Zusammenfassung Unix – Werkzeuge

1. RPM – Red Hat Paketmanager

- seit 1995 für automatische Konfiguration von Paketen bei Installation
- anfangs langsam (Perl)
- sorgt dafür, dass alle installierten Programme zusammenpassen
- mittlerweile mit Datenbank
    - installierte Pakete mit Files
    - Versionsnummern
    - Abhängigkeiten der Pakete untereinander
    - unter /var/lib/rpm/

- **Installation von Paketen**
    - rpm -i [Optionen] filename0.rpm {filename.rpm}
      (--install)
    - prüft:
        - alle benötigten Pakete installiert?
            - Konflikte mit anderen Paketen?
            - schon vorhanden?
        - Ablauf:
            - pre-installation-scripte
            - entpacken
            - Files an vorgegebene Orte
            - post-installations-scripte
            - Behandlung von Konfigurationsfiles
            - RPM-Datenbank aktualisieren
        - Installation über Netzwerk möglich
    - **Optionen:**

--test	(nur Test)
--replacepkgs	(ersetzen des Pakets durch neue Kopie)
--replacefiles	(Installieren auch bei Konflikten)
--force	(Datei- und Paketkonflikte ignorieren, Abhängigkeiten nicht)
--nodeps	(keine Abhängigkeiten)
--root <patth>	(setze Wurzellverzeichnis auf path)
--dbpath <path>	(path für Suche der RPM-Datenbank)
--justdb	(nur verändern der Datenbank)

- **Löschen**
    - rpm -e [Optionen] paketname0 {paketname}
      (--erase)
    - Ablauf:
        - prüft Abhängigkeiten von anderen Paketen
        - Löschscripte
        - Löschen der Files
        - eventuell Modifikation Config-Files
        - entfernen aus Datenbank

    - **Optionen:**
        - ähnlich wie bei Installation
        --allmatches (löschen von Paketnamen, Version egal)

- Update
    - rpm -U [Optionen] filename0.rpm {filename.rpm}
    - Ablauf:
        - Installation
        - löschen des alten
    - **Optionen:**
        --oldpackage (frühere Version installieren)

-Informationen
    - rpm -q [Optionen] [Paketname]
    - Optionen:
        --whatrequires <cap> (Pakete, die cap benötigen)
        --whatprovides <cap> (Pakete, die cap bereitstellen)

- Überprüfung
    - rpm -v [Optionen] paket1...paketn
        (--verify)
    - **Optionen:**
        - f <file> (überprüft Pakte mit <file>)

- spezielle Optionen
    --pipe <Kommando> (Ausgabe an Pipe)
    --initdb (Anfangszustand RPM-Datenbank)

- Spec
    - Beschreibung zum Bilden und Installieren

    - Präambel (%description) (Informationen über Paket)
    - Prep-Sektion (Informationen über Vorbereitung des Baus)
    - Build-Sektion (Befehle zum Bilden des Binary)
    - Install-Sektion (Installation der gebildeten Software)
    - File-List-Sektion

- Paket bilden
    - rpmbuild -b<Stufe> [Optionen] <spec-file> {spec-file}
    - **Stufe:**

p	%prep ausführen	patch
c	%prep + build	compile
i	%prep + build + %install	install
b	%prep + build + %install + Binarypaket	binary
a	%prep + build + %install + Binary- + Sourcepaket	all
l	prüft %files	list
s	%prep + build + %install + Sourcepaket	source

    - **Optionen:**
        --buildroot <path> (path als Wurzelverzeichnis)

- **Umgebung** für Bau: SOURCES, SPECS, BUILD, RPMS, SRPMS

- Macros
    %setup   - entpackt Quellarchive, Vorbereitung für Bau
    %patch   - patchen der entpackten Quellarchive

2. make

- für Verwaltung von Projekten
- Formalisierung von Abhängigkeiten der Quelltexte
- 1. makefile 2. Makefile
- - unterdrückt sofortigen Abbruch nach Fehler eines Kommandos

Zielobjekt: {Quellobjekt}
    {Kommando}

- Ziel: alle Aktionen für Bildung des Ziel werden ausgeführt
- Quelle: Ziel hängt von dieser ab
- Kommando: für Bildung des Ziels aus Quelle
- bei „:“ Zielobjekt kann mehrfach auf linken Seite stehen, aber nur einmal ein Kommando
- bei „::“ mehrere Kommandoteile möglich
- Makros möglich **Makroname = String**
- Zugriff über **${Makroname}** oder **$(Makroname)**

	e	!e
- **4 Makroarten**:		
- Nutzerdefinierte Makros	3.	2.
- Umgebungsvariablen der Shell	2.	3.
- Schlüsselwortparameter beim Aufruf von make	1.	1.
- interne Makros	4.	4.

- **interne Makros**:
    $@   Name des ungültigen Zielobjektes (was erstellt werden soll)
    $(@D)   Pathname
    $(@F)   Filename
    $?   Namen aller ungültigen Quellen neuer als Ziel
    $<   Name des Objekts, das übersetzt werden soll
    $(<D) $(<F)
    $*   Name ungültigen Ziel ohne Suffix, was Transformation ursprünglich ausgelöst hat
    $(*D) $(*F)
    $%   wenn Ziel Bibliothek (lib.a(file.o)), dann $%=file.o und $@ lib.a
    $(%D) $(%F)

- make hat interne **Suffixregeln**, wenn keine explizite Bildungsvorschrift vorhanden ist (werden durch . SUFFIXES: gelöscht)

- .DEFAULT – wenn keine Regel zur Erstellung eines Objektes vorhanden
- .IGNORE – Fehler ignorieren
- .PRECIOUS – Ziele, die bei Fehlern nicht gelöscht werden (Bibliotheken)

- **Optionen:**
  - `-e` Umgebungsvariablen überschreiben makedateien
  - `-f` makedatei
  - `-j [N]` erlaube N Jobs gleichzeitig
  - `-t` touch Targets
  - `-W Datei` Datei als immer neu ansehen

- einige **Funktionen:**

`$(findstring find, text)`	suche find in text
`$(sort list)`	sortieren, entferne doppelte
`$(words text)`	Anzahl Wörter in text
`$(suffix names...)`	Suffixes von names...
`$(basename names...)`	names... ohne Suffixes
`$(shell kommando)`	führt in Shell Kommando aus und gibt Rückgabewert aus
`$(foreach var, words, text)`	

3. CVS

- = Concurrent Versions System
- Versionsverwaltungssystem
- was? - Textfiles (Quelltexte...), aber keine Binarys

- **Ziele:**
  - Kontrollierter Zugriff
  - frühere Versionen herstellen
  - kontinuierliche Generationsfolge sicheern
  - Dokumentation der Änderungen
  - paralleler Zugriff (Synchronisation)
  - Speicherplatz sparen
  - Kosten reduzieren
  - Zeit sparen

- **Repository:**
  - Speicher für die Daten
  - mehrere Versionen aufgehoben (wiederherstellbar)

- **Sandbox:**
  - Kopie des Repository, auf der gearbeitet wird

- Datenbasis initialisieren: **init**
- Datenbasis mit Anfangszustand füllen: **import**
- Ausgabe an Nutzer: **checkout**
- Einfügen in Datenbasis: **commit**
- neue Version bilden: **import**
- Ausgabe vollständige Version: **update**

- mehrere Nutzer können gleichzeitig an selben Datei arbeiten
(Konflikt erst bei Rückgabe bemerkt! zweite wird unterdrückt!)
- Sperrmechanismus durch Admin möglich

- für jede Datei weitere Informationen
  - allgemeine Beschreibung

  - Unterschiede zu Vorgängern
  - Änderungsdaten
  - Kommentare zu Änderungen
  - Name des Verantwortlichen für Änderungen
  - Versionsnummer
- einmalig **cvs init** nötig (CVSROOT muss als Variable gesetzt sein)

- **Import:**
  cvs import -m „Beschreibung" Repository vendorTag releaseTag
  Bsp.:
  cvs import -m „ein Test" Test Alex default
  -importiert alle Files aus dem aktuellen Verzeichnis in ein Repository Test mit Versionsnummer default (1.1)

- **Checkout:**
  - **cvs checkout Test**
  - holt alle Dateien aus dem Repository Test in das aktuelle Verzeichnis
  - mit checkout Test/Beispiel einzelne Datei extrahiert

- **Update:**
  - **cvs update Makefile**
  - Änderungen am Makefile mit dem Repository abgleichen
  - falls Konflikte, müssen diese vom User behoben werden
  - ohne Angabe, werden alle Dateien aus dem aktuellen Verzeichnis genommen

- **annotate:**
  - **cvs annotate -r 1.1 Makefile**
  - zeigt alle Zeilen der spezifizierten Objekte an
  - mit Änderungsdatum und Nutzer
  -r gibt an, welche Version angezeigt werden soll

- **Commit:**
  - **cvs update Makefile**
  - Makefile aus Sandbox in Repository übertragen
  - Änderungen müssen in einem Editor erfasst werden
  - ohne Angabe, werden alle Dateien aus dem aktuellen Verzeichnis genommen

- **log:**
  - **cvs log Makefile**
  - zeigt alle Versionen, die es gab
  - wann, wer und wieviel geändert
  - Kommentare Nutzer

- **add:**
  - **cvs add Makefile2**
  - vorbereiten des Hinzufügens von Datei/Directory zur Sandbox
  - durch commit in Repository übertragen
  - es muss Kommentar eingefügt werden

- **diff:**
    - **cvs diff -c -r 1.1 -r 1.2 Makefile**
    - zeigt Änderungen zwischen Version 1.1 und 1.2 an und gibt Zeilen an

- **export:**
    - **cvs export Makefile**
    - Dateien aus Repository kopiert, aber nicht in Sandbox

- **release:**
    - **cvs release Makefile**
    - Löschen einer Sandbox oder Teile davon

- **remove:**
    - **cvs remove Makefile**
    - Löschen einer Datei aus Repository
    - wird nicht wirklich gelöscht, kann mit add wiederhergestellt werden
    - muss mit commit dem Repository übergeben werden

- **status:**
    - **cvs status Makefile**
    - zeigt Informationen über Datei an

- **watch:**
    - **cvs watch Makefile**
    - setzt Beaobachtungspunkt auf Datei
    - mit watchers alle Beobachter anzeigen

- **edit:**
    - **cvs edit Makefile**
    - teilt anderen „watchern“ mit, dass an einer Datei gearbeitet wird
    - mit editors alle Bearbeiter anzeigen

- **unedit:**
    - **cvs edit Makefile**
    - beendet Dateibearbeitung

4. Reguläre Ausdrücke

- **DFA** (Deterministischer Finiter(endlicher) Automat
    - Textgesteuert – geht vom Text aus und prüft, ob Ausdruck passt
    - schnell
    - nicht von Programmierung des Ausdrucks abhängig
    - POSIX-Konform (verlangt längsten frühesten Treffer)
    - keine Klammern oder Rückwärtsreferenze

-**NFA** (Nichtdeterministischer Finiter Automat)

- Ausdrucksgesteuert - geht vom Ausdruck aus und prüft, ob Text passt
- Programmierer kann durch Ausdrucksgestaltung Arbeitsweise des Suchens beeinflussen
- leistungstärker, da Klammern erlaubt
- nicht immer PSOIX-Konform
- verschiedene Implementierungen liefern unterschiedliche Ergebnisse
- eventuell langsamer wegen Backtracking

- **Metazeichen:**

^	Zeilenanfang
$	Zeilenende
.	beliebiges Zeichen
\<	Wortanfang
\>	Wortende
[...]	verlangt Zeichen aus der Menge
[^...]	verlangt Zeichen, die nicht in der Menge sind
?	vorangestellte Zeichen(klasse) 1 oder 0 mal
+	vorangestellte Zeichen(klasse) 1 oder mehrmals
*	vorangestellte Zeichen(klasse) 0 oder mehrmals
(...)	Gruppierung mehrerer Zeichen
|	trennt Suchmuster (jedes allein erfüllt Bedingung)
\1..\9	Rückwärtsreferenz auf in Klammern eingeschlossenen Text (max. 9)
x{m}	m-fache Wiederholung von x
x{m,}	mindestens m-fache Wiederholung von x
x{m,n}	mindestens m-fache, maximal n-fache Wiederholung x

5. sed

- Stream-Editor (häufig in Scripten zur Modifizierung von Daten eingesetzt)
- schneller als awk, aber nicht so leistungsfähig
- liest Editierkommandos von Kommandozeile oder aus Datei
- liest zu editierende Daten von Standardeingabe oder aus Datei
- Ausgabe immer auf Standardausgabe
- Arbeitsweise:
    1. einlesen Editierkommandos
    2. einlesen erste Zeile der Eingabedatei in Puffer
    3. ausführen der Editierkommandos, wenn möglich
    4. Ergebnisse aller Kommandos auf Standardausgabe
    5. Eingabepuffer löschen
    6. neue Zeile wenn vorhanden, dann weiter bei 3.

- **Aufruf:**

  sed [Optionen] [-e <Editierkommando>] [-f <Scriptdatei>] {Filename}

  - mehrere Dateinamen möglich
  - wenn keine Angabe, dann Standardeingabe

- **Optionen:**

-n	Zeile wird nur bei explizitem p-Kommando ausgegeben

- **Kommandos:**
  - a — fügt 1 oder mehrere Zeilen an aktuelle Zeile an
  - c — ersetzt Text in aktuellen Zeile
  - d — löscht Zeile(n) (mit 7)
    sed -e "/7/d" testfile2
  - g — kopiert temporären Puffer in Eingabepuffer
  - G — fügt temporären Puffer an Eingabepuffer
  - h — kopiert Eingabepuffer in temporären Puffer
  - H — fügt Eingabepuffer an temporären Puffer
  - i — fügt Text oberhalb der aktuellen Zeile ein
  - I — zeigt nichtdruckbare Zeichen
    sed -n -e l testfile2
  - n — wendet nächstes Kommando anstatt aktuellen Kommandos auf nächste Zeile an
  - p — druckt Zeile(n)
  - r — liest Zeilen aus einer Datei
  - ! — wendet Kommando auf Zeilen an, die nicht zutreffen

- **Substitutionskommandos:** (werden aus Kontext heraus gewählt)
  - g — globale Ersetzung jedes Vorkommens in der Zeile, sonst immer nur das erste Vorkommen bearbeitet!!!
  - p — Ausgabe der Zeile (s verschluckt Ausgabe!) (in der /RegExp/ ist)
    sed -n /hallo/p testfile2 (siehe -n)
  - s — Ersetzen eines Musters durch ein anderes
    sed 's/dies/das/[g]' testfile2 (Flag = g)
  - w — bearbeitete Zeile in Datei schreiben
    sed 's/dies/das/gw test' testfile2
  - x — tauscht Zwischenspeicher mit aktuellen Zeile
  - y — ersetzt ein Zeichen durch andere (Zeichenlisten müssen beide gleiche Länge haben)
    sed 'y/abcd/1234/' testfile2 (a=1,b=2...)

- **Reguläre Ausdrücke:** (Abweichungen)
  - Maskierung für Metazeichen!!!
  - kann auch Zeichenklassen!
  - & — enthält Suchmuster

- **Adressierung:**
  - ka — alle Zeilen
  - Nummer — genau diese Zeile
  - Start, Ende — diese Zeilen
  - $ — letzte Zeile
  - RegEx — Zeilen, die regulären Ausdruck enthalten
  - 1, RegEx — von 1 bis zur Zeile, die RegEx enthält

- **Kommandos für Ablaufsteuerung im sed-Script:**
  - b — branch (Sprung zur Marke)
  - t — test jump (bedingter Sprung)
  - ! — Verneinung der Adressierung
  - :ma — Marke im sed-Script
  - q — quit
  - = — Ausgabe der Zeilennummer als eigene Zeile

```
#         Kommentare
```

6. grep (egrep, fgrep, gnu-grep)

- **grep [Optionen] Muster Datei**
- zum Suchen in Files und geben Zeilen, die Pattern (nicht) enthalten aus
- bei mehreren Pattern, muss nur eins matchen
- unterscheiden sich in Optionen und Funktionalität und damit in Geschwindigkeit des Suchens
- grep: Pattern mit einfachen regulären Ausdrücken
- egrep (extended grep): Pattern mit vollen regulären Ausdrücken
- fgrep (fixed grep): Pattern mit Strings
- mehrere Files möglich (keins, dann Standardeingabe)

- **Optionen:**

-F	nur feste Strings
-E	volle reguläre Ausdrücke
-c	Anzahl der gefundenen Stellen
-h	Unterdrückt Filenamen vor jeder Stelle, bei mehreren Files
-i	ignoriere Groß- und Kleinschreibung
-l	gibt nur Dateinamen mit Fundstellen aus, ohne Wiederholung
-n	Zeilennummer zur Fundstelle (beginnt bei 1)
-v	nur Zeilen, die Pattern nicht enthalte
-w	nur Zeilen, die Pattern als komplettes Wort enthalten
-e list	mehrere Pattern durch Leerzeichen getrennt
-f File	enthält Pattern, nach denen gesucht wird
-q	Quiet, Rückgabe 0, wenn gefunden
-x	Zeilen, die Wort als ganze Zeile enthalten

- **Rückkehrwerte:**

0	mindestens 1 Zeile gefunden
1	keine Zeile gefunden
2	Syntaxfehler oder Datei nicht gefunden

7. find

- **find Verzeichnis [-Optionen] [-Test] [-Aktion]**

- **Optionen:**

-follow	folgt symbolischen Links

- **Tests:**

+ N	> N
- N	< N
N	genau N
-amin N	vor N Minuten auf Datei zugegriffen
-gid N	Datei gehört Gruppe mit Kennzahl N
-group Name	Datei gehört Gruppe Name
-name Dateiname	(Quoten bei Wildcards in Dateiname)

```
-perm 000           (Zugriffsrechte 000)
-perm -000          (Mindestzugriffsrechte 000)
-links n            (alle, die n links besitzen)
-ctime n            (innerhalb der letzten n Tage modifiziert
                    oder Dateiattribute geändert)
-newer fname        (alle, die neuer als fname sind)
-exec command       (alle, führt command aus, \; = Ende, {} =
                    Argument für command)
-user Name          Datei gehört Nutzer Name
```

- **Operatoren:**

```
(Ausduck)           fasst Ausdruck zu einer Operation zusammen
! Ausdruck          Negation
-and
```

8. awk

- awk [-F<Feldseparator>] 'awk-Programm' {<Datei>}

- awk [-F<Feldseparator>] -f 'awk-Programm-File' {<Datei>}

- Textmuster in Dateien finden und Aktionen ausführen
- grob C Syntax
- awk ist langsam
- Wertzuweisung über Kommando ist möglich

./test.awk n=3 Datei (test.awk = Script mit n)

- Variablen können in awk nur deklariert werden, der Typ hängt vom Kontext ab und awk initialisiert die Variablen automatisch leer oder 0

- bei Vergleichen unterschiedlicher Typen, werden Typen konvertiert
- einfaches Konvertieren durch Anhängen von „“ oder 0

- Verkettung von Strings: <string1><string2>

- Formatgesteuerte Ausgabe mit **printf** (wie bei C)
  - **Formatierung: %FW(.G)U**

```
F       Formatierungszeichen „-“ linksbündig
W       Mindestanzahl Auszugebener Zeichen
G       Genauigkeit
            Strings     Maximalzahl Zeichen
            Zahlen      Nachkommastellen
U       Umwandlungszeichen
            c           Ascii
            d           Dezimal
            s           String
            x           Hexa
            %           %-Zeichen
```

- **Arbeitsweise:**
  1. BEGIN
  2. zeilenweises Einlesen der Eingabedatei
     - wird in mehrere Eingabefelder zerteilt...$1, $2...
     - $0 ist Eingabezeile

   Ausführen der Aktionen, wenn Bedingung erfüllt ist
3. END

- **weitere Variablen:**

NF	Anzahl Wörter (Felder) in der Zeile
(F)NR	Nummer der aktuellen Zeile (FNR beginnt für jede neue Datei von vorn!)
FILENAME	Name der Inputdatei
FS	Feldseparator
ARGC	Anzahl Parameterliste
ARGV	Felde der Parameterliste
ENVIRON	Feld der Umgebungsvariablen
FILENAME	aktuell bearbeitete Datei
OFMT	Ausgabeformat für Zahlen („%.6g“ = Gleitkommazahl mit 6 Stellen nach Komma)
OFS	Trennzeichen für Ausgabefelder
ORS	Trennzeichen für Ausgabezeilen
RS	Trennzeichen für Eingabezeile

- **Anweisung:**

  Zeichenkette ~ /Muster/ {Kommandos;}

  - **Zeichenkette**
    - $1...
    - Standard ist $0 für ganze Zeile

  - **Muster**
    - optional
    - fehlt es, wird Aktion für jede Zeile ausgeführt
    - reguläre Ausdrücke
    - Variablenvergleiche
    - Zeichenklassen zusätzlich zu regulären Ausdrücken (weil nicht auf jedem System der Zeichensatz gleich sein muss(durchgängig)):

[[:digit:]]	alle Zahlen
[[:alpha:]]	alle Buchstaben
[[:alnum:]]	Alphanumerische Zeichen
[[:blank:]]	Whitspaces
[[:cntrl:]]	Steuerzeichen
[[:lower:]]	Kleinbuchstaben
[[:upper:]]	Großbuchstaben

  - **Kommandos:**
    - fehlt es, wird die Zeile ausgegeben

  - **BEGIN:**
    - meist Variableninitialisierung
    - am Anfang

  - **END:**
    - nachdem letzte Zeile der letzten Datei gelesen wurde

- **Operatoren:**

<(=)		n, s
>(=)		n, s
==	gleich	n, s
!=		n, s
~	rechter String (auch reguläre Ausdrücke erlaubt) in linkem enthalten	s
!~		s
?:	a?b:c wenn a, dann b, sonst c	
||	logisches oder	
&&	logisches und	
!	Verneinung	
in	n in array = True, wenn array[n] existiert	

- **getline**
  - nächste Zeile einlesen und fortfahren im Programm
- **getline VAR**
  - Zeile lesen und in VAR packen
- **getline < file**
  - Zeile aus file lesen
- **getline VAR < file**
  - Zeile aus file lesen und in VAR packen
- **"Kommando" | getline**
  - STDOUT des Kommandos einlesen
- **"Kommando" | getline Var**
  - STDOUT des Kommandos in die Var
- (close "Kommando": File Pointer in Kommandoausgabe wieder auf Anfang)
- Rückkehrwert: 1 = Zeile gelesen, 0 = EOF, -1 = Fehler

- **for (Ausdruck) { Anweisung }**

- **for (VAR in Feld) { Anweisung }**
  - für jedes Feldelement führe Anweisung aus

- **do { Anweisung } while (Ausdruck)**
  - Anweisung, Ausdruck berechnen, wenn != 0, dann von vorn

- **while (Ausdruck) { Anweisung }**
  - Ausdruck berechnen, wenn != 0, Anweisung und von vorn

- **break** (Verlassen der Schleife auf nächste Ebene)
- **continue** (Verlassen aktueller Anweisung, nächste Durchlauf)

- **next** liest nächste Zeile und setzt mit erstem Pattern des Programms fort

- **exit** springt zu END, wenn in END, dann sofortige Beendigung

- **built-in-Funktionen** wie cos(x), sin(x), int(x), rand(), srand(x)

(neuer Startwert für rand, funzt net)

- **string-buil-in** wie gsub(regAusdruck, ersetzedurch, ziel), length(s1), match(string, Ausdruck)

- **function Funktionsname (Parameterliste) { Anweisungen; return R }**

9. Shell

- ist Verbindungsstück zwischen User und Betriebssystem
    1. liest Kommandozeile
    2. zerlegt sie in Token
    3. History aktualisieren
    4. auflösen der Quotes
    5. ersetzen der Aliase und Funktionen
    6. Organisation von E/A-Umleitungen, Hintergrundprozessen und Pipes
    7. ersetzen der Variablen durch Inhalt
    8. Substitution von Kommandos durch Ergebnisse
    9. Substitution von Dateinamen
    10. Ausführen des Programms

- Begriffe:
    - Shellbefehl: built-in-Befehl der Shell
    - Metazeichen: Zeichen, das von der Shell eine besondere Bedeutung zugewiesen bekommen hat

SH - Bourne Shell

- Urvater der Shells
- sh -Optionen Shellscript
- sh -Optionen -c Kommandos
- True = 0, False = irgendwas
- **Metazeichen:** |, *, ?, [...], \, ...
    - `&` führt Kommando im Hintergrund aus
    - `k=`cmd`` k wird die Standardausgabe von cmd zugewiesen
    - `( )` Subshell
    - `{ }` neuer Block
    - `|` Pipe, Standardausgabe des 1. auf Standardeingabe des 2. Kommandos
    - `&&` nachfolgende Kommando wird ausgeführt, wenn 1. 0 (true) liefert
    - `||` nachfolgende Kommando wird ausgeführt, wenn 1. !0 (false) liefert

- **Shellvariable:**
    - Zuweisung: Bezeichner = Wert
      read Bezeichner (liest von STDIN)
    - Abfrage: ${Bezeichner}
    - löschen: unset Bezeichner
    - exportieren: export Bezeichner
    - Achtung bei Zusammensetzung! $XXwort != ${XX}wort

- **vordefinierte Variablen:**

$-	Aufrufparameter der Bourne
$?	Returnwert des letzten Kommandos
$$	PID des aktuellen Prozesses
$!	PID des zuletzt asynchron gestarteten Kommandos (&)
$#	Anzahl der Argumente
$*	Argumente als eine Zeichenkette
$@	Argumente als Folge von $# Zeichenketten
$0	Name der Prozedur

- Spezielle Operationen:

  ${Variable:-SONST}
  - Variable != „“ => Variable, sonst SONST

  ${Variable-SONST}
  - Variable nicht definiert, dann SONST
  - Variable kann aber leer sein

  ${Variable:=SONST}
  - Variable nicht definiert oder = „“
  => Variable = SONST und Rückgabe, sonst Variable als Rückgabe

  ${Variable=SONST}
  - Variable nicht definiert => Variable = SONST und Rückgabe, sonst Variable als Rückgabe

  ${Variable:?SONST}
  - Variable !=““ => Variable, sonst SONST (wenn nicht leer) und Scriptabbruch

  ${Variable?SONST}
  - Variable nicht definiert, Scriptabbruch und Ausgabe SONST (wenn nicht leer)

  ${Variable:+SONST}
  - Variable !=““ => SONST, sonst 0

  ${Variable+SONST}
  - Variable definiert => SONST, sonst 0

- **Expandieren von Dateinamen**
  * beliebige Zeichen
  ? ein beliebiges Zeichen
  . und /. und / müssen explizit angegeben werden!

- **Ein- und Ausgabe**
  - Standardeingabe: Kanal 0
  - Standardausgabe: Kanal 1
  - Standardfehlerausgabe: Kanal 2

  - wenn fd nicht angegeben, dann Standardkanal:
  fd> file Ausgabekanal fd in file umlenken

```
fd<file         umlenken Eingabekanal fd von file
fd<&-           schließt Eingabekanal fd
fd>&fd0         Umlenken der Ausgabe von fd auf Kanal fd0
fd>>file        Umlenken von fd nach file mit Anfügen
<<ENDE          lesen aus Script bis ENDE
```

- **Script-Parameter:**
  - $1...$9
  - mit shift nach links verschieben, um auf weitere Parameter zugreifen zu können

- **Kommandos:**

```
if Kommandos                    (wenn Rückgabe = 0 dann then,
     then Kommandos             sonst bei elif/else weiter)
     { elif Kommandos }
     [ else Kommandos ]
fi

case WORT in
     Muster ) Kommandos ;;      (wenn WORT mit Muster matcht,
                                Kommandos)
     { Muster ) Kommandos ;; }
esac

while Kommandos                 (wenn Rückgabe = 0 dann zu do
do                              und von vorn, break zum
     Kommandos                  Verlassen)
done

until Kommandos                 (wenn Rückgabe != 0 dann do und
do                              von vorn, break)
     Kommandos
done

for Laufvar [in wort {wort}]    (Laufvar = alle Werte aus wort,
do                              jedesmal do, fehlt in,
     Kommandos                  Parameterliste des Scripts,
done                            break)
```

- **interne Shellkommandos:**

:...;	Kommentar, dem nach ; ein weiteres Kommando folgt
.file	liest Kommandos aus file in aktuellen Shell
eval {Argumente}	Argumente werden durch Shell interpretiert
exec {Argumente}	führt Argumente im Shell-Prozess aus und beendet diese dann ($?=Abfrage)
export {Argumente}	exportiert Argumente an Umgebung

getopts opstring name {Argument}
Aufrufargumente des Scripts (nicht Parameter)
opstring: m:n:xy
m + n mit einem Parameter
x + y ohne
name = Shellvariable mit den Optionen

type Kommando welches Kommando wird bei Kommando wird bei Kommando ausgeführt

trap Kommando # Signalbehandlung von # durch Kommando

umask 000 welche bits werden nicht gesetzt

wait PID wartet auf das Ende einer Subshell

Funktion (){Kommandos}
Funktionen immer ohne Parameter

CSH – C-Shell

- csh -Optionen Shellscript | -c Kommando
- Metazeichen:
  - wie bei Sh, nur ohne { }
  - neu:
    ~ Homedirectory
    ! ^ Historysubstitution
    : für History Substitution

- **Kommandos:**
  ; Kommandos nacheinander
  ( ) Kommandos in Subshell (nicht in Scripten!)

- **Variablen:**
  - set VAR = Wert
  - setenv VAR = Wert (Umgebungsvariable)
  - set VAR=(Wert1 Wert2...) (Feld)
  - Zugriff über $VAR oder $VAR[Index] (Index = * => alle), auch Bereiche möglich 2-9
  - löschen mit unset oder unsetenv
  - $?VAR liefert 1 = wenn gesetzt, 0 = nicht gesetzt
  - $#VAR = Anzahl der Einträge

- **vordefinierte Shellvariablen:**
  - wie bei sh
  $cwd = pwd

- **Quoting:**
  - wie bei sh, aber bei „...“ verliert \ normales Zeichen!

- **Ein/Ausgabe:**
  - nur Standardkanäle umleitbar

- sonst wie bei sh
- mit ! wird noclobber (Schreibschutz) ignoriert
$< liest Variablenwert von Standardeingabe
>& >>& Standardausgabe in file oder anhängen

- **Ausdrücke:**
  - werden wie in C berechnet (False = 0, True = irgendwas)

- **Kommandos:**
  - hier zählt der Wert des gesamten Ausdrucks und nicht nur der wert des letzten Kommandos (wie zum Beispiel Test)
  - es sind normale Ausdrücke erlaubt

```
if ( Ausdruck ) then
     else
endif

switch (Wort)
     case Muster :
          breaksw (benötigt zum Stoppen der Abarbeitung)
     default :
endsw


while ( Ausdruck )
end

repeat n cmd (n mal cmd wiederholen)

foreach laufvar in (wort wort) (auch felder möglich!)
end

goto marke
```

10. Configure und Autoconfig

- autoconf und automake sind Tools zur Erstellung von Konfigurationsscripten für Unix-Systeme
- Anpassung von Pfaden für Programme, Bibliotheken etc.
- durch configure wird wird Software an konkrete System angepasst

- **benötigte Dateien:**

Quelltexte	
configure.in	(für autoconf)
Makefile.am	(für automake)
bindende Dokumentation	(Authors, Copying...)
Scripte für Anwender	(mkinstalldirs, install.sh...)

- **Arbeitsschritte:**
  1. erstelle configure.in
     - Benutzung von autoscan (untersucht Header, System- und Bibliotheksaufrufe, ermittelt Portabilitätsprobleme) =>

configure.scan, als Basis nutzen
2. erzeuge aclocal.m4
   - benutze aclocal
   - wichtig für Benutzung von m4-Dateien
3. erzeuge configure
   - autoconf + configure.in => configure, config.status
4. erzeuge Makefile.in
   - automake + Makefile.am => Makfile.in (wird auf Zielsystem von configure für Erzeugung von Makefile benötigt)
5. fertig, Anwender kann jetzt ./configure und danach make rufen

ps

- print process status

top

expr

- berechnet arithmetische und logische Ausdrücke

xargs

- erzeugt eine Liste, die einem Programm übergeben werden kann
- find . -name "test*" | xargs grep -h -E "[a]+"
  - gibt alle Zeilen aus, die im aktuellen Verzeichnis in Files test* mindestens ein a enthalten (Angabe des Files wird durch -h unterdrückt)
- durch -i wird {} als Parameter aktiviert

du (disk usage)

df (disk free)

tar (+komprimieren)

- packen: tar -cvzf Ziel.tar Quelle
- entpacken: tar -xvzf Tar-file

chmods

- Änderung der Dateiattribute
- r (read), w (write), x (execute)

0 : dem Benutzer ist keine Operation mit dem Objekt gestattet
1 : der Benutzer darf die Datei ausführen bzw. in das Verzeichnis wechseln
2 : der Benutzer darf in die Datei schreiben bzw. Dateien und Unterverzeichnisse im Verzeichnis erstellen und löschen
4 : der Benutzer darf aus der Datei lesen bzw. sich die Dateien im Verzeichnis anzeigen lassen

- chmod ugo   (u = user/owner, g = group, o = other)
- Addition der Zahlen (s.o.)
- Bsp.:   chmod x764
=> u: 1, 2, 4
=> g: 2, 4
=> o: 4
=> x: 4(user), 2(group), 1(other)

- mit -R rekursiv für alle Unterordner
- sticky files: Datei wird im swap gehalten (von Linux ignoriert)
- sticky directories: seine Dateien erstellen, löschen und ändern, aber auch andere lesen (Bsp.: /tmp)
- chmod [u,g,o]+[r,w,x]

ls

-Optionen:
    -l 10bits, #Hardlinks, owner, group, Bytes, Änderung, Name
    -1.bit: d = Verzeichnis
              l = symbolischer Link
              - = normale Datei
              p = named pipe
    -9bit Zugriffsrechte

Test

- überprüft Dateien und Ausdrücke
- []
- Optionen:
    bei Strings
    !   not
    -a   and
    -o   or

    bei Zahlen:
    -eq =
    -ge >=
    -gt >
    -ne !=

    bei Dateien:
    -nt neuer als
    -e   existiert
    -d   existiert und ist Directory
    -f   existiert und ist File

Quoting

\   maskiert nur nächstes Zeichen
'...'   maskiert alle eingeschlossenen Zeichen, außer '
„...“   maskiert alle eingeschlossenen Zeichen, außer $ \ `

IPCS

- Informationen über Interprozesskommunikation