERE's - ersetzt egrep)
-F	Suche mit String-Konstanten (ersetzt fgrep)
-e *muster*	Die Option kennzeichnet das nachfolgende Argument als Musterbeschreibung (kann auch mit einem Minuszeichen beginnen), zur Mehrfachnennung von Suchmustern
-f *mdatei*	Suchmuster werden aus der Datei *mdatei* ausgelesen

Mustersuche in Dateien – grep

- Die regulären Ausdrücke von grep:

grep	grep -E	Bedeutung
^	^	Zeilenanfang
$	$	Zeilenende
.	.	ein beliebiges Zeichen
[...]	[...]	eines der Zeichen aus der Liste oder aus dem Zeichenbereich (-)
[^...]	[^...]	ein Zeichen, das nicht in der Liste oder dem Zeichenbereich (-) steht
\z	\z	maskiert Metazeichen *z*
	(...)	gruppiert mehrere Zeichen
	... | ...	ODER-Verknüpfung

Mustersuche in Dateien – grep

- Die regulären Ausdrücke von grep (cont.):

grep	grep -E	Bedeutung
*z**	*z**	0 bis n-malige Wiederholung von *z*
.*	.*	eine beliebige Zeichenfolge
	z+	1 bis n-malige Wiederholung von *z*
	z?	0 oder 1-malige Wiederholung von *z*
z\{*n*,*m*\} *z*\{*n*,\} *z*\{*n*\}	*z*{*n*,*m*} *z*{*n*,} *z*{*n*}	*n* bis *m*-malige Wiederholung von *z* mindestens *n* Wiederholungen von *z* genau *n* Wiederholungen von *z*
\<		Wortanfang
\>		Wortende
\(… \)		Speicheranforderung
n		Speicher Nr. *n* auslesen (1 <= *n* <= 9)

Mustersuche in Dateien – grep

- Vorrangregeln für die Operatoren in BRE's:

Operator	Bedeutung
*****m*	Geschützte Metazeichen (z.B. \$)
[ ]	Klammerausdrücke
**\(\) ***n*	Speicheranforderung und Speicherbezüge
*** \{ \}**	Wiederholungen
kein Symbol	Verkettung
^ $	Anker (Zeilenanfang, Zeilenende)

Mustersuche in Dateien – grep

- Vorrangregeln für die Operatoren in ERE's:

Operator	Bedeutung
*****m*	Geschützte Metazeichen (z.B. \$)
[ ]	Klammerausdrücke
( )	Gruppierung
*** + ? { }**	Wiederholungen
kein Symbol	Verkettung
^ $	Anker (Zeilenanfang, Zeilenende)
|	Logisches ODER

Sortieren von Textdateien – sort

- Aufruf-Syntax: sort [optionen] [-t x] [+pos [-pos]] ... [datei ...]

Option	Bedeutung
-d	Sortierung noch Wörterbuchordnung (nur Buchstaben, Ziffern, Leer- und Tabulator-Zeichen sind signifikant)
-f	Groß- und Kleinschreibung ignorieren
-r	umgekehrte Sortierreihenfolge (absteigend)
-u	identische Zeilen nur einmal in die Ausgabe stellen
-o *file*	Ausgabe in Datei *file* statt auf Standard-Ausgabe
-t *x*	Trennzeichen für Spalten ist *x*
-n	numerische Sortierung
-b	führende Leerzeichen/Tabulatoren ignorieren
-M	Sortierung nach Monatskürzel „JAN“ < „FEB“ < ... < „DEC“
+*f1***[.***c***]**	Beginn der Sortierung hinter Feld *f* und Zeichen *c* Zeichenposition optional, Standard: Spaltenanfang
-*f2***[.***c***]**	Ende der Sortierung hinter Feld *f* und Zeichen *c* Zeichenposition optional, Standard: Spaltenanfang Endfeld optional, Standard: letztes Feld
-k *f1***[.***c***][,***f2***[.***c***]]**	Neuere Unix-/Linux-Varianten kennen diese Option, mit der die Sortierspalten exakt angegeben werden können, zum Beispiel –k 4,7 (von Spalte 4 bis 7). Zeichenposition optional, Standard: Spaltenanfang Endfeld optional, Standard: letztes Feld

Anfang/Ende einer Datei anzeigen – head/tail

- Aufruf-Syntax: head [optionen] [datei]

Option	Bedeutung
-n	Ausgabe der ersten *n* Zeilen

- Aufruf-Syntax: tail [optionen] [datei]

Option	Bedeutung
-n	Ausgabe der letzten *n* Zeilen
+n	Ausgabe der Datei ab Zeile *n* bis zum Ende
-f	Ständige Ausgabe von neu hinzugekommene Zeilen am Dateiende, bis diese mit Strg+C abgebrochen wird

Text aus einer Datei ausschneiden – cut

- Aufruf-Syntax: cut -c liste [datei ....]
  cut [-d x] -f liste [-s] [datei ....]

Option/ Argument	Bedeutung
-c	Zeichenweise ausschneiden
-f	Feld/Spaltenweise ausschneiden
-d *x*	Feldtrennzeichen ist *x* (Standard: genau 1 Tabulator)
liste	*n*,*m* Position *n* und *m* *n*-*m* Position *n* bis *m* *n*- Position *n* bis zum Ende -*m* Anfang bis Position *m*
-s	Zeilen, die das Trennzeichen nicht enthalten, werden in der Ausgabe unterdrückt

Zeichenersetzung – tr (translate)

- Aufruf-Syntax: tr [optionen] ‘Zeichenfolge1‘ [‘Zeichenfolge2‘]

Option	Bedeutung
-s	(squeeze) In der ersten Folge mehrfach hintereinander auftretende Zeichen werden nur einmal abgebildet
-d	(delete) Die angegebenen Zeichen aus *zeichenfolge1* werden gelöscht
-c	(complement) alle Zeichen außer denen in der ersten Folge werden behandelt

Zeichenersetzung – tr (translate)

- POSIX-Zeichenklassen:

Klasse	Bedeutung	Klasse	Bedeutung
[:alpha:]	Buchstaben	**[:lower:]**	Kleinbuchstaben
[:digit:]	Ziffern	**[:upper:]**	Großbuchstaben
[:alnum:]	Buchstaben und Ziffern	**[:punct:]**	Interpunktionszeichen
[:blank:]	Leerz. und TAB	**[:xdigit:]**	Hexadezimale Ziffern
[:graph:]	Buchstaben, Ziffern und Interpunktion	**[:print:]**	Druckbare Zeichen
[:space:]	Whitespace-Zeichen	**[:cntrl:]**	Steuerzeichen

Dateien suchen – find

- Aufruf-Syntax: find startdir ... [-kriterium [arg]] ... [-aktion] ...
- Auszug aus den Kriterien:

Kriterium	Bedeutung
-name *pattern*	Suche nach Namen, die mit dem angegebenen Muster übereinstimmen
-type *t*	Suche nach Einträgen, die dem angegebenen Dateityp entsprechen f reguläre Datei d Directory l symbolic Link b Block Device c Character Device p Named Pipe
-user *uname*	Suche nach Einträgen, die dem Nutzer <uname> gehören (uname kann UID oder Name sein)
-group *gname*	Suche nach Einträgen, die der Gruppe <gname> gehören (gname kann GID oder Name sein)

Dateien suchen – find

- Auszug aus den Kriterien (cont.):

Kriterium	Bedeutung
-mtime [+|-]*n* **-atime** [+|-]*n* **-ctime** [+|-]*n*	Suche nach Einträgen mit Modifikations-, Zugriffs- bzw. Statusänderungszeit vor <*n*> Tagen (+*n* – *n* oder mehr Tage , -*n* – bis zu *n* Tagen)
-newer *file*	Suche nach Einträgen, deren Modifikations- oder Statusänderungszeit neuer ist als die von <file>
-size [+|-]*n*[c|k]	Suche nach Einträgen, deren Dateigröße <*n*> Blöcke ist (c – Byte, k – kByte) (+*n* – *n* oder mehr, -*n* – bis zu *n*)
-inum *num*	Suche nach Einträgen, deren Inode-Nummer mit <num> übereinstimmt
-mount	durchsuche nur das Filesystem, in dem sich das <startdir> befindet
-perm [-]*onum*	Suche nach Einträgen, deren Zugriffsrechte genau mit <onum> übereinstimmen (-onum - es werden nur die angegebenen Rechte überprüft)

Dateien suchen – find

- Mehrere Kriterien können logisch miteinander Verknüpft werden:

Verknüpfung	Bedeutung
\(... \)	Gruppierung, zur Änderung des Vorrangs
!	Logische Negation
-a	Logisches UND (Default)
-o	Logisches ODER

Dateien suchen – find

- Mögliche Aktionen:

Aktion	Bedeutung
-print	Ausgabe der gefundenen Einträge auf stdout als Pfad; meistens die Standardaktion, wenn die Aktion in der Syntax fehlt (Ausgabeformat: startdir/eintrag)
-ls	Ausgabe der gefundenen Einträge auf stdout entsprechend dem Format des Kommandos ls –lisd
-exec cmd ... { } ... \;	Weiterverarbeitung der gefundenen Einträge mit dem <cmd> ({ } – Platzhalter für gefundenen Eintrag)
-ok cmd ... { } ... \;	Weiterverarbeitung der gefundenen Einträge mit dem <cmd> (mit interaktiver Rückfrage über stderr)

2

KORN-SHELL: GRUNDLAGEN

DIE WICHTIGSTEN SHELLS

- Bourne-Shell (sh)
- C-Shell (csh)
- Korn-Shell (ksh)
- Bourne-Again-Shell (bash)

SCHREIBEN VON SKRIPTEN

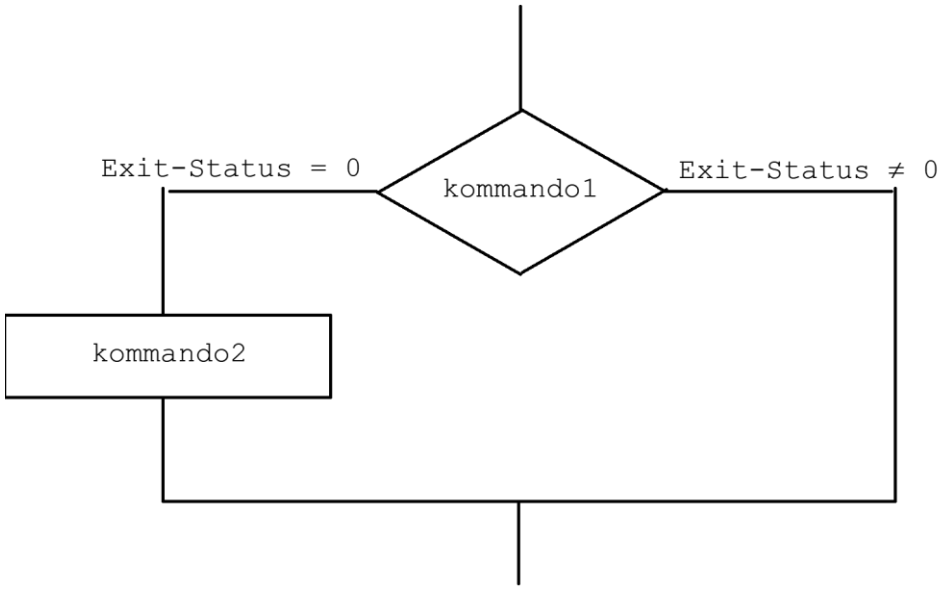
1. Benennen des Skriptes
   - type *skriptname*
   - whereis *skriptname*
2. Editieren des Skriptes
   - #!/bin/ksh – Shebang-Zeile
3. Testen des Skriptes
   - Debug-Optionen: -n , -u , -v , -x
4. Ausführen des Skriptes
   - x-Recht setzen
   - Start in Child-Prozess: *skriptname* [*argliste*]

DIE KOMMANDOZEILE

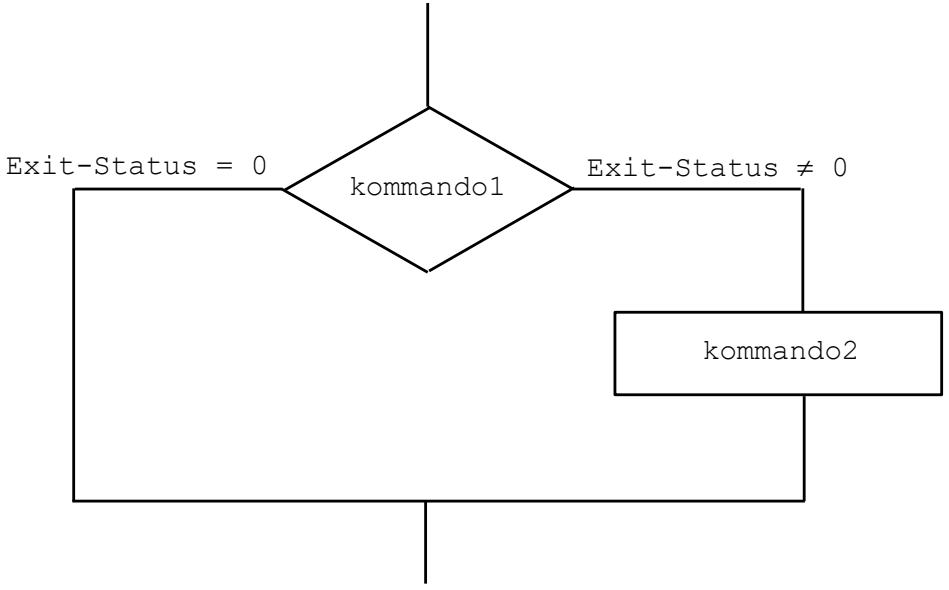
1. Analyse der Kommandozeile
2. Ersetzen von Sonderzeichen
   - Variablen-Substitution: $name
   - Dateinamen-Expansion: * , ? , [ ] , ...
   - Kommando-Substitution: `kmd ....` oder $(kmd ....)
   - Ein/Ausgabe-Umlenkung: < , > , >> , | , ...
3. Steuerung des Ablaufs
   - if, for, while, ...
4. Suchen nach dem angegeben Kommando via PATH
5. Prozess-Steuerung

BEDINGTE KOMMANDOAUSFÜHRUNG

- Syntax:
  kommando1 **&&** *kommando2*



- Syntax:
  kommando1 **||** *kommando2*



GRUPPIERUNGEN

- Aufruf in einer Subshell

Syntax:

(*cmd1*; *cmd2*; *…***)**

Einsatzmöglichkeiten:

(*kdo1*; *kdo2*) > datei
(*kdo1*; *kdo2*) | *kdo3*
(*kdo1*; *kdo2*) &

- Reine Gruppierung

Syntax:

{ *cmd1*; *cmd2*; *…* **; }**

DAS HERE-DOCUMENT

- Syntax:

```
kommando .....   <<[-] MARKE
.....
.....          Dynamische Textdaten durch:
.....          $varname
.....          $( kommando .... )  oder  ` kommando .... `

.....          Aufheben von $ bzw. ` durch \  möglich
.....
MARKE
```

SHELL-OPTIONEN – DER BEFEHL set -o

- Einstellungen der Shell lassen sich über Shell-Optionen bzw. Schalter-Variablen beeinflussen. Die Steuerung erfolgt über das Kommando set.

set -*option*	Option aktivieren
set +*option*	Option aktivieren
set -o *option*	Schalter aktivieren
set +o *option*	Schalter deaktivieren
set -o	Alle Schalter mit Zustand anzeigen

SHELL-OPTIONEN – DER BEFEHL set -o

- Die wichtigsten Schalter für die interaktive Arbeit:

Schalter (Default/Empfehlung)	Bedeutung
bgnice (on/on)	Hintergrund-Jobs mit schlechterer Priorität ausführen
vi (off)	Aktivieren des Built-in-Editors vi für die Kmd.-Zeilen-History
emacs (off/on)	Aktivieren des Built-in-Editors vi für die Kmd.-Zeilen-History
Ignoreeof (off/on)	Login-Shell kann nicht mit **^D** beendet werden Logout muss über **exit** erfolgen
Noclobber (off/on)	Existierende Datei kann nicht mit Ausgabe-Umlenkung überschrieben werden
markdirs (off)	Bei der Dateinamen-Expansion alle Directories mit einem / ergänzen

SHELL-OPTIONEN – DER BEFEHL set -o

- Die wichtigsten Schalter für die Shellprogrammierung:

Option	Schalter (Default)	Bedeutung
-n	**noexec (off)**	Kmd.'s werden nur gelesen und nicht ausgeführt; erster Syntaxcheck
-u	**nounset (off)**	Zugriffe auf nicht definierte Variablen werden als Fehler gewertet
-v	**verbose (off)**	Kmd.-Zeilen vor der Ausführung anzeigen
-x	**xtrace (off)**	Kmd.-Zeilen nach der Substitution/Expansion, vor der Ausführung anzeigen
-		verbose und xtrace abschalten
-e	**errexit (off)**	Die Shell wird bei Auftreten eines Fehlers sofort beendet.
-k	**keyword (off)**	Variablen-Definitionen werden am = erkannt und nicht an der Position Default: $ v1=w1 v2=w2 skriptname …. set –k : $ skriptname … v1=w1 v2=w2 …

3

KORN-SHELL: VARIABLE

Variable

- Bestandteil der aktuellen Prozessumgebung
- Erlöschen nach Prozessende (Beenden der Shell)
- Name einer Variable darf aus Buchstaben, Ziffern und dem Unterstrich “_“ bestehen
- Per Konvention werden Umgebungsvariable in Großbuchstaben geschrieben

- **Syntax**

```
var=wert                    (Definition mit Wert wert)
        var=                (Definition ohne Wert)
        var=$var2”Zusatz…”  (Wertzuweisung mit Konkatenation)
        var=$var2$var3
```

- **Zugriff**

Syntax

```
$var
${var}
```

- **Löschen**

Syntax

```
unset var
```

Environmentvariable

- Variablendefinition ist nur innerhalb des aktuellen Prozesses gültig.
- Soll ein Child-Prozess die Variablen vom Parent-Prozess erben, muss die Variable exportiert werden.

export *variable*	Variable ins Environment stellen
set	Anzeiger aller Variablen aus dem aktuellen Prozess
env	Anzeige der Environment-Variablen
export	Anzeige der Environment-Variablen

Vordefinierte Shellvariable

- **Suchpfade**

Variable	Bedeutung
PATH	Directories, in denen die Shell nach Kommandos sucht
FPATH	Directories, in denen die Shell nach Dateien mit Funktionsdefinitionen sucht

- **Terminal und Prompt**

Variable	Bedeutung
TERM	Terminaltyp
DISPLAY	Definition des Ausgabe-Displays für X-Clients (Default: :0.0)
PS1	Primäres Promptzeichen (Default: $)
PS2	Promptzeichen für Folgezeilen (Default: >)
PS3	Prompt beim *select*-Kommando (Default: #?)
PS4	Prompt bei der *xtrace*-Option (Default: +)

Vordefinierte Shellvariable

- Sonstige

Variable	Bedeutung
ENV	Datei, die beim Start eines Korn-Shellskriptes abgearbeitet wird (in *.profile* gesetzt), bevor die erste Skriptzeile ausgeführt wird. Zur Definition von Variablen und Funktionen, die nahezu in jedem Korn-Shellskript als Vorlauf gebraucht werden
HOME	Pfad zum eigenen Home-Directory
HOSTNAME	Rechnername
LOGNAME **USER**	Anmeldename (Login-Name); je nach Unix-/Linux-Derivat wird für den Benutzernamen eine dieser Variablen verwendet (USER: BSD-Unix-Derivate; LOGNAME: System-V-Unix-Derivate).
PWD	AktuellesDirectory
OLDPWD	Vorheriges Directory
IFS	Internal Field Separator: Trennzeichen zwischen Kommando, Optionen und Argumenten (Default: Blank, Tab, Newline)

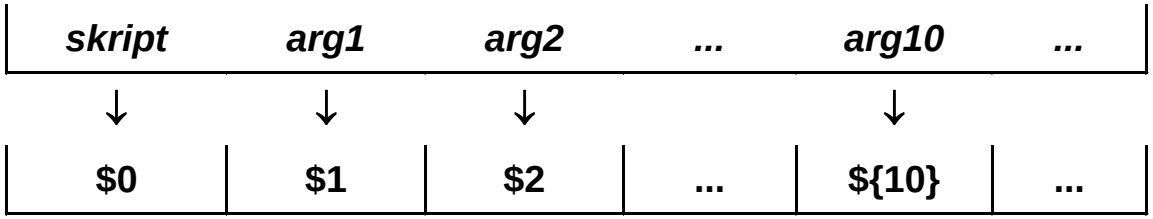
Vordefinierte Shellvariable

- Interne

Variable	Bedeutung
$?	Abfrage des Endestatus des letzten Befehls
$!	Prozess-ID des letzten Hintergrundprozesses
$$	Prozess-ID der aktuellen Shell
$-	Liefert String mit den gesetzte Shell-Optionen

Positionsparameter

- Ermöglichen Zugriff auf die Argumente eines Shell-Skriptes
- Werden beim Aufruf eines Skriptes automatisch belegt:



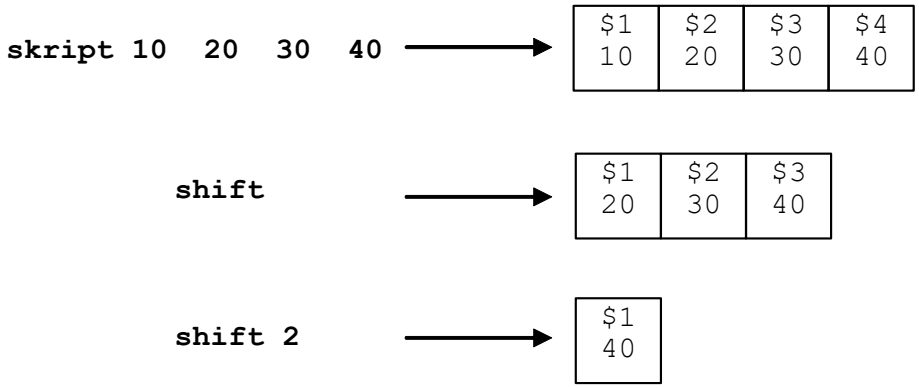
$0	Name des Shell-Skriptes
$n	1. bis 9. Positionsparameter (n ≤= 9)
${n}	Ab 10. Positionsparameter (n ≥= 10)
$*	Liste aller Positionsparameter ($1... $n) “$*“ ⇒ “$1 $2 … $n“
$@	Liste aller Positionsparameter ($1... $n) “$@“ ⇒ „$1“ „$2“ … „$n“
$#	Anzahl der übergebenen Parameter

Parameter bearbeiten

- Parameterliste verschieben – shift

Syntax:

shift [n]



- indirekte Wertzuweisung

Syntax:

set [-s] [- -] *arg1 arg2*

- Löschen

Syntax:

set –

shift $#

Erweiterte Variablen-Prüfung

- Verwendung von Defaultwerten
  - Wenn Variable *var* existiert und nicht leer ist, gebe ihren Wert zurück ansonsten den Default-Wert
  - Als Variable kann auch ein Positionsparameter eingesetzt werden

${*var*:-*default*}	(Test auf Existenz und nicht leeren Wert)
${*var*-*default*}	(Test nur auf Existenz)

- Zuweisung von Defaultwerten
  - Wenn Variable *var* existiert und nicht leer ist, gebe ihren Wert zurück ansonsten den Default-Wert und weise diesen zu
  - Ein Positionsparameter kann nicht eingesetzt werden!

${*var*:=*default*}	(Test auf Existenz und nicht leeren Wert)
${*var*=*default*}	(Test nur auf Existenz)

Erweiterte Variablen-Prüfung

- Einfache Fehlerbehandlung
  - Wenn Variable *var* existiert und nicht leer ist, gebe ihren Wert zurück ansonsten beende das Programm mit einer Fehlermeldung
  - Als Variable kann auch ein Positionsparameter eingesetzt werden

${*var*:?*error*}	(Test auf Existenz und nicht leeren Wert)
${*var*?*error*}	(Test nur auf Existenz)

- Test auf Existenz
  - Wenn Variable *var* existiert und nicht leer ist, gebe String zurück ansonsten ersetze durch einen Leerestring
  - Als Variable kann auch ein Positionsparameter eingesetzt werden

${*var*:+*string*}	(Test auf Existenz und nicht leeren Wert)
${*var*+*string*}	(Test nur auf Existenz)

4

KORN-SHELL: EIN- UND AUSGABEERWEITERUNGEN

Ein- und Ausgabe von Text

Escape-Sequenz	Bedeutung
\a	Systempiep (Alarm)
\b	Backspace (CTRL-H)
\c	Newline am Zeilenende unterdrücken
\f	Formfeed (Seitenvorschub)
\n	Newline (CTRL-J)
\r	Return (CTRL-M)
\t	Tabulator (CTRL-I)
\v	Vertikaler Tabulator (CTRL-K)
\0*n*	ASCII-Zeichen in oktalem Wert *n*
****	ein Backslash

Ausgabe mit echo oder print

Syntax:

echo [*text*]

Syntax:

print [*optionen*] [*text*]

Auszug aus den Optionen für print:

Option	Bedeutung
--	nächstes Argument ist keine Option
-n	kein Newline (identisch mit \c)
-u *n*	Ausgabe nach Kanal *n* (eventuell mit exec öffnen)

Formatierte Ausgabe - typeset

Syntax:

typeset [-*opt*] *var*[=*wert*]

Auszug aus den Optionen:

Option	Bedeutung
-L[*n*]	Linksbündig: Führende Leerzeichen werden entfernt. Ist *n* angegeben, wird rechts mit Leerzeichen aufgefüllt oder abgeschnitten(!).
-LZ[*n*]	Linksbündig: Führende Nullen werden entfernt. Ist *n* angegeben, wird rechts mit Leerzeichen aufgefüllt oder abgeschnitten(!).
-R[*n*]	Rechtsbündig: Nachstehende Leerzeichen werden entfernt. Ist *n* angegeben, wird links mit Leerzeichen aufgefüllt oder abgeschnitten.
-Z[*n*] **-RZ[*n*]**	Rechtsbündig wie R*n:* Anstelle von Leerzeichen werden führende Nullen eingefügt.
-l	Konvertieren in Kleinbuchstaben (lowercase)
-u	Konvertieren in Großbuchstaben (uppercase)

Eingabe mit read

Syntax ksh:

read [*optionen*] [*var1[**?“**promptstring“] var2 …*]

Syntax bash:

read [*optionen*] [*-p „promptstring“] var1 var2 …*]

Auszug aus den Optionen:

Option	Bedeutung
-u*n*	Lesen vom Kanal *n* (mit exec öffnen)
-r	raw mode. Zeilenende kann nicht mit \ maskiert werden

Erweitertes Kanalkonzept

- Standard Ein/Ausgabe-Umlenkungen:

Umlenkung	Bedeutung
kdo n> datei	*datei* zum Überschreiben öffnen und mit Kanal *n* verbinden
kdo n>> datei	*datei* zum Anhängen öffnen und mit Kanal *n* verbinden
kdo n< datei	*datei* zum Lesen öffnen und mit Kanal *n* verbinden
kdo < datei	*datei* zum Lesen öffnen und mit *stdin* verbinden
kdo1 | kdo2	Pipe: *stdout* von *kdo1* als *stdin* von *kdo2* verwenden
kdo >| datei	*stdout* auf *datei* zwingen, auch wenn die Option *noclobber* aktiv ist
kdo <> datei	*datei* für Ein- und Ausgabe verwenden
kdo << [-]marke **...** ***marke***	Here-document: *stdin* von *kdo* wird aus Folgezeilen genommen (bis *marke*)
kdo m>&n	Duplizieren der Schreib-Verbindung von Kanal *n* über Kanal *m*
kdo m<&n	Duplizieren der Lese-Verbindung von Kanal *n* über Kanal *m*

Erweitertes Kanalkonzept- exec

- Dauerhafte Kanalumlenkung – exec:

Eingabe-Umlenkung	Bedeutung
exec *n< datei*	*datei* mit Datei-Deskriptor *n* zum Lesen öffnen
exec *m<&n*	Datei-Deskriptor *m* als eine Kopie von Datei-Deskriptor *n* zum Lesen erzeugen
exec *n<&-*	schließen der zum Lesen geöffnete Datei, die mit Datei-Deskriptor *n* verknüpft ist
Ausgabe-Umlenkung	**Bedeutung**
exec *n>datei*	*datei* mit Datei-Deskriptor *n* zum Überschreiben öffnen
exec *n>>datei*	*datei* mit Datei-Deskriptor *n* zum anhängenden Schreiben öffnen
exec *m>&n*	Datei-Deskriptor *m* als eine Kopie von Datei-Deskriptor *n* zum Schreiben erzeugen
exec *n>&-*	schließen der zum Schreiben geöffnete Datei, die mit Datei-Deskriptor *n* verknüpft ist
E/A-Umlenkung	**Bedeutung**
exec *n<>datei*	*datei* mit Datei-Deskriptor *n* zum Lesen und Schreiben öffnen

5

KORN-SHELL: MUSTERERKENNUNG UND STRINGMANIPULATION

Erweiterter Mustervergleich

integrata cegos

Syntax:
operator(*muster***)**

Operator	Bedeutung
(muster***)**	0 bis *n* Mal
+(*muster***)**	1 bis *n* Mal
?(*muster***)**	0 oder 1 Mal
@**(***muster***)**	genau 1 Mal. Sinnvoll bei @(*muster1* | *muster2* |...)
!(*muster***)**	Negation: alles, was nicht durch *muster* beschrieben wird

Hinweis: Eine Oder-Verknüpfung zwischen zwei Mustern ist durch das Pipesymbol "|“ möglich (*muster1* | *muster2*).

Stringmanipulation

- Abschneiden vom Anfang

Syntax:

${*var#muster***}**	(kürzesten Teil entfernen)
${*var##muster***}**	(längsten Teil entfernen)

- Abschneiden vom Ende

Syntax:

${*var***%***muster***}**	(kürzesten Teil entfernen)
${*var***%%***muster***}**	(längsten Teil entfernen)

- String-Länge ermitteln

Syntax:

${*#var***}**

6

KORN-SHELL: ABLAUFSTEUERUNG 1 – VERZWEIGUNGEN

Bedingungen prüfen

test *bedingung* oder [*bedingung*]

Beide Testkommandos sind aus allen Shells heraus verfügbar.

[[*bedingung*]]

Zusätzliches Testkommando der Korn-Shell.

Alle 3 Konstrukte setzen den Exit-Status auf Erfolg (0) oder nicht Erfolg (!=0).

Folgende Operationen werden von allen 3 Testkommandos unterstützt:

- Dateitest-Operationen
- Integer-Vergleiche
- String-Vergleiche
- Logische Verknüpfung von Ausdrücken

Dateitest-Operationen

test, […], [[…]]	Dateiattribute
-a *dat* oder **-e** *dat*	existiert *dat* (beliebiger Typ)
-f *dat*	ist *dat* eine reguläre Datei (file)
-d *dat*	ist *dat* ein Directory
-L *dat*	ist *dat* ein symbolischer Link
-b *dat*	ist *dat* eine blockorientierte Gerätedatei
-c *dat*	ist *dat* eine character-orientierte Gerätedatei
-r *dat*	ist *dat* lesbar (read)
-w *dat*	ist *dat* schreibbar (write)
-x *dat*	ist *dat* ausführbar (execute)
-s *dat*	ist *dat* nicht leer (size)
-O *dat*	gleiche UID wie Eigentümer von *dat* (owner)
-G *dat*	gleiche GID wie die Gruppe von *dat* (group)
-u *dat*	ist für *dat* das SUID-Bit gesetzt
-g *dat*	ist für *dat* das SGID-Bit gesetzt
-k *dat*	ist für *dat* das Sticky-Bit gesetzt
dat1 **-nt** *dat2*	ist *dat1* neuer als *dat2* (newer than)
dat1 **-ot** *dat2*	ist *dat1* älter als *dat2* (older than)
dat1 **-ef** *dat2*	ist *dat1* nur ein anderer Name für *dat2* (hardlink)
-o option	ist *option* gesetzt (für Schalter: bspw. noclobber)

Integer-Vergleiche

test, [...], [[…]]	Integervergleiche
z1 **-eq** *z2*	gleich (equal)
z1 **-ne** *z2*	ungleich (not equal)
z1 **-lt** *z2*	kleiner (less than)
z1 **-le** *z2*	kleiner gleich (less equal)
z1 **-gt** *z2*	größer (greater than)
z1 **-ge** *z2*	größer gleich (greater equal)

String-Vergleiche

test bzw. [...]	String-Vergleiche	[[…]]
str **=** *string*	*str* wird durch *muster* beschrieben (Mustervergleich)	*str* **=** *muster*
str **!=** *string*	*str* wird nicht durch *muster* beschrieben	*str* **!=** *muster*
-n *str*	*str* ist nicht leer (non zero)	**-n** *str*
-z *str*	*str* ist leere Zeichenkette (zero)	**-z** *str*
	str1 ascii-mäßig kleiner als *str2*	*str1* **<** *str2*
	str1 ascii-mäßig größer als *str2*	*str1* **>** *str2*

Logische Verknüpfungen

test bzw. [...]	Verknüpfung	[[…]]
\(... \)	Gruppieren	**(** … **)**
! *bed*	Negation	**!** *bed*
bed1 **-a** *bed2*	logisches UND	*bed1* **&&** *bed2*
bed1 **-o** *bed2*	logisches ODER	*bed1* **||** *bed2*

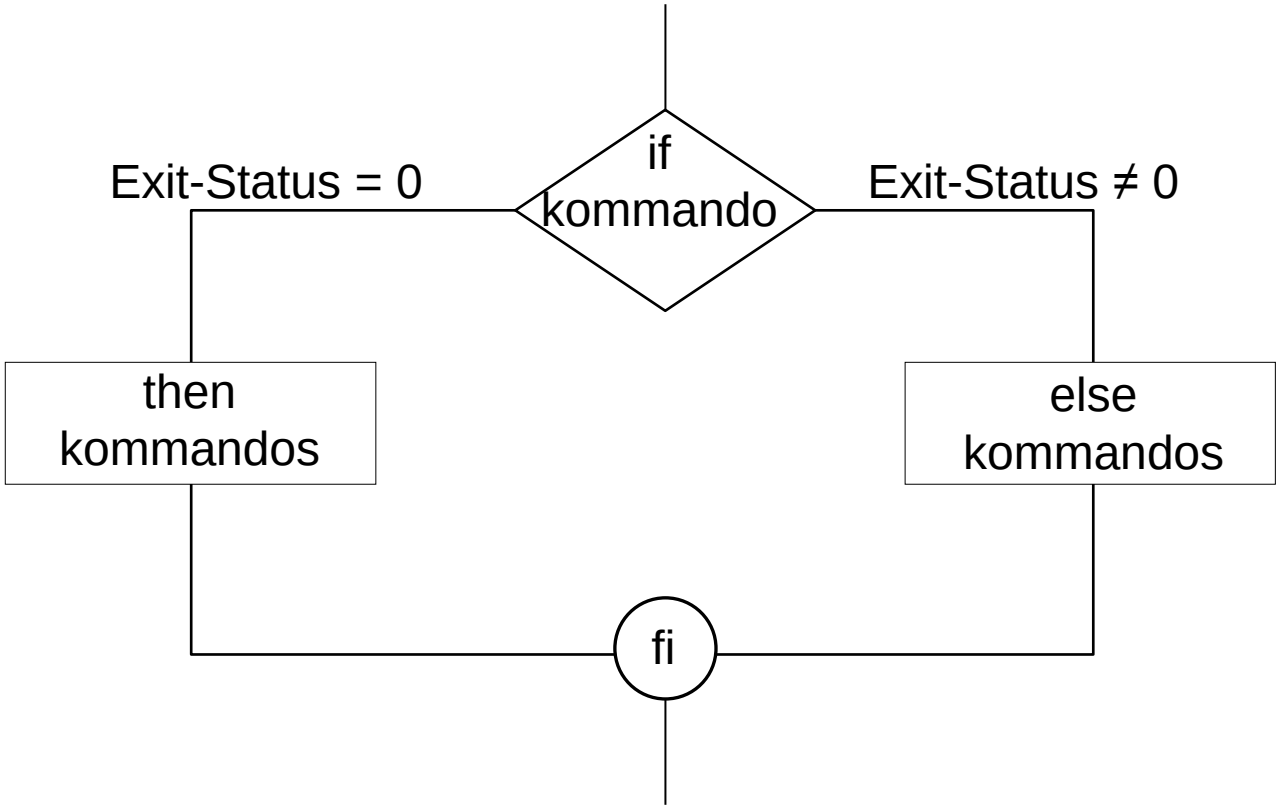
if – Anweisung (1)

Syntax:

```
if    kommando(s)
then
      kommando(s)
fi
```

```
if          kommando(s)
then
            kommando(s)
else
kommando(s)
fi
```

if - Anweisung (2)



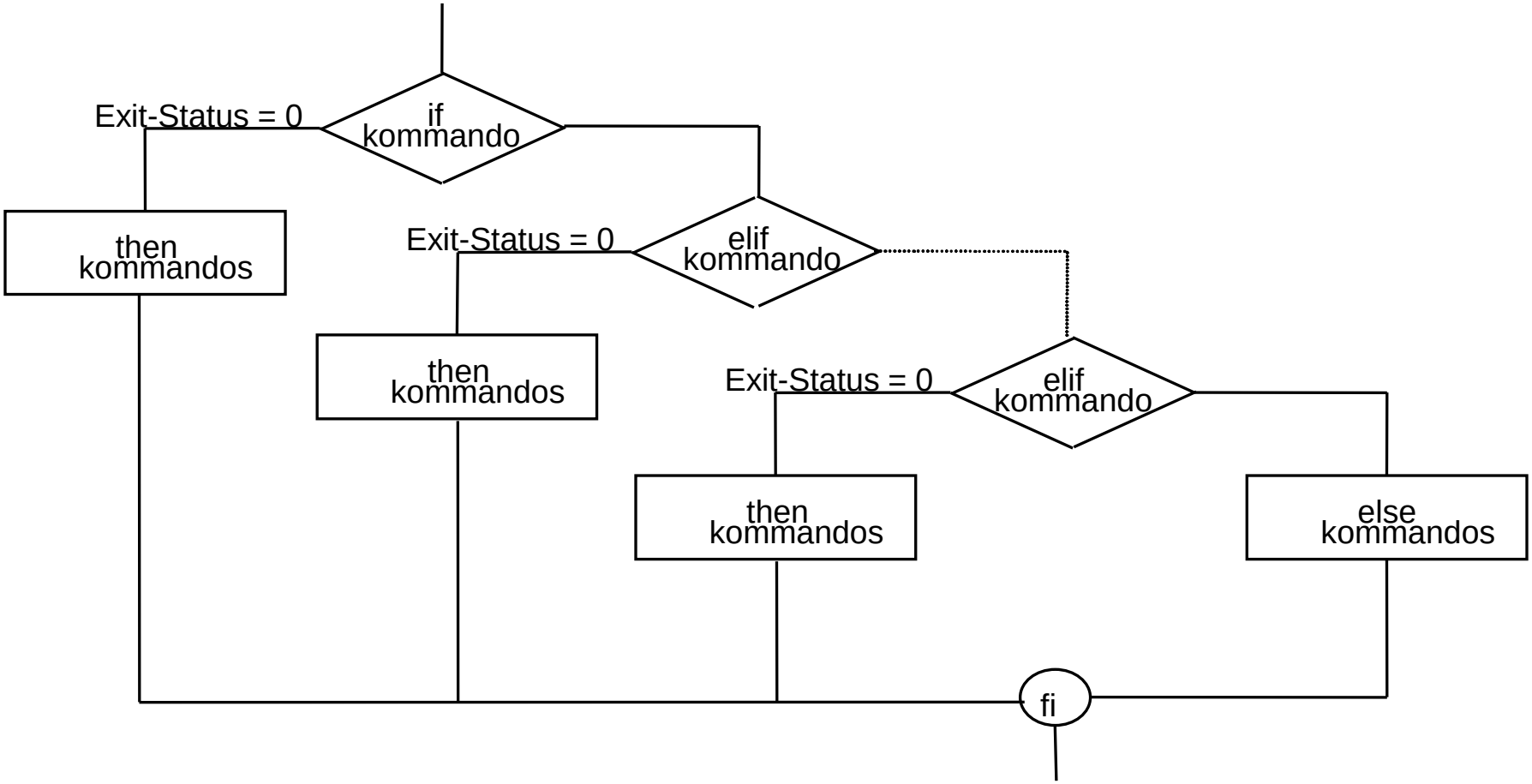
Verschachtelte if – Anweisungen (1)

Syntax:

```
if    kommando(s)
then
      kommando(s)
elif  kommando(s)
then
      kommando(s)
…
fi
```

```
if          kommando(s)
then
            kommando(s)
elif        kommando(s)
then
            kommando(s)
…
else
            kommando(s)
 fi
```

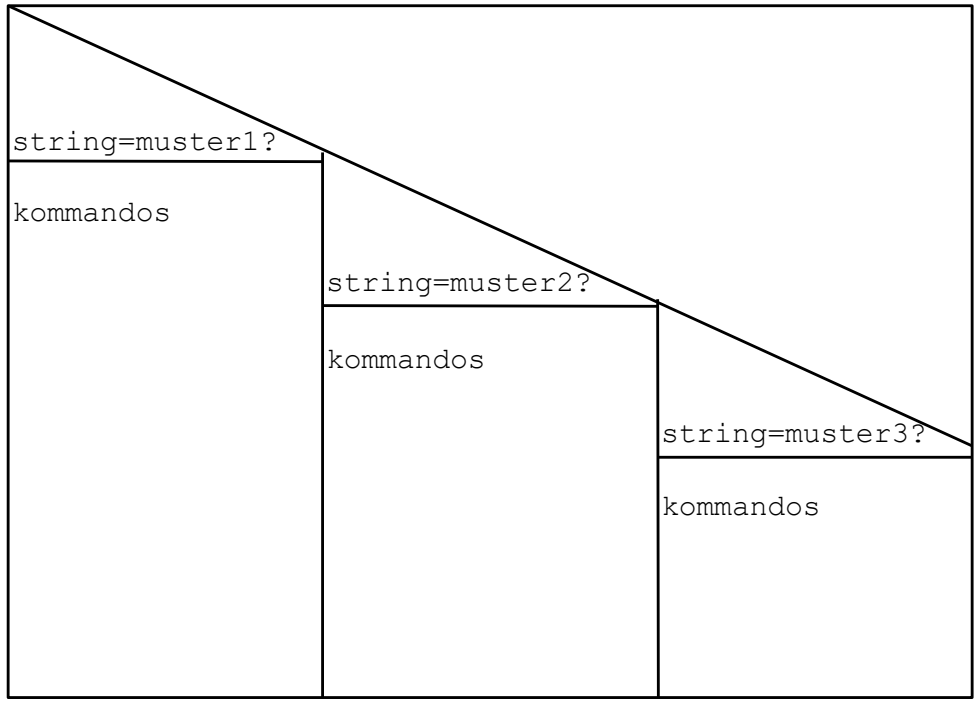
Verschachtelte if – Anweisungen (2)



case - Anweisung

Syntax

```
case string in
        muster1) kommando(s) ;;
        muster2) kommando(s) ;;
        muster3) kommando(s) ;;
        ...
                * ) kommando(s) ;;
esac
```



7

KORN-SHELL: ABLAUFSTEUERUNG 2 – SCHLEIFEN

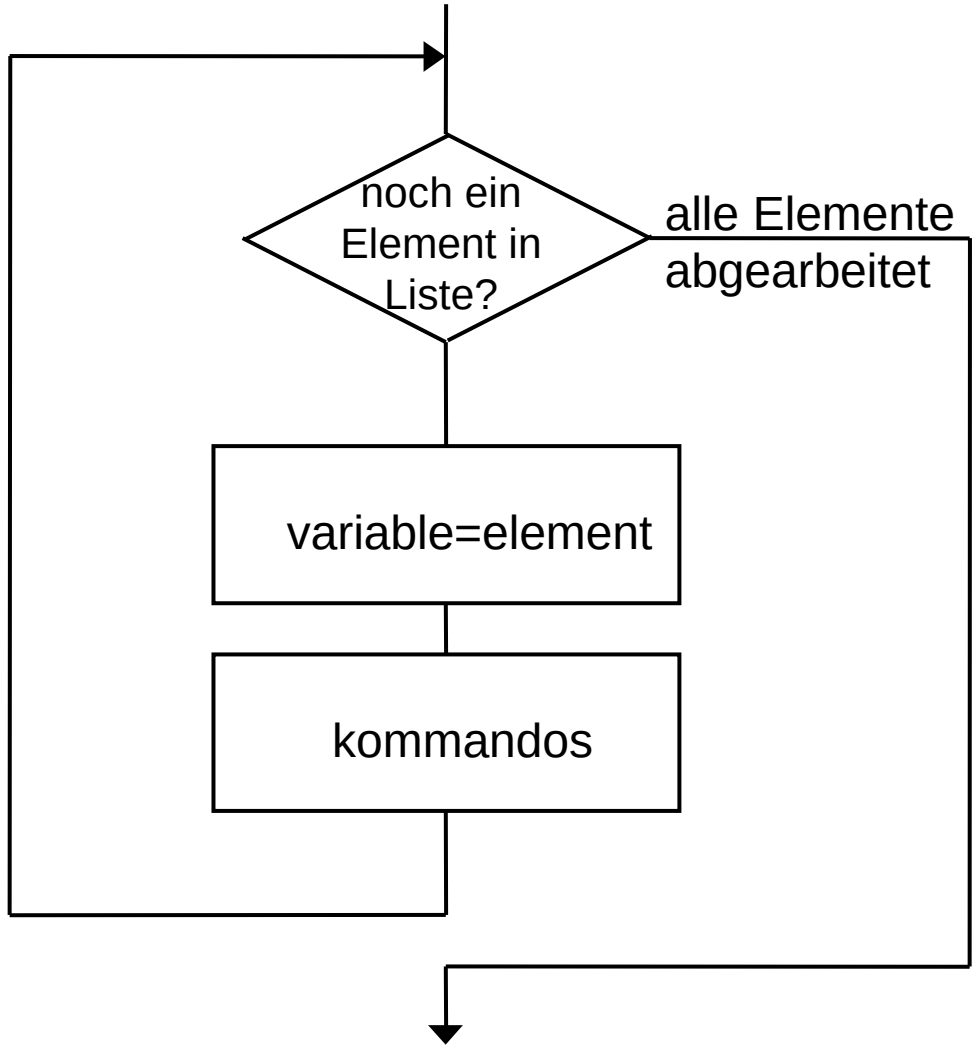
for - Schleife

Syntax:

```
for variable in element1 element2 ...
do
        kommando(s)
done
```

- zur Verarbeitung von Argumentlisten
- der Variable variable werden nacheinander alle Elemente zugewiesen
- Anzahl der Schleifendurchläufe = Anzahl der Elemente

for - Schleife (2)

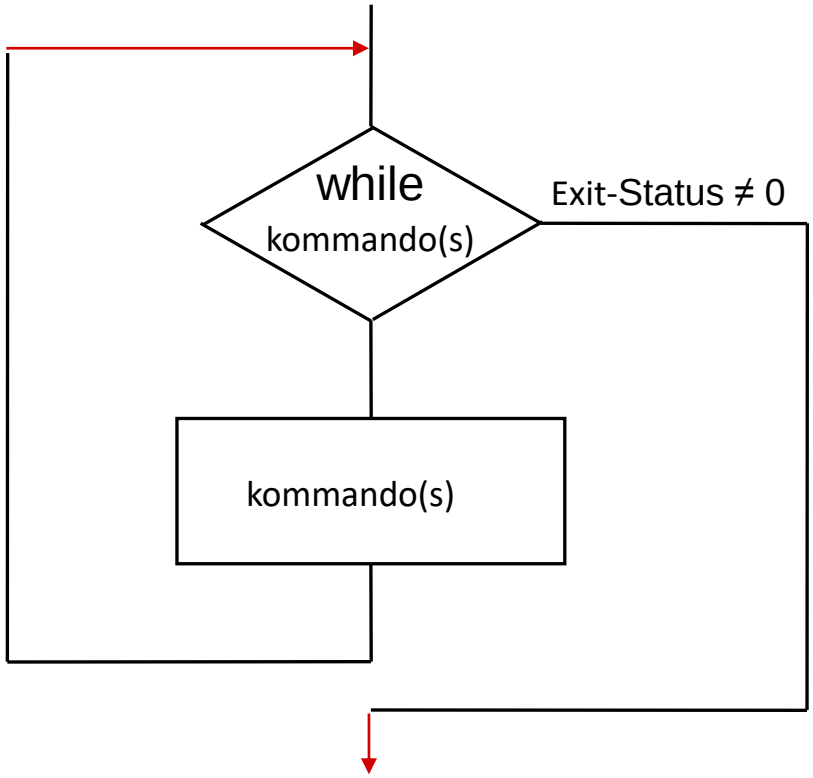


while - Schleife

Syntax:

```
while kommando(s)
do
        kommando(s)
done
```

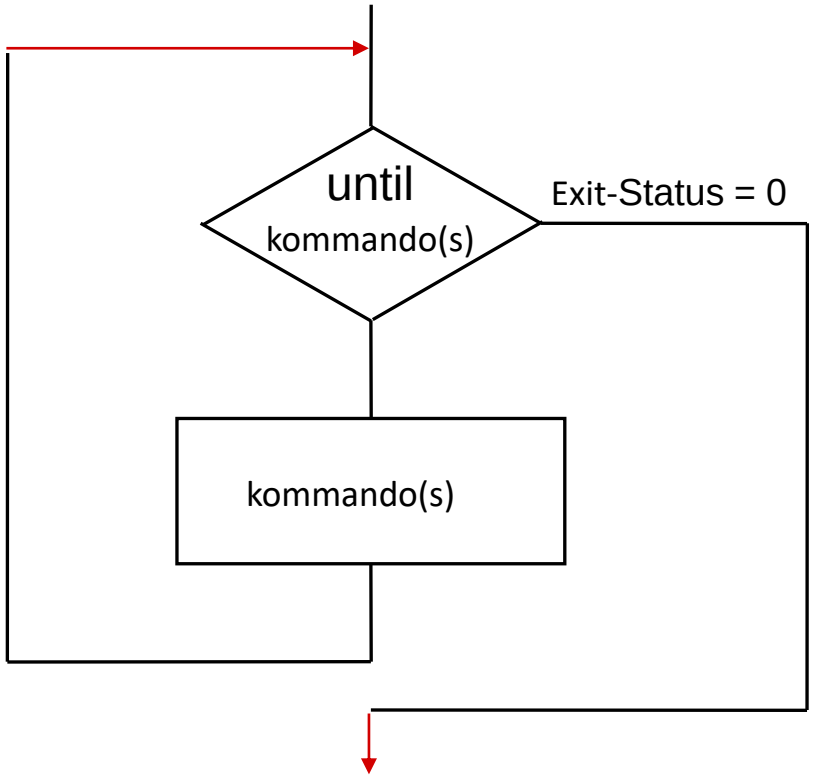
Die while-Schleife läuft, solange der Exit-Status eines Kommandos gleich 0 ist.



until - Schleife

Syntax:

```
until kommando(s)
do
        kommando(s)
done
```



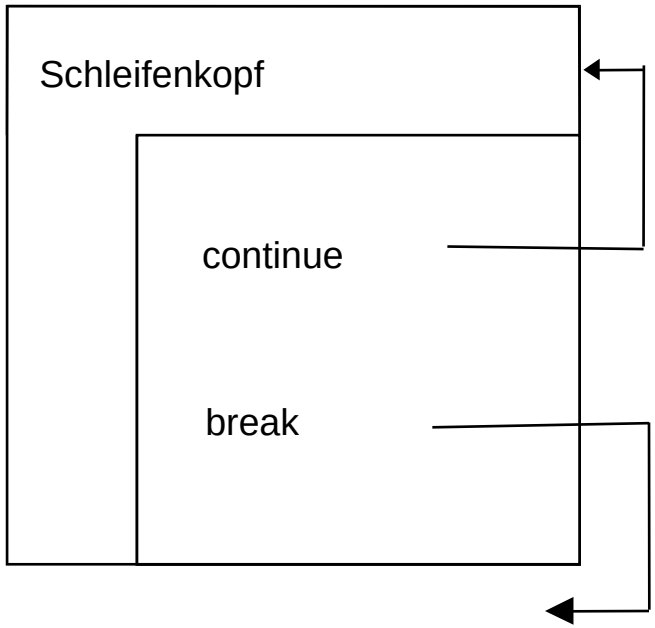
Die until-Schleife läuft, solange der Exit-Status eines Kommandos ungleich 0 ist.

Schleifenunterbrechung

Syntax:

break *[n]*
continue *[n]*

n entspricht der Anzahl der Schleifenebenen, die unterbrochen werden sollen (Default: 1).



Abbruch des aktuellen Durchgangs, Sprung zum Schleifenkopf

Abbruch der Schleife, Sprung hinter Schleifenende

8

KORN-SHELL: ARITHMETIK UND FELDER

Arithmetik

- In der Korn-Shell integriert

Syntax:

((...)) oder let ” ... ”	Stille Berechnung oder Vergleich
$((...))	Berechnung und Rückgabe des Ergebnisses

Arithmetische-Operatoren
Vergleichs-Operatoren

integrata cegos

Rechenoperator	Bedeutung
+	Addition
-	Subtraktion
*	Multiplikation
/	Division
%	Modulo (Rest der Division)

Vergleichsoperator	Bedeutung
<	kleiner
>	größer
<=	kleiner gleich
>=	größer gleich
==	gleich
!=	ungleich

Logische Verknüpfungen
Bit-Operatoren

Verknüpfungen	Bedeutung
(...)	Gruppierung
!	logische Negation
&&	logisches UND
||	logisches ODER

Bit-Operator	Bedeutung
&	Bitweise UND
|	Bitweise ODER
^	Bitweise exklusives ODER
~	Bitweise Negation
<<	Bit-shift links
>>	Bit-shift rechts

Operator-Priorität

Operatoren	Assoziativität
( )	→
- ! ~ (unär)	←
* / %	→
+ -	→
<< >>	→
<= >= < >	→
== !=	→
&	→
^	→
|	→
&&	→
||	→
= *= /= %= += -= <<= >>= &= ^= |=	←

Variablentypen

Syntax:

integer *variable*
typeset [*-opt*] *var[=wert]*

Option	Bedeutung
-i *n*	Wie Schlüsselwort *integer*: Beschleunigt die Korn-Shell Arithmetik und bringt Fehler bei Zuweisung nichtnumerischer Werte. Ist *n* angegeben, ist dies die Ausgabebasis (2 bis 36 – nur in ksh implementiert) Ohne *n* wird der Alias *integer* bevorzugt.
-r	Variable ist readonly (identisch mit Alias *readonly*)

Felder / Arrays

- Eindimensional
- Maximal 1024 Elemente (Index beginnt bei 0)
- In der Praxis reduzierte Einsatzmöglichkeit
- Wertzuweisung

  Syntax:

set –A *array element1 element2*	(Gesamtzuweisung)
*array***[***index***]**=*element*	(Einzelwert-Zuweisung)

- Zugriff

  Syntax:

${*array***[***index***]}**			(Einzelnes Element)
${*array***[*]}**	oder	**${***array***[@]}**	(Liste aller Array-Element)
${#*array***[*]}**	oder	**${#***array***[@]}**	(Anzahl definierter Elemente)

9

KORN-SHELL: ABLAUFSTEUERUNG 3 - SPEZIALSCHLEIFEN

Select-Schleife

Syntax:

```
select name   in  menupunkt1 menupunkt2 …
do
        kommando(s)
done
```

- Erstellt einfache Auswahlmenüs in Form einer Endlosschleife
- Prompt-Variable PS3 enthält Eingabeaufforderung
- REPLY enthält Eingabe des Benutzers
- *name* enthält Text des zugehörigen Menüpunktes (sofern zulässige Nummer eingegeben)
- Aussprung über break oder exit

While-Schleife mit getopts

Syntax:

getopts *optionstring variable* [*argumentliste*]

- Verarbeiten von übergebenen Optionen
- In der Regel zusammen mit *while*-Schleife; dort als Bedingung eingesetzt
- Verarbeitet beliebig viele Optionen mit oder ohne zugehörige Argumenten
- Pro Aufruf wird eine Option verarbeitet

While-Schleife mit getopts (2)

integrata cegos

Argumente Variablen	Beschreibung
optionstring	Beschreibung der gültigen Optionen: a einfache Option -a ohne Argument b: Option -b mit Argument, z.B. *-b x* Argument x zur Option b steht in der Variablen *OPTARG*
variable	Name der Variablen, in der jede gültige gefundene Option (z. B. *a*) steht. Bei einer unbekannten Option wird *variable* ein Fragezeichen zugewiesen.
OPTARG	Standardvariable: Enthält das Argument zu einer Option
OPTIND	Standardvariable: Ab der Stelle *OPTIND* enthalten die Argumente keine Optionen mehr.

10

KORN-SHELL: FUNKTIONEN

Funktionen

- Funktions-Definition (in eigener Datei, auch mehrere möglich):

Bourne-Shell/Korn-Shell

```
fktname( )
{
    kommando(s)
}
```

Korn-Shell

```
function fktname
{
    kommando(s)
}
```

- Funktions-Deklarartion (im Skript):

```
. pfad_funktions_definitions_datei
```

- Funktions-Aufruf (im Skript):

```
fktname  [argliste]
```

Funktionen (2)

- Unterschied zum Skript

  ⇒ kein separater Prozess (schneller)

  ⇒ Übergabeparameter **$1, ..., $*, $#** wie beim Skriptaufruf
  (überdecken Positionsparameter des Skripts)

  ⇒ Exit-Status definierbar über: **return *n***

  ⇒ Sichtbarkeit der Variablen standardmäßig global (Namenskonflikte)

  ⇒ funktionslokale Variable definierbar: **typeset *var*** (überdecken globale Variable)

- Funktion löschen: unset -f *fktname*

- Funktionen anzeigen:

functions	Ausgabe aller Funktionsdefinitionen
functions *fktname*	Ausgabe der Definition von *fktname*
typeset +f	Ausgabe aller Funktionsnamen

Autoload-Funktionen

- Directory für alle Autoload-Funktionen anlegen

  $ mkdir ~/*Fkt_Directory1*

- Funktion in einer separaten Datei im Directory für die Autoload-Funktionen definieren Dateiname = Funktionsname!!!

  $ vi ~/*Fkt_Directory1*/*fkt_name*

  fkt_name()

  { .....

  }

- In der ENV-Datei .kshrc Variable FPATH definieren und Autoload-Funktion aktivieren

  export FPATH=~/ *Fkt_Directory1*[:~/*Fkt_Directory2*]

  autoload *fkt_name* ....

- Für sofortige Verfügbarkeit Initialisierungsdatei laden

  $. ~/.kshrc

- Kontrolle

  $ functions [*fkt_name*]

11

KORN-SHELL: PROZESSSTEUERUNG

Signale

- Wichtige Signale:

Signal	Nr.	Bedeutung
HUP	**1**	Hang up (Hörer auflegen) Dämonprozesse: Konfigurationsdatei neu einlesen
INT	**2**	Interrupt: Eingabe von CTRL-C
QUIT	**3**	Quit: Erzeugt über CTRL-\
KILL	**9**	Kill. Prozess wird sofort beendet
TERM	**15**	Terminate: Normale Beendigung eines Prozesses

Signale senden und behandeln

- Signale senden – der Befehl kill

  Syntax:

  kill *[-signalnummer / -signalname] PID*

- Signale behandeln - der Befehl trap

  Syntax:

trap '*kommando*' *sig1* [*sig2* ...]	(Signal neu definieren)
trap ' ' *sig1* [*sig2* ...]	(Signal ignorieren)
trap *sig1* [*sig2* ...]	(Defaultbehandlung einstellen)
trap	(Umdefinierte Signal anzeigen)

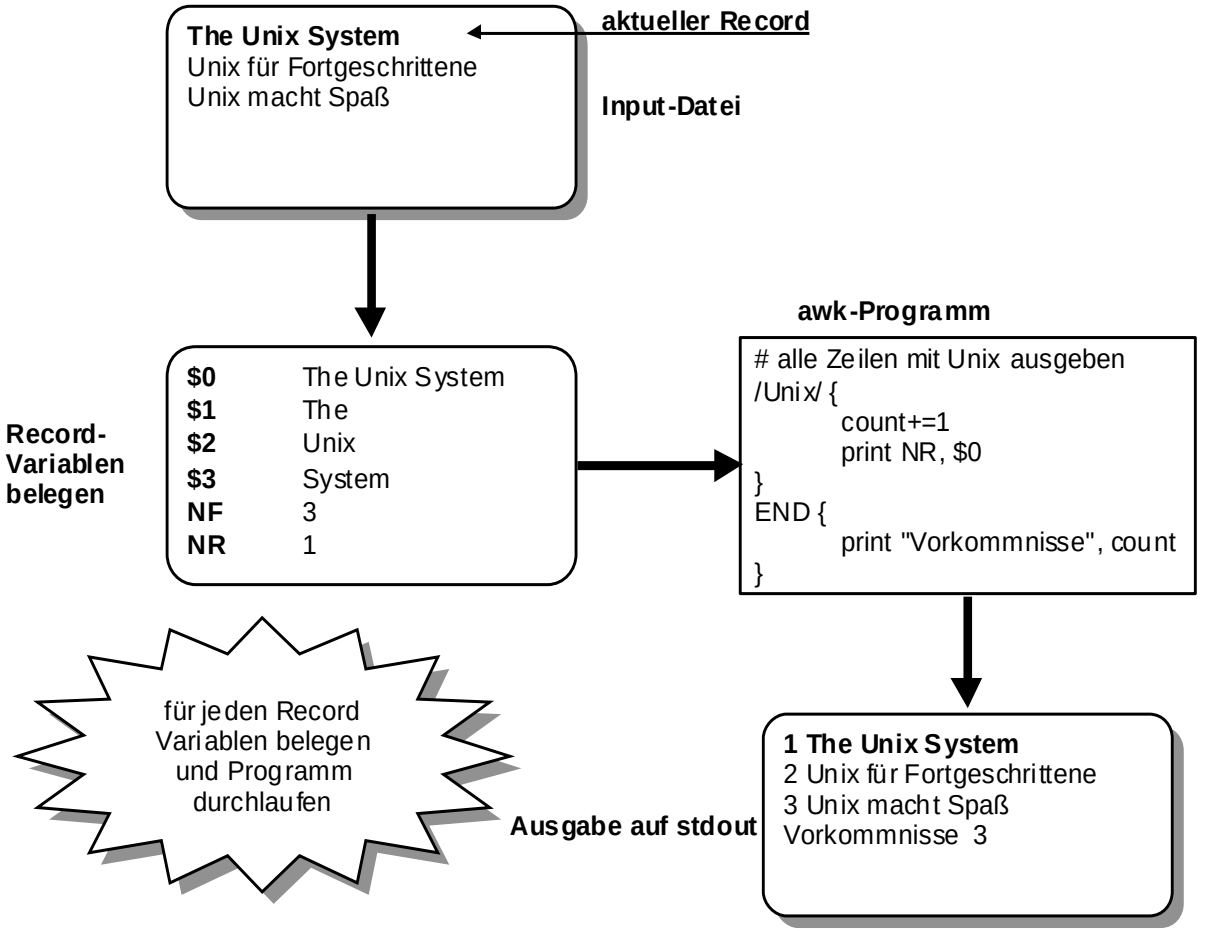
12

AWK: GRUNDLAGEN

Überblick

- Anwendung des awk
  - Listenausgaben
  - Statistische Problemstellungen
  - Prüfen von Daten auf syntaktische und semantische Korrektheit
  - Eingabedaten neu gruppieren, formatieren
- Merkmale des awk
  - Eingabedaten werden automatisch in Records (Zeilen) und Felder (Worte) strukturiert
  - Unterstützung von Gleitkomma- und Stringvariable
  - Arithmetische und Stringoperatoren
  - Allgemeine Programmierkonstrukte (Schleifen, Bedingungen)
  - UNIX-Kommandos sind ausführbar und die Ergebnisse können weiterverarbeitet werden
  - Programmiersprache ähnlich zu C.

Funktionsweise



Aufruf-Syntax

Syntax:

awk [**-F** *ERE*] [**-v** *var=wert*] ‘ *awkprogramm* **'** [*var***=***wert*] [*dat1* ...]

awk [**-F** *ERE*] [**-v** *var=wert*] **-f** *progfile* [*var***=***wert*] [*dat1* ...]

Parameter	Bedeutung
-F *ERE*	Zur Definition des Feldtrennzeichens Besonderheit: hier ist ein *ERE* möglich (siehe grep –E) Standard: Leerzeichen und/oder TAB
'*awkprogramm* **'**	Das eigentliche *awk*-Programm. kann auf mehrere Zeilen verteilt werden
-f *progfile*	*awk*-Programm steht in Datei *progfile* (nur read-Recht erforderlich)
dat1 **...**	Eingabedateien - ohne Datei liest *awk* von Kanal 0
-v *var=wert*	Definition einer Variable, auf die dann im awk-Programm zugegriffen werden kann; Variable ist bereits in der Initialisierungsphase des Programms verfügbar
var=wert	Definition einer Variable, auf die erst im Hauptteil des awk-Programms zugegriffen werden kann

Struktur des awk - Programms

```
BEGIN       { start_aktion(en) }                    # optionale Init.-Phase

kriterium_1 { aktion(en)_1 }        # Beginn des Hauptteils

kriterium_2 { aktion(en)_2 }

kriterium_3                         # Standard-Aktion

            { aktion(en)_4 }        # globale Aktion

.........               .........

kriterium_n { aktion(en)_n }        # Ende des Hauptteils

END         { ende_aktion(en) }                     # optionale Abschluss-Phase
```

Kriterium – reguläre Ausdrücke

integrata cegos

Regulärer Ausdruck	Bedeutung
^	Record/Feldanfang
$	Record /Feldende
.	ein beliebiges Zeichen
[...]	Zeichenauswahl zulässig
[^ ...]	Zeichenauswahl nicht zulässig
*****	0 bis *n*-malige Wiederholung des vorangehenden Ausdrucks
+	1 bis *n*-malige Wiederholung des vorangehenden Ausdrucks
?	0 oder 1-malige Wiederholung des vorangehenden Ausdrucks
{*n*,*m*}	*n* bis *m*-malige Wiederholung des vorangehenden Ausdrucks
.*	beliebiger String (auch Leerstring)
*****m	maskiert Metazeichen *m* (z. B. **\.**)
...|...	ODER-Verknüpfung zwischen Ausdrücken
(...)	gruppiert mehrere Zeichen zu einem Ausdruck

Kriterium – relationale Ausdrücke

Operator	Bedeutung
<	kleiner
>	größer
<=	kleiner gleich
>=	größer gleich
==	gleich
!=	ungleich
str ~ /RE/	Regulärer Ausdruck *RE* ist in String *str* enthalten
***str* !~ /RE/**	Regulärer Ausdruck *RE* ist in String *str nicht* enthalten

Kriterium – Auswahlbereich logische Verknüpfungen

integrata cegos

Ausdruck	Bedeutung
/RE1/, /RE2/	jeweils vom ersten Record, der RE1 enthält, bis zum ersten Record, der RE2 enthält (auch mehrfach)
Operator	**Bedeutung**
(...)	Priorität setzen
!	Negation
&&	UND-Verknüpfung
||	ODER-Verknüpfung

Variablen

- Zwei Typen:
  - String
  - Floating-Point / Integer
- Stringkonstanten müssen immer in Anführungszeichen stehen
- Keine Deklaration nötig
- Variablen-Interpretation nach Kontext
- Für das Aneinanderhängen von Strings kann bei einer Variablen-Zuweisung die automatische String-Verkettung des awk ausgenutzt werden.

Format der Eingabedatei

- Record-Variablen

Variable	Bedeutung
$0	aktueller Record
$1	erstes Feld
$2	zweites Feld
$*n*	Feld *n*
NF	Anzahl der Felder im aktuellen Record
NR	Nummer des aktuellen Records (Zeilennummer). Wird bei mehreren Dateien fortgezählt.
FNR	Nummer des aktuellen Records in der aktuellen Datei.
FILENAME	Name der aktuell gelesenen Datei

Format der Eingabedatei

- Trennzeichen-Variablen

Variable	Bedeutung
FS	Input Field Separator (Default: Leerzeichen, TAB) änderbar über: -F ERE oder FS="ERE"
OFS	Output Field Separator (Default: 1 Leerzeichen) änderbar über: OFS="String"
RS	Input Record Separator (Default: Newline) änderbar über: RS="ERE"
ORS	Output Record Separator (Default: Newline) änderbar über: ORS="String"

Benutzerspezifische Variablen

- Eigene Variablen-Bezeichner dürfen aus Buchstaben, Ziffern und Unterstrich gebildet werden, wobei das erste Zeichen keine Ziffer sein darf.
- Ohne explizite Definition werden sie bei der ersten Verwendung angelegt. Nicht explizit initialisierte Variablen werden automatisch initialisiert. Der Initialwert ist abhängig vom Kontext:
  - Leerstring für Stringvariablen
  - 0 für numerische Variablen
- Für die Deklaration / Zuweisung einer Variablen existieren damit folgende Möglichkeiten:

Syntax:

var = 42	(Zuweisung einer Zahl)
var = ”String”	(Zuweisung eines Strings)
var = var2	(Zuweisung einer anderen Variable)
var = var2 ”String”	(Zuweisung eines Strings bestehend aus den Werten von *var2* und “String”)

Strings – Escape-Sequenzen

- Strings
  - Strings stehen zur Abgrenzung gegenüber Variablen in Anführungszeichen "..."
  - Innerhalb von Strings sind folgende Escape-Sequenzen erlaubt

Escape-Sequenz	Bedeutung
\a	Piep (Alert)
\b	Backspace
\f	Formfeed
\n	Newline
\r	Carriage return
\t	Tabulator
\v	Vertical Tabulator
ooo	Zeichen mit Oktalwert *ooo*
\c	irgendein Zeichen Literal (Beispiel: \\)

Das print-Kommando

print

- schreibt den aktuellen Record auf Standard-Ausgabe und schließt die Ausgabe mit dem ORS ab

print ausdr1 ausdr2 ....

- schreibt die Ausdruckswerte direkt verkettet (ohne Trennzeichen) auf Standard-Ausgabe und schließt die Ausgabe mit dem ORS ab

print ausdr1 , ausdr2 ....

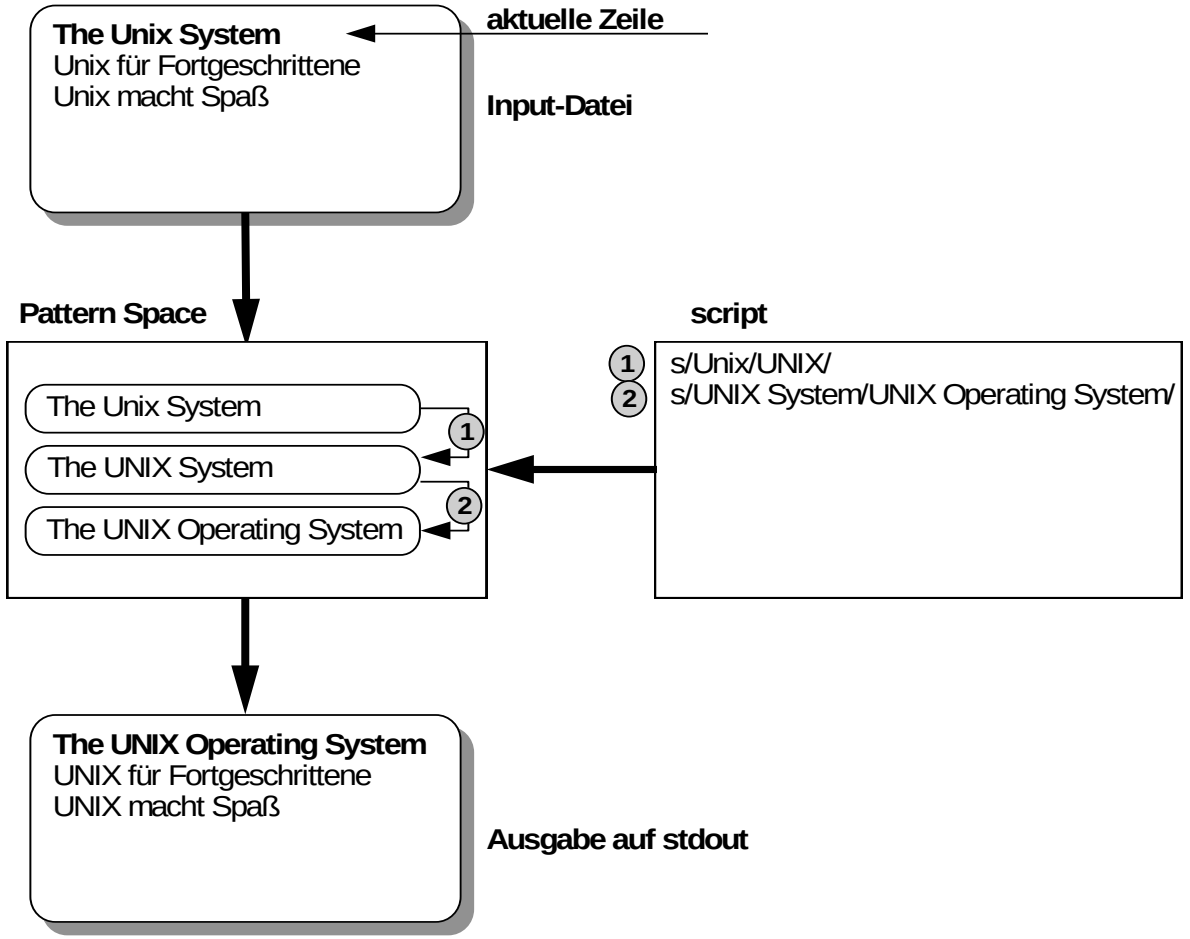
- schreibt die Ausdruckswerte getrennt durch den OFS auf Standard-Ausgabe und schließt die Ausgabe mit dem ORS ab

13

SED

Funktionsweise von sed

- Zeilenweise einlesen der Eingabe
- Auf jede Zeile alle Editoranweisungen ausführen
- Ergebnis auf *stdout* ausgeben
- Originaldaten werden nicht verändert



Aufruf von sed

Syntax:

sed *[* -n *]* ' *sedprogramm* ' *[dat1 ...]*
sed *[* -n *]* -e '*sedprog1*' -e '*sedprog2*' *[dat1 ...]*
sed *[* -n *]* -f *progfile [dat1 ...]*

Parameter	Bedeutung
-n	Defaultausgabe auf Kanal 1 wird unterdrückt. Explizite Ausgabe mit Kommando p (print) möglich.
'*sedprogramm*'	Das eigentliche sed-Programm. Kann auf mehrere Zeilen verteilt sein.
dat1 ...	Eingabedateien Ohne Datei liest der *sed* von *stdin* / Kanal 0
-f *progfile*	sed-Programm steht in Datei *progfile* (selten benutzt)
-e '*sedprog*'	Zum Spezifizieren mehrerer Editierkommandos in einer Zeile oder zum Mischen mit *-f* Optionen

Editierbefehle zu sed

- Zeileneinschränkung

  Syntax:

  [adr1 [,adr2]] **kdo** *[argumente]*

Adresse	Bedeutung
<fehlt>	Global, jede Zeile
1	Zeile 1
5,15	Zeile 5 bis Zeile 15
50,$	von Zeile 50 bis zur letzten Zeile

Befehlsübersicht zu sed

Befehl	Bedeutung
d	delete: Zeile löschen. Die nächste Zeile wird gelesen und ein neuer Zyklus beginnt mit dem ersten Kommando.
p	print: Ausgeben der Zeile. Ohne die n-Option wird hiermit eine Zeile dupliziert.
a\ *text*	append: Schreibt *text* auf stdout nach der adressierten Zeile. Zeilenende mit \ quoten.
i\ *text*	insert: Schreibt *text* vor die adressierte Zeile.
c\ *text*	change: Löscht die adressierten Zeilen, ersetzt sie durch *text* und beendet den aktuellen Zyklus

Befehlsübersicht zu sed (2)

integrata cegos

Befehl	Bedeutung
s*/alt/neu/flag*	**substitute: Ersetzt *alt* in der Zeile durch *neu* Flags sind:** **<leer> nur erstes Auftreten ersetzen** **g alle Vorkommnisse ersetzen (global)** ***n* *n*-tes Vorkommnis ersetzen** **p Zeile ausgeben, wenn Ersetzung durchgeführt wurde (bei mehreren Ersetzungen wird die Zeile mehrfach ausgegeben)** **w *file* Zeile in *file* schreiben** **Sonderzeichen im Ersetzungsteil *neu*:** **& Füge gefundenen Text (*alt*) ein**
=	**Ausgabe der Zeilennummer**
q	**quit: sed verlassen. Nur sinnvoll mit Einzeladresse**
#kommentar	**Kommentarzeile**

Reguläre Ausdrücke von sed

Text-Suchmuster mit Freiheitsgraden, ähnlich Wildcards, an zwei Stellen einsetzbar:

1. Zeilenadressierung
   Hierfür werden die regulären Ausdrücke in Slashes eingeschlossen: /.../
2. Suchen und Ersetzen

regulärer Ausdruck	Bedeutung
^	Zeilenanfang
$	Zeilenende
.	ein beliebiges Zeichen
[a-c]	eines der Zeichen a, b oder c
[^a-c]	ein Zeichen, jedoch nicht a, b oder c
*****	0 bis *n*-malige Wiederholung des vorangehenden Zeichens
.*	beliebige Zeichenfolge
\x	maskiert Metazeichen *x* (z. B. \.)

Copyright und Impressum

Integrata Cegos GmbH
Zettachring 4
70567 Stuttgart