8.4-12.4 9:00 - 12:00
14:00 - 17:00
INF 226
Raum 1.204

Linux als Arbeitsplattform im Wissenschaftlichen Umfeld

Jörg Marks, Physikalisches Institut, INF 226
marks@physi.uni-heidelberg.de

Programm Überblick

- Linux Einführung und Arbeitsumgebung
- Installation und Überblick über wichtige/nützliche Programme
- Mikroeinführung in die Programmiersprache C/C++
- Installation und Funktion des Analysewerkzeugs ROOT
- Mikroeinführung in die Verwendung von GPUs
- Virtualisierung

Organisatorisches

- 2 Leistungspunkte:
  - Anwesenheitspflicht mit Übungen im Tutorialmodus
- Kurs web page

  http://www.physi.uni-heidelberg.de/~marks/linux_einfuehrung

Installation von OpenSuSE

Als zu installierende Linux Distribution wollen wir OpenSuSE verwenden. Diese Distribution existiert seit 1992. Neben der öffentlichen, freien Version gibt es eine Enterprise Version, die verkauft wird und für die auch Support gekauft werden kann.
Die öffentliche Version ist lediglich community supported.

- home page der freien Version
  https://www.opensuse.org/
- Software download der stabilen freien Version 15.0 (leap)
  https://www.opensuse.org/#Leap
- Installationsnotizen für unseren Kurs
  http://www.physi.uni-heidelberg.de/~marks/linux_einfuehrung/Folien/SuSE_InstallationsNotizen.pdf

Linux Werkzeuge in KDE

Das Interface zu den Linux Programmen ist die Benutzeroberfläche. OpenSuse besitzt eine sehr gut integrierte KDE Oberfläche.

- Konfiguration der Taskleiste
  - Durch einen Rechtsklick mit der Maus auf die Taskleiste kann `Panel Settings` selektiert werden. Dies erlaubt eine Konfiguration des Panels.
- Gruppierung der Oberfläche in verschiedene Desktopbereiche
  - Durch einen Rechtsklick mit der Maus auf die Desktops kann `Configure Desktop` ausgewählt werden und u.a. die Anzahl verändert werden.
- Zugang zu allen Programmen über den SuseButton links in der Taskleiste
  - Die Programme sind unter zusammengehörigen Gruppen zu finden.
  - Eine Programmsuchfunktion kann ebenfalls benutzt werden.
  - Wichtige Programme können in der Taskleiste verlinkt werden. Mit einem Rechtsklick auf die Anwendung kann `Add to Panel` selektiert werden.
- Der Filemanager Dolphin erlaubt eine windowsartige Nutzung
  `Suse Button → System → Dolphin`

Nützliche Anwendungen in KDE

- Welche Programme finden weit verbreitete Anwendung in der Linux Welt

- `firefox`	web browser
`google-chrome`	
- `thunderbird`	Mail client
- `dolphin`	File manager
- `k3b`	Brennen von DVDs
- `okular`	PDF viewer, Acrobat ist nicht unterstützt
- `gv`	Postscript Viewer
- `gwenview`	jpeg, gif … File Viewer
- `gimp`	Bildbearbeitungssoftware
- `digikam`	Photoverwaltungssoftware
- `amarok`	Musik Player
- `kaffeine`	Video Player spielt lizensierte Files mit libdvdcss
`mplayer, vlc`	
- `ssh`	Secure shell client und server Programm zum Verbinden mit anderen Rechnern
- `python, python2(3)`	python Interpreter
- `gcc`	Compiler Suite
- `libreoffice(soffice)`	Open Office Anwendungen, analog zu den Microsoft Produkten
- `latex`	Textsatz Programm

Arbeitsvorschläge:

- Loggen Sie sich mit dem userid `student` ein und konfigurieren Sie die Taskleiste.
  i) Vergrößern Sie die Taskleiste
  ii) Fügen Sie buttons für `Konsole`, `Firefox` und `Terminal Super User` ein.
  iii) Installieren Sie die Terminalanwendung `Terminator`. Warum ist diese Anwendung nützlich?
- Vergrößern Sie die Anzahl der Desktops
- Rufen Sie die home page der Veranstaltung auf.
- Öffnen Sie den Filemanager und browsen Sie durch Ihr home directory.

Netzwerkverbindungen

Während der Installation wird die Netzwerkkonfiguration automatisch vorgenommen, wenn eine Netzwerkverbindung vorliegt. Netzwerkkonfigurationen können aber auch über den KDE network manager hinzugefügt werden. Er befindet rechts unten im KDE panel. Zur Verfügung stehende WLAN Netzwerke werden angezeigt und können durch Eingabe der Sicherheitsparameter aktiviert werden.

- Konfigurieren von eduroam

  Der WLAN-Dienst "eduroam" (education roaming) ist weltweit an vielen Universitäten und Forschungseinrichtungen verfügbar und bietet einen verschlüsselten Internetzugang ohne dass vorher an der anderen Einrichtung ein Account beantragt werden muss. https://www.urz.uni-heidelberg.de/de/eduroam-unter-linux-einrichten

- Systemweite Netzwerk Konfiguration

  `Suse Button → System → Yast` als user

  `Yast → System → Network Settings` öffnet die Netzwerkkonfiguration

  Abschalten des Network managers

  `Global Options → Network Setup Method → wicked service`

  `Change Default route via dhcp` selektieren

  `Overview` Konfiguration der Netzwerkadapter

  `add oder edit → Dynamic Address` Adresse wird automatisch bezogen

Arbeitsvorschlag:

- Konfigurieren Sie im network manager von KDE das WLAN Netz eduroam

Shells als Zugang zu Linux

Eine shell stellt den wichstigsten Zugang zu Linux dar. Über shell commands schicken wir Befehle an das Linuxsystem oder erhalten Antworten.

- Terminal Programme
  - `Suse Button → System → Konsole`
  - `Suse Button → System → Terminal Super User Mode`
  - `Suse Button → System → Terminator`
- Editieren von Dateien
  - `emacs <filename>` Erklärungen im Programm
  - `vi <filename>`

`i / a`	in den Edit Mode / hinter einem Wort editieren
`cw`	curser auf ein Wort positionieren und ein Wort ändern
`yy`	Kopieren der Zeile
`dd / ndd`	Löschen der Zeile / Löschen von n Zeilen
`p`	paste
`Esc Taste`	verlassen des edit modes
`:q!`	quit ohne zu sichern
`:wq!`	write and quit

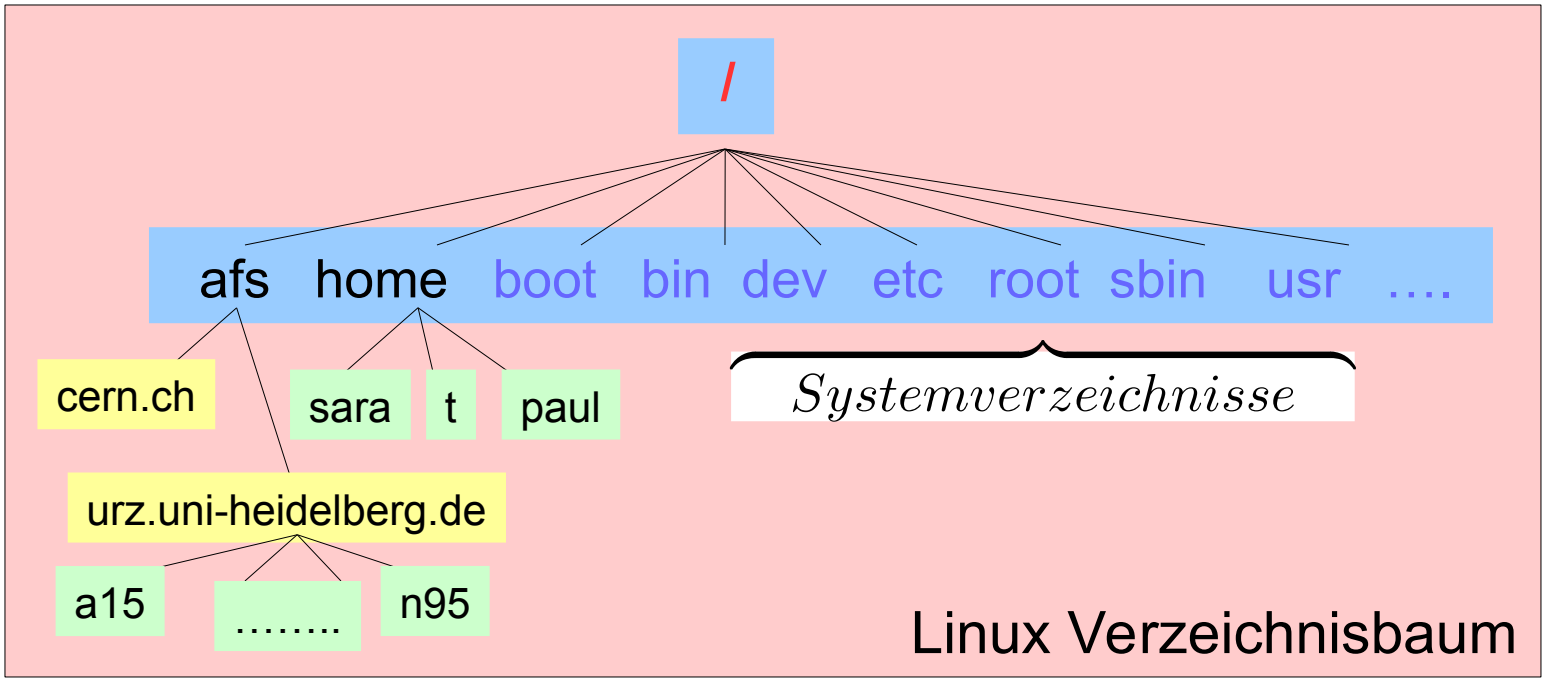
vi summary: http://www.linfo.org/vi/summary.html

Arbeitsvorschläge:

- Öffnen Sie eine shell. Probieren Sie beide Editoren `emacs` und `vi`.

```
vi myTextFile.txt
emacs myTextFile.txt
```

Linux Verzeichnisstruktur



- Interaktion mit Userprogrammen der Linux Distributionen erfolgt über eine graphische Oberfläche oder über die shell (Eingabe Fenster).
- Für unseren Kurs brauchen wir einige shell Komandos.

  Zusammenfassung einiger shell Kommandos mit link zu einem guten Online Tutorial:
  http://www.physi.uni-heidelberg.de/~marks/root_einfuehrung/Folien/UnixEinfuehrung.pdf
  Mikroübersicht zum Arbeiten im CIP Pool:
  http://www.physi.uni-heidelberg.de/~marks/root_einfuehrung/Folien/cipEinfuehrung.pdf

Nützliche Linux Komandos

- Hilfe unter Linux

`man <cmd>`	Beschreibung zu Komandos anzeigen
`info`	Linux Hilfe System anzeigen

- Verzeichnisse und Files

`drwxr-xr-x` `u g o`	Rechte <dir> or <file> : d = directory / r = read / w = write / x = execute u = user / g = group / o = other
`.`	Aktuelles Directory
`..`	Directory oberhalb des aktuellen Directories
`chmod u+x/u+r/u+w <file>`	Beispiele: Ändern der Filerechte (execute/read/write) für user, analog für other oder group

- Verzeichnisbaum

`ls -l -a -t <dir>`	Listen aller Files im Directory <dir> a für all auch . Files, t für zeitsortiert

```
drwxr-xr-x 2 marks users     4096 Oct 28 13:35 code
-rw-r--r-- 1 marks users 3025217 Oct 31 13:20 einleitung.odp
```

`cd <dir>`	in das Directory <dir> wechseln
`mkdir <dir>`	Erzeugen des Directories <dir>
`pwd`	Verzeichnisbaum des aktuellen Directories

Nützliche Linux Komandos

- Files

`cat <file>`	Fileinhalt auf der shell ausgeben
`echo "bla bla" > <file>`	String in <file> kopieren
`echo "more .." >> <file>`	String an <file> anhängen
`mv <fileA> <fileB>`	File oder Directory von <fileA> nach <fileB> umbenennen
`less <file>`	Fileinhalt ansehen mit Blättern
`vi <file>`	Editieren und erzeugen eines Files mit vi
`.myConfigFile`	Files und Directories mit Punkt im Namensbeginn sind nicht sichtbar
`ls -a <dir>`	Zeigt auch Files mit Punkt im Namensbeginn
`chmod a+x <file>`	Files für alle user ausführbar machen
`ls -l <dir>`	Zeigt Files mit Rechten, Zeit, user and group

- Suchen im Verzeichnisbaum oder im File

`find <dir> -name <filename>`	Suchen eines Files vom Directory <dir>
`grep pattern <file>`	Ausdruck pattern in <file> finden
`locate pattern`	Files mit string pattern im Namen finden

`locate` benötigt eine Datenbank mit allen File Namen, die durchsucht wird. Das Programm `updatedb` erzeugt die DB und muss als root user gestartet werden.

Nützliche Linux Komandos

- Löschen von Files und Directories

  Vorsicht mit dem Löschen: Files und Directory werden instantan entfernt und sind weg oder nur aus dem backup zurück zu holen.

`rm <file>`	Löschen des Files <file> im aktuellen directory
`rm –rf <dir>`	Rekursives löschen aller Files und der darunter-liegenden Directory Struktur von <dir> (vorsicht)
`rm –rf *`	Rekursives löschen aller Files und der darunter-liegenden Directory Struktur vom aktuellen directory.
`rmdir <dir>`	Löscht das leere Directory <dir>

- Verwendung von Pipes

  Das Pipe Zeichen | unter Linux bedeutet, das der Output vom links stehenden Command als Input für das rechts stehende Command benutzt wird.

`commandA | commandB`	Pipe
`cat File.txt | grep –i "Pet"`	Sucht nach pattern Pet in File.txt

Nützliche Linux Komandos

- Weitere Commands
  - ~ ist ein Synonym für das home directory des eingelogten Benutzers (cd ~/myDir).
  - * wirkt als Wildcard und kann in File und Directory Namen eingesetzt werden.
  - ; Mehrere commands können in einer Zeile hintereinander ausgeführt werden, wenn Sie durch ; getrennt sind.
  - top zeigt eine Liste mit allen Prozessen, die im System laufen.
  - history erzeugt eine Liste der eingebenen letzten 1000 Befehle.
  - Mit den Pfeil Up/Down Tasten lässt sich in den bereits eingegebenen Komandos blättern.
  - Mit /pattern kann in `vi` und `less` nach dem string pattern gesucht werden.
  - !g führt das letzte mit g beginnende Komando aus.
  - Tabulator Taste ergänzt eine begonnene Eingabe
  - Strg + c bricht ein Komando ab, das gerade ausgeführt wird.
  - Komando & das Komando wird im Hintergrund ausgeführt.
  - Strg + z bg <return> ein Komando, das gerade ausgeführt wird, wird in den Hintergrund schickt.
  - ps — zeigt Prozesse, die in der shell ausgeführt werden
  - ps -aux — zeigt Prozessliste
  - wc -l <file> — zählt die Anzahl der lines in einem File <file>
  - kill -9 <process id> — entfernen des Prozesses mit der id <process id>

Arbeitsvorschläge:

- Erzeugen Sie ein Directory “LinuxEinfuehrung“. Wechseln Sie in das Directory und erzeugen Sie “folien“. Jetzt laden Sie LinuxCommands.pdf, einleitung.pdf und InstallationsNotizen.txt von unserer Kurswebpage und speichern die Files in “folien“. Wo befinden sich die heruntergeladenen Files zunächst?

  Suchen Sie in den InstallationsNotizen.txt nach dem string Software ?

  Gibt es eine Option für grep mit der Groß- und Kleinschreibung ignoriert wird?

- Probieren Sie beide Editoren `emacs` und `vi`. Schicken Sie `emacs myFile.txt` in den Hintergrund.

- `lsmod` listet die geladenen Kernelmodule. Bestimmen Sie die Anzahl der aktive Kernel module.

- Öffnen Sie das File InstallationsNotizen.txt mit `less`. Verwenden Sie `ps` und `top` um die process number zu finden. Suchen Sie nach der process number und beenden Sie den Prozess.

- Das Komando `uname -r` zeigt Ihnen die Kernel Version, die Sie verwenden. In dem directory /lib/modules/..kernel../ sind alle Kernelmodule in weiteren Directories zu finden. Die Module haben die Fileendung *.ko . Wie können wir die Anzahl aller Module herausfinden?

Linux Shell Scripts - Einleitung

- Einleitung

  Beim Arbeiten am und mit einem Linux System treten wiederholt gleiche und komplizierte Befehlsketten auf. Shell Skripte sind ein Weg Administrations- und Analyseprozesse zu automatisieren.

  - Mögliche Skript Sprachen: bash, csh, perl, python, ……
    (auch in Kombination mit kompilierten C/C++ Prozessen)
  - Shell Script ist ein ausführbares Textfile in dem in der ersten Zeile mit der Sequenz #!Pfad/zum/Interpreter: `#!/bin/bash` (shebang)
    die Sprache festgelegt wird.
  - Alle Zeichen im Skript, die nach einem # stehen werden ignoriert.
  - Ausführen von einem Skript

```
chmod u+x myScript.sh
./myScript.sh
```

- Ein Beispiel

```
#!/bin/bash
# ein kleiner test
echo Hello world!
echo "so stellen wir" \" Sonderzeichen dar "*" \$Maehh
# Ausdruck hinter \ wird woertlich genommen
exit 0
```

Linux Shell Scripts - Variable

- Variable und Zuweisungen
  - Synthax
    `variable=MyValue`
  - Keine Blanks links und rechts des = Zeichens in der Zuweisung!
  - bash ist case sensitive
  - Mit einem `$` direkt vor der Variablen wird der Inhalt zurückgeholt.
    `myNewVariable=$variable`
    `myNewVariable` enthält jetzt `MyValue`
  - Variablen Namen sollten taskbeschreibenend sein!
- Environment Variable im Shell Script

Einige vordefinierte Variable:

`$0`	Name des eigenen Skripts
`$1 – $9`	Enthält Parameterwerte die dem Skript mitgegeben werden
`$#`	Anzahl der Argumente die dem Skript mit gegeben werden
`$@`	Alle Argumente des Skripts
`$?`	Exit status des gerade ausgeführten Prozesses
`$$`	Prozess-id des ausgeführten Skripts
`$USER`	User-id des Users der das Skript ausführt
`$HOME`	home directory des Users der das Skript ausführt
`$HOSTNAME`	Name des Rechners der das Skript ausführt

Linux Shell Scripts - Variable

- Zuweisungen
  - Fehler durch diese Zuweisung
    ```
    variable=Hi there
    ```
  - Ok
    ```
    variable="Hi there from $0"
    variable='Hi there'
    echo $variable
    ```
  - Command substitution
    `$( command )` führt im Skript command aus und gibt den Output zurück
    dies erlaubt auch eine Zuweisung
    `LocalDir=$(ls)` weist `LocalDir` den output von `ls` zu
  - `unset myVariable` löscht den Inhalt der Variablen

- Lebensdauer von Variablen
  - Die Lebensdauer der Variablen ist auf den Skript Prozess begrenzt, d.h die Variablen leben nur in der Umgebung des Skripts. Mit dem Kommando `export myGlobalVariable=foo` wird die Variable auch für Tochterprozesse zugänglich. Beachten Sie das im Tochterprozess `myGlobalVariable` nicht modifiziert und dann im rufenden Prozess verwendet werden kann.

Linux Shell Scripts - Variable

- Beispiel Lebensdauer von Variablen

Script `parent.sh`

```
#!/bin/bash
# set variables in parent.sh
varA=bar
varB=foo
# verify their current value
echo $0 :: varA : $varA, varB : $varB
export varA
./daughter.sh
# Are they modified ?
echo $0 :: varA : $varA, varB : $varB
```

Script `daughter.sh`

```
#!/bin/bash
# set variables in daughter.sh
echo $0 :: varA : $varA, varB : $varB
# change their values
varA=flip
varB=flop
echo $0 :: varA : $varA, varB : $varB
```

Arbeitsvorschläge:

- Schreiben Sie ein Shell Skript, das das Setzen und die Ausgabe von Variablen testet. Verwenden Sie dabei auch doppelte und einfache Anführungszeichen. Verwenden Sie auch Sonderzeichen.

- Schreiben Sie ein Shell Skript, das die Anzahl der Files und Directories für einen Pfad bestimmt, der mit dem Skript übergeben wird.

- Testen Sie die Lebensdauer von Variablen mit einen eigenen Skript

Linux Shell Scripts – Zahlen

- Integer Berechnungen in der Shell
  - Synthax

    ```
    $(( expression ))
    ```

  - Beispiele (Input und Output auf der Komandozeile)

    ```
    mymachine> echo $((100 / 3))
    mymachine> 33
    mymachine> x="12"
    mymachine> y="2"
    mymachine> z=$(($x + $y)) ; echo $z
    mymachine> 14
    mymachine> echo ((x++))
    mymachine> 15
    mymachine> echo $(((5+2)*2))
    mymachine> 14
    mymachine> $((10 ** 18))
    mymachine> 1000000000000000000
    mymachine> $((10 ** 19))
    mymachine> -8446744073709551616
    ```

    32 bit limit reached

Im Shell Skript lassen sich nur Integer Ausdrücke prozessieren. Für komplexere Berechnungen muss das Komando `bc` verwendet werden.

Linux Shell Scripts – Verzweigungen

- Verzweigungen
  - Synthax

```
if  [ BedingungA ] ; then
      VariableA=MyValue
else
      VariableA=YourValue
fi
```

  - `then` kann auch ohne `;` in einer neuen Zeile stehen.
  - Verknüpfungen von mehreren Bedingungen

```
BedingungA && BedingungB
BedingungA || BedingungB
```

  - Bedingungen

`String1 = String2`	Ist wahr, wenn String1 gleich String2 ist.
`Zahl1 -eq Zahl2`	Ist wahr, wenn Zahl1 gleich Zahl2 ist. (-eq = equal)
`Zahl1 -lt Zahl2`	Ist wahr, wenn Zahl1 < Zahl2 ist. (-lt = less than)
`Zahl1 -gt Zahl2`	Ist wahr, wenn Zahl1 > Zahl2 ist. (-gt = greater than)
`Zahl1 -le Zahl2`	Ist wahr, wenn Zahl1 <= Zahl2 ist. (-le = less or equal)
`Zahl1 -ge Zahl2`	Ist wahr, wenn Zahl1 >= Zahl2 ist. (-ge = greater or eq)
`Zahl1 -ne Zahl2`	Ist wahr, wenn Zahl1 nicht gleich Zahl2 ist. (-ne = not eq)
`!foo`	Ist wahr, wenn foo falsch ist.

Linux Shell Scripts – Eigenschaften

Beim Arbeiten mit Shell Skripten müssen häufig Eigenschaften getestet werden. Unten ist eine unvollständige Liste der Optionen von `test` angegeben,

- test command
  - Synthax

```
test <EXPRESSION>
[ <EXPRESSION> ]
```

  - <EXPRESSION>

`-d <file>`	Ist wahr, wenn <file> ein directory ist
`-e <file>`	Ist wahr, wenn <file> existiert
`-L <file>`	Ist wahr, wenn <file> existiert und ein link ist
`-w <file>`	Ist wahr, wenn <file> existiert und schreibbar ist
`-r <file>`	Ist wahr, wenn <file> existiert und lesbar ist
`-x <file>`	Ist wahr, wenn <file> existiert und ausführbar ist
`-s <file>`	Ist wahr, wenn <file> existiert und file Größe > 0 ist
`<file1> -nt <file2>`	Ist wahr, wenn <file1> neuer als <file2> ist
`<file1> -ot <file2>`	Ist wahr, wenn <file1> älter als <file2> ist
`-z <string>`	Ist wahr, wenn <string> leer ist
`-n <string>`	Ist wahr, wenn <string> nicht leer ist
`-v <VARIABLENAME>`	Ist wahr, wenn <VARIABLENAME> gesetzt ist

Linux Shell Scripts – Arrays

Feldvariablen werden als "Arrays" bezeichnet. Es werden mehrere Werte innerhalb einer Variablen abgespeichert. Die einzelnen Werte können über den Index des Feldes einzeln angesprochen werden.

- Arrays
  - Synthax

```
myArray[index]=value
declare -a myArray=(element1 element2 …. elementN)
myArray=( element1 element2 ….. elementN)
```

  - Zugriff auf Arrays

`${myArray[index]}`	Zugriff auf Array Elemente
`${myArray[@]}`	Listet alle Array Elemente
`${myArray[*]}`	Listet alle Array Elemente
`${#myArray[index]}`	Gibt die Länge des Wortes `#myArray[index]` an
`${#myArray[*]}`	Gibt die Länge des Arrays an
`${!myArray[*]}`	Gibt alle Indices des Arrays an

  - Beispiel

```
array=(one two three four [5]=five)
for index in ${!array[*]}
do
   echo $index ${array[$index]}
done
```

Linux Shell Scripts – Schleifen

Eine wichtige Eigenschaft ist das wiederholte Ausführen von Kommando Sequenzen
Dazu verstehen verschieden Schleifen Konstrukte zur Verfügung.

- for Schleife
  - Synthax

```
for var in ListOfParameters
 do
  Komando1
  …..
  KomandoN
 done
```

Oder

```
for (( var=Anfangswert ; Bedingung ; Zähler ))
 do
  kommando1
  …..
  kommanodN
done
```

Linux Shell Scripts – Schleifen

- for Schleife Beispiele

```
# Gibt alle Textdateien des aktuellen Verzeichnisses aus
 for datei in *.txt
  do
   echo $datei
   [ -r $datei ] && echo "... ist lesbar"
   [ -w $datei ] && echo "... ist schreibbar"
  done
```

oder

```
# Schleife von 10 bis 1
 for (( i=10 ; i>0 ; i--))
  do
   echo $(($i*1000))
 done
```

Linux Shell Scripts – Schleifen

Eine wichtige Eigenschaft ist das wiederholte Ausführen von Kommando Sequenzen. Dazu verstehen verschieden Schleifen Konstrukte zur Verfügung.

- while Schleife
  - Synthax

```
while [Bedingung]
do
  Kommando_1 ; Kommando_n
done
```

  - Beispiel User Eingabe

```
# Usage of a while loop to read console input
while [ "$input" != "ende" ]
do
 echo "Continue with „enter“ key or stop with „ende“ "
 read input
done
echo "End reached"
```

  - Synthax `until` Schleife

```
until [ "$input" = "ende" ]
do
  Kommando_1 ; Kommando_n
done
```

Arbeitsvorschläge:

- Schreiben Sie ein Shell Skript, das für ein gegebenes Directory alle Textfiles (Endung .txt) auf Lesbarkeit und Schreibbarkeit überprüft. Das Directory wird als Argument übergeben. Es soll getestet werden, ob nur ein Argument übergeben wird.

- Schreiben Sie ein Shell Skript, das den Eingabe Input in ein Array schreibt und mit ENTER den neuen Input erwartet oder durch Eingabe von “ende“ das Programm beendet. Die Anzahl der eingegebenen Werte soll gezählt werden. “ende“ soll nicht in die Liste gelangen.

Linux Shell – awk Kommando

awk ist eine Skriptsprache und wurde in den 1970iger Jahren entwickelt (Alfred Aho, Peter Weinberger, Brian Kernighan). Sie sollte komplexe pattern-matching Operationen in Datenströmen vereinfachen. Sie kann als Vorläufer von perl betrachtet werden. Sie wird in Kombination mit Shell Skripten verwendet oder auf der Shell in einer pipe.

1	12	100	1010	2020
2	23	210	2200	4100
3	30	301	3201	6011
4	42	422	4100	8012
5	51	501	5103	10201
6	64	634	6071	12056
7	72	729	7191	14000
8	81	823	8100	16123
9	93	921	9231	18322
10	101	1011	9995	20178

- awk Beispiele
  - awk arbeitet Textfiles zeilenweise ab.
  - Vertauschen der ersten und zweiten Spalte

```
awk '{ print $2, $1 }' myText.txt
```

  - $1-$NF Platzhalter der Spalten
    NF = Anzahl der Zeilenfelder
    NR = Anzahl der Zeilen seit Skriptstart
  - Summieren der Werte der ersten Spalte und Mittelwert bilden

```
awk ' { s += $1 } END {print s/NR} ' myText.txt
```

  - Summieren der Werte der dritten Spalte und Mittelwert bilden für Zeilen > 5

```
awk ' { if ($1 > 5) s += $3 } END {print s/NR} ' myText.txt
```

  - Die Feldseparatoren (FS) können ausgewählt werden, default sind Leerzeichen

```
awk -F : '{ print $1 , $4 }' /etc/passwd
```

Arbeitsvorschläge:

- Summieren Sie die dritte und vierte Spalte des Textfiles myText.txt und fügen Sie das Ergebnis als weitere Spalte zur Tabelle hinzu. Schreiben Sie die Tabelle in ein neues File.

- Lesen aus der Datei `/etc/passwd` den group Id aller user und sortieren Sie die group Id in aufsteigender Reihenfolge. Verwenden Sie zum Sortieren `sort`.

ASCII-Zeichentabelle

0 =	18 = ↕	36 = $	54 = 6	72 = H	90 = Z	108 = l
1 = ☺	19 = ‼	37 = %	55 = 7	73 = I	91 = [	109 = m
2 = ☻	20 = ¶	38 = &	56 = 8	74 = J	92 = \	110 = n
3 = ♥	21 = §	39 = ’	57 = 9	75 = K	93 =]	111 = o
4 = ♦	22 = ▬	40 = (	58 = :	76 = L	94 = ^	112 = p
5 = ♣	23 = ↨	41 =)	59 = ;	77 = M	95 = _	113 = q
6 = ♠	24 = ↑	42 = *	60 = <	78 = N	96 = ‘	114 = r
7 = •	25 = ↓	43 = +	61 = =	79 = O	97 = a	115 = s
8 = ◘	26 = →	44 = ,	62 = >	80 = P	98 = b	116 = t
9 = ◇	27 = ←	45 = -	63 = ?	81 = Q	99 = c	117 = u
10 = ◙	28 = ∟	46 = .	64 = @	82 = R	100 = d	118 = v
11 = ♂	29 = ↔	47 = /	65 = A	83 = S	101 = e	119 = w
12 = ♀	30 = ▲	48 = 0	66 = B	84 = T	102 = f	120 = x
13 = ♪	31 = ▼	49 = 1	67 = C	85 = U	103 = g	121 = y
14 = ♫	32 =	50 = 2	68 = D	86 = V	104 = h	122 = z
15 = ☼	33 = !	51 = 3	69 = E	87 = W	105 = i	123 = {
16 = ►	34 = "	52 = 4	70 = F	88 = X	106 = j	124 = ¦
17 = ◄	35 = #	53 = 5	71 = G	89 = Y	107 = k	125 = }