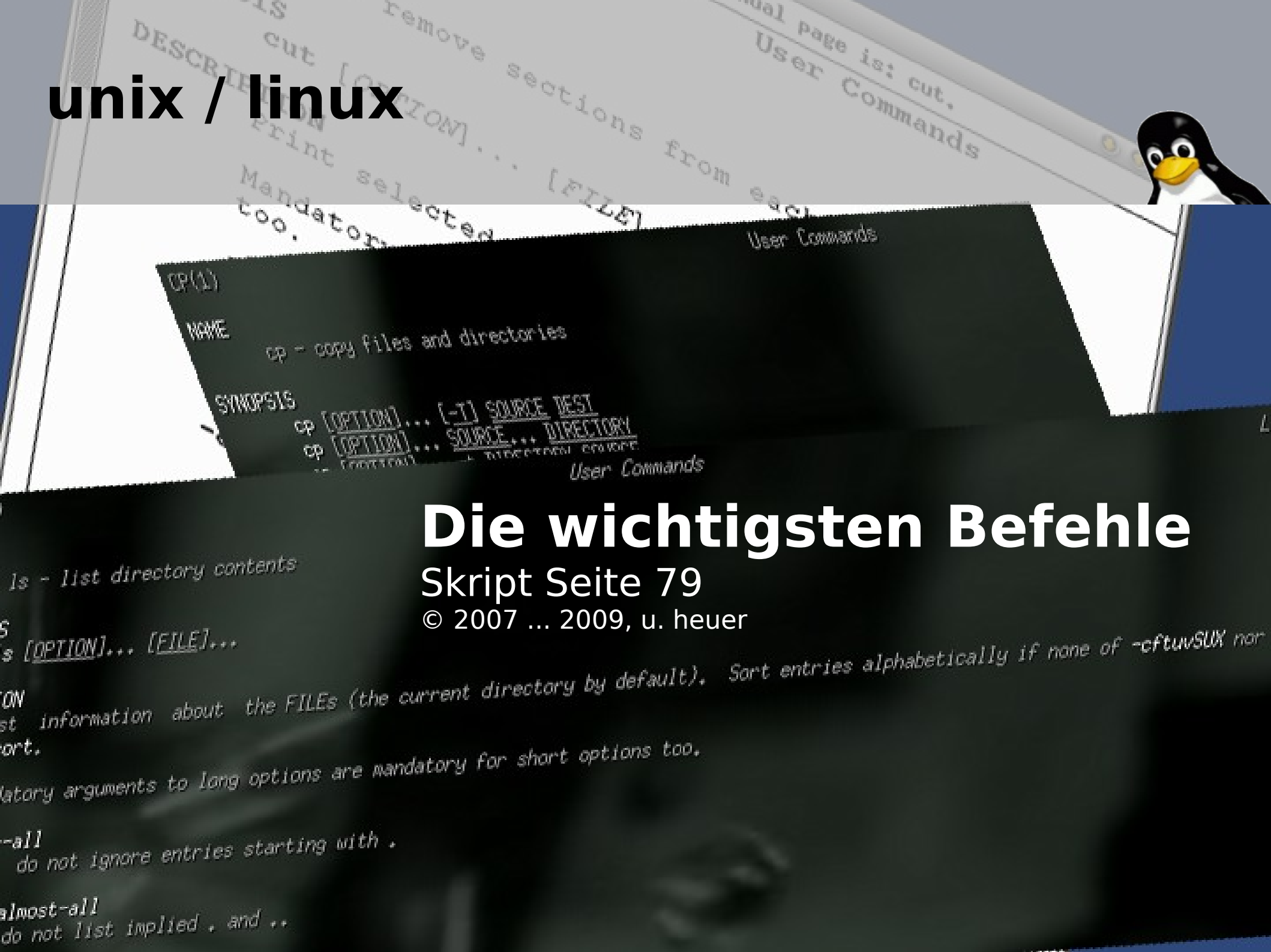
unix / linux



Die wichtigsten Befehle

Skript Seite 79

Die wichtigen Kommandos

Ziele:

- Die wichtigsten Kommandos kennen lernen und anwenden können
- Reguläre Ausdrücke kennen und einsetzen können

$ echo

echo gibt die Argumente aus

```
$ w=Welt
$ echo Hallo $w
Hallo Welt

$ echo Hallo ${w}enbummler
Hallo Weltenbummler
```

Mit der Option **-n** wird am Schluss **kein** <cr><lf> angehängt.

☞ **echo** gibt es als internen wie auch externen Befehl!

$ date

date zeigt - je nach Optionen – die aktuelle Zeit und das aktuelle Datum an

```
$ date
Wed Mar 28 22:46:12 CEST 2007

$ TZ=Europe/London date
Wed Mar 28 21:47:59 BST 2007
# TZ=... definiert eine Environments Variable, die nur für den nachfolgenden Befehl
# (hier date) gültig ist.

$ year="$(date +%Y)"
$ echo $year
2007
# die genauen Format Definitionen sind in der man- oder Info-Page von date
# dokumentiert
```

$ date

date kann auch die Systemzeit setzen. Die Zeit muss im Format `MMDDhhmm[[CC]YY.ss]` angegeben werden. (M month, D day, h hour, m minutes, C century, y year, s seconds).
Logischerweise kann das nur der Benutzer **root** erledigen!

```
$ date 032823202007.51
Wed Mar 28 23:20:51 CEST 2007
```

☞ **date** setzt nur die UNIX interne Uhr!

Übungen

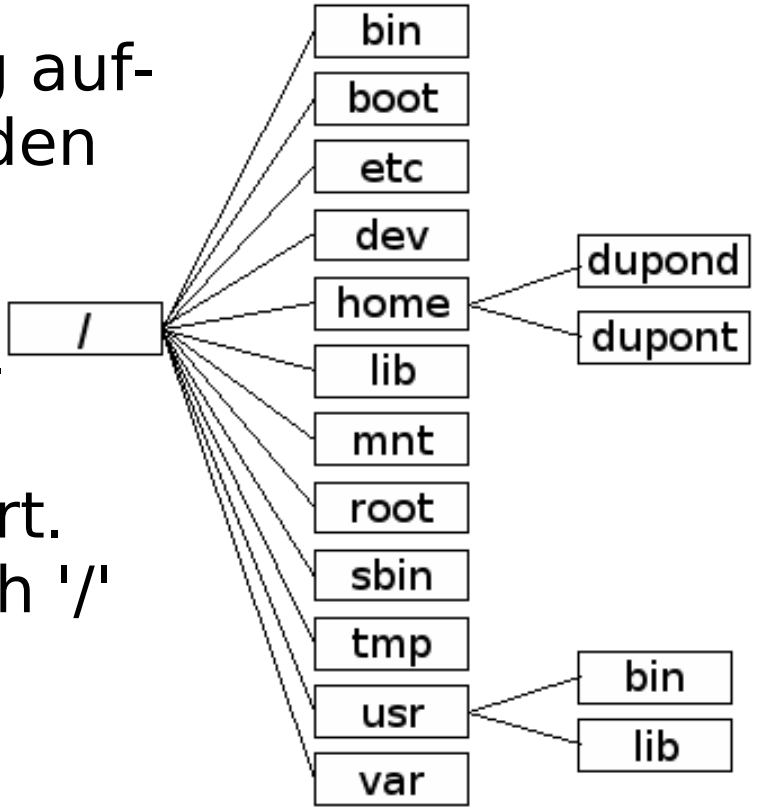
5.1 Nehmen wir an, jetzt ist der 22. Oktober 2003, 12:34 Uhr und 56 Sekunden. Studieren Sie die Dokumentation von `date` und geben Sie die Formatierungsanweisungen an, mit denen Sie die folgenden Ausgaben erreichen können:

1. 22-10-2003
2. 03-294 (KW43) (Zweistellige Jahreszahl, fortlaufende Nummer des Tags im Jahr, Kalenderwoche)
3. 12h34m56s

5.2 Wie spät ist es gerade in Los Angeles?

Pfade

Pfade geben an wie man sich durch die Verzeichnisse des UNIX Dateisystem wandern kann.
Das Dateisystem ist baumförmig aufgebaut. Der Pfad setzt sich aus den durchwanderten Directories zusammen.
Will man ein Verzeichnis in Richtung der root durch wandern, so wird dies mit 2 Punkten '..' notiert.
Jede Stufe wird durch einen Slash '/' getrennt. Die Wurzel (root) wird durch einen Slash '/' dargestellt

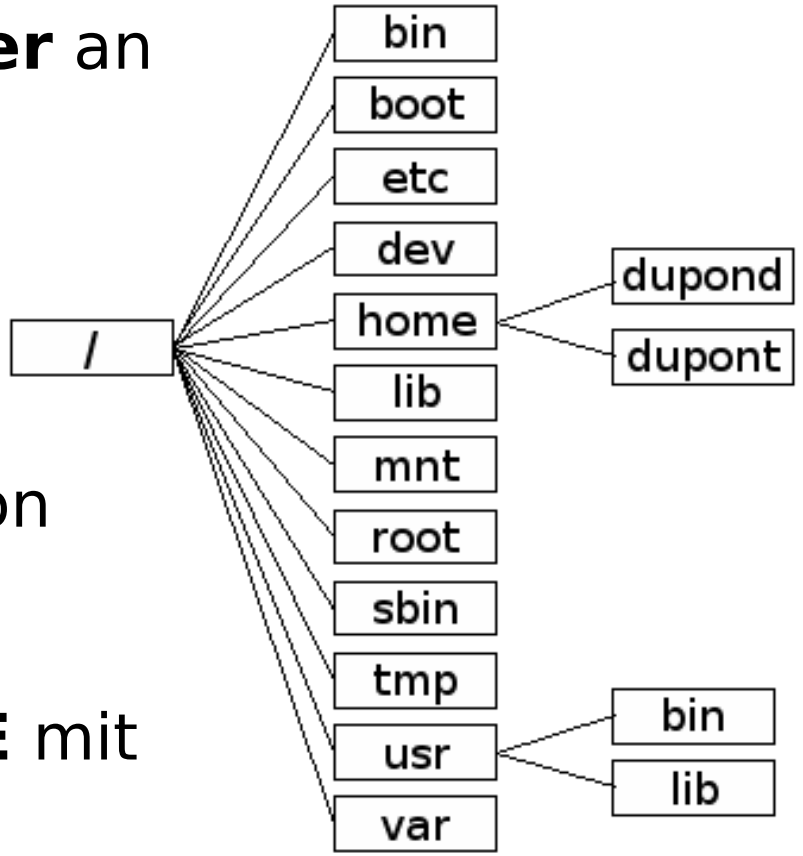


Pfade

Absolute Pfade

Absolute Pfade beginnen **immer** an der Wurzel des Dateisystems.

☞ Absolute Pfade beginnen **immer** mit einem Slash '**/**'



Relative Pfade

Relative Pfade gehen immer von der aktuellen Position im Dateisystem aus.

☞ relative Pfade beginnen **NIE** mit einem Slash '/'

`/home/dupond` ist eine absoluter Pfad.
`../lib` ist ein relativer Pfad.

Pfade

Pfad-Abkürzungen:

~	das Homeverzeichnis des aktuellen Benutzer
~user	Das Homeverzeichnis des Benutzers user
.	das aktuelle Verzeichnis
..	das übergeordnete Verzeichnis

```
$ echo ~
/home/knoppix
$ echo ~root
/root
```

../gugus, /etc Fragen ?

$ ls

ls (**l**i**s**t) erzeugt eine Liste aller Files im aktuellen Verzeichnis bzw der angegeben Pfaden. **ls** ist vergleichbar mit dem Befehl **dir** aus der DOS/Windows-Welt

ls kennt eine Menge von Optionen:

`-a` oder `--all`	Zeigt auch versteckte Dateien
`-i` oder `--inode`	Gibt die eindeutige Inode-Nummer aus
`-l` oder `--long`	Zeigt viele zusätzliche Infos an
`-F` oder `--classify`	Markiert den Dateityp
`-r` oder `--reverse`	kehrt die Sortierreihenfolge um
`-R` oder `--recursive`	durchsucht die Unterverzeichnisse
`-S` oder `--sort=size`	Sortiert anhand der Dateigrösse
`-t` oder `--sort=time`	Sortiert anhand der Modifikationszeit
`-X` oder `--sort=extension`	Sortiert anhand der Extension

Plus viel weitere Optionen (siehe man-page)

$ ls

Bei verschiedenen Distributionen ist ls oft mit einem Alias 'verdeckt'.

Beispielsweise bei Knoppix sind folgende Aliase definiert:

```
alias l='ls -a --color=auto'
alias la='ls -la --color=auto'
alias ll='ls -l --color=auto'
alias ls='ls --color=auto'
```

$ ls

```
$ ls -l *.txt
-rw-r--r-- 1 test users 4711 Oct 4 11:11 datei.txt
```

Beim langen Format wird an der ersten Stelle der FileType (`-`) als Kennbuchstabe angezeigt.
Die restlichen 9 Zeichen (`rw-r--r--`) geben Auskunft über die Zugriffsrechte (siehe Kapitel 7). Danach folgt der Linkreferenz Zähler (`1`), der Eigentümer (`test`) und die Gruppe (`users`), die Grösse (`4711`), das Modifikations Datum (`4. Oct, 11:11`) und der Name der Datei (`datei.txt`).

$ ls

Der Dateityp wird wie folgt angezeigt:

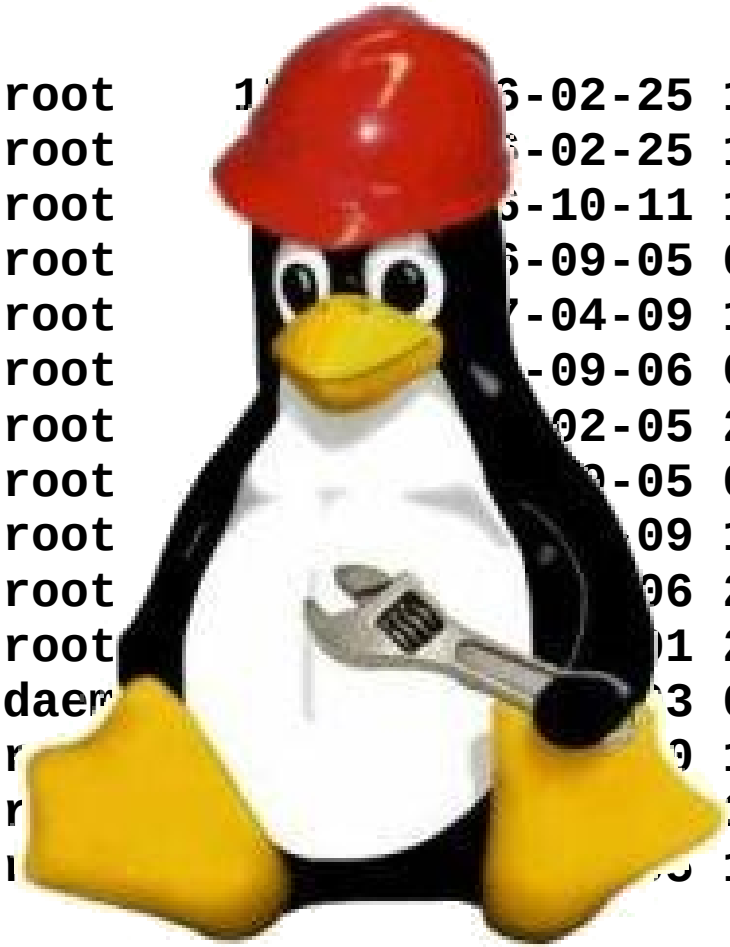
Dateityp	**Farbe** --color	**Zeichen** --classify	**Kennbuchstabe** --long	
gewöhnliche Datei	-	keine	-	
ausführbare Datei	grün	*	-	
Directory	blau	/	d	(directory)
Symbolischer Link	cyan	@	l	(link)

Beispiel:

```
$ ls -1F --color
datei.txt
directory/
gruppe.txt@
programm*
```

ls Fragen ?

```
$ ls -lF /etc/
total 1304
-rw-r--r--  1 root   root     1[illegible]6-02-25 10:52 a2ps.cfg
-rw-r--r--  1 root   root     [illegible]-02-25 10:52 a2ps-site.cfg
drwxr-xr-x  3 root   root     [illegible]-10-11 10:30 acpi/
-rw-r--r--  1 root   root     [illegible]-09-05 08:09 adduser.conf
-rw-r--r--  1 root   root     [illegible]-04-09 12:32 adjtime
drwxr-xr-x  2 root   root     [illegible]-09-06 00:05 afbackup/
-rw-r--r--  1 root   root     [illegible]02-05 20:37 aliases
drwxr-xr-x  3 root   root     [illegible]-05 08:38 alsa/
drwxr-xr-x  2 root   root     [illegible]09 19:22 alternatives/
drwxr-xr-x  6 root   root     [illegible]06 21:14 apm/
drwxr-xr-x  4 root   root     [illegible]1 20:23 apt/
-rw-r-----  1 root   daem     [illegible]3 08:15 at.deny
drwxr-xr-x  3 root   r        [illegible] 15:19 avahi/
-rw-r--r--  1 root   r        [illegible]22:59 bash.bashrc
-rw-r--r--  1 root   r        [illegible] 15:16
bash_completion
-rw-r--r--  1 root   root     1283 2007-03-30 12:00 blkid.tab
-rw-r--r--  1 root   root     1283 2007-03-30 12:00 blkid.tab.old
-rw-r--r--  1 root   root     6813 2007-01-15 03:04 bogofilter.cf
```



Übungen

5.3 Erklären Sie den Unterschied zwischen **ls** mit einem Dateinamen als Argument und **ls** mit einem Verzeichnisnamen als Argument.

5.4 Wie können Sie **ls** dazu bringen, bei einem Verzeichnisnamen als Argument Informationen über das benannte Verzeichnis selbst anstatt über die darin enthaltenen Dateien zu liefern? (Tipp: Dokumentation.)

Verzeichnisse

Nach dem einloggen ist ihr aktuelles Verzeichnis ihr Homeverzeichnis (abgekürzt ~).

Welches das aktuelle Verzeichnis ist, kann mit dem Befehl **pwd** (intern) bzw **/bin/pwd** (extern) abgefragt werden.
Die Environments-Variable $PWD zeigt ebenfalls auf das aktuelle Verzeichnis.
(`ls -l /proc/self/cwd`) zeigt auch aufs aktuelle Verzeichnis

Beispiel:

```
$ pwd
/home/knoppix
```

$ cd

Mit dem Befehl **cd** (change directory) kann in ein anderes Verzeichnis gewechselt werden. Als Argument wird das neue Verzeichnis (als absolute oder relative Pfadangabe) angegeben.
Fehlt das Argument, so wechselt **cd** in das eigene Homeverzeichnis

Beispiel:

```
$ pwd
/home/knoppix
$ cd ../../var/tmp ; pwd
/var/tmp
$ cd ; pwd
/home/knoppix
```

$ cd

mit **cd -** kann ins letzte Verzeichnis gewechselt werden:

Beispiel:

```
$ pwd
/etc
$ cd ~heuer ; pwd
/home/staff/heuer
$ cd - ; pwd
/etc
$ cd - ; pwd
/home/staff/heuer
```

$ mkdir

Eigene Verzeichnisse können mit **mkdir <name>** (**m**a**k**e **dir**ectory) erstellt werden. <name> kann ein absolute oder relative Pfade sein. Mehrere Pfade können gleichzeitig angegeben werden.

Beispiel:

```
$ mkdir test1
$ mkdir test2 test3 test4
$ cd test1 ; mkdir ../test5 ~/test6
```

Die Option **-p** oder **--parents** erzeugt wenn notwendig die Parent-Verzeichnisse

```
$ mkdir --parents test1/sub1/sub2/sub4
```

$ rmdir

Ein leeres Verzeichnis kann mit **`rmdir <name>`** gelöscht werden (Shell-Wildcards sind erlaubt - diese werden durch die Shell expandiert).

Löschen von Directory sub3:

```
$ rmdir test1/sub1/sub2/sub3
```

`rmdir` kennt auch die Option **`-p`** oder **`--parents`**. Wird diese Option verwendet, so werden alle leeren Verzeich-nisse vom angegebenen Pfade von rechts her gelöscht.

Löscht alle Direktories: test1/sub1/sub2/sub3, test1/sub1/sub2, test1/sub1, test1 (sofern keine Dateien darin enthalten sind)

```
$ rmdir -p test1/sub1/sub2/sub3
```

cd, mkdir, rmdir Fragen ?

Übungen

5.5 Ist `cd` ein externes Kommando oder als internes Kommando in die Shell eingebaut? Warum?

5.6 Lesen Sie in der man-Page der **bash** über die Kommandos **pushd**, **popd** und **dirs** nach. Überzeugen Sie sich, dass diese Kommandos funktionieren.

$ cp

Mit dem Befel **cp** (**c**o**p**y) können Dateien kopiert werden.

cp <option> <datei1> <datei2>
Kopiert die `datei1` in die `datei2`. Wenn bereits eine `datei2` existiert, so wird diese überschrieben! Die Option **-i** oder **--interactive** fragt den Benutzer, bevor eine Datei überschrieben wird, oder die Option `-b`, `--backup` erzeugt von der alten Datei ein Backup an.

cp <option> <datei1> <datei2> <directory>
Kopiert `datei1`, `datei2` ins Verzeichnis `directory`. Existiert im Ziel-Verzeichnis schon Dateien mit denselben Namen wie `datei1`, `datei2` ... $datei_n$, so werden diese überschrieben!

$ cp Optionen

`-b, --backup=[CTRL]`	Erzeugt ein Backup, wenn die Datei bereits vorhanden ist. `name` → `name~` oder `name[CTRL]`
`-f, --force`	wenn ein Zieldatei nicht geöffnet werden kann, so lösche diese und versuche es nochmals.
`-i, --interactive`	Frage den Benutzer, bevor ein File überschrieben wird
`-p, --preserve[=attr]`	Kopiert auch den Eigentümer, Gruppe und Zugriffsrechte der Datei
`-r, -R, --recursive`	kopiert auch die Unterverzeichnisse
`-u, --update`	Kopiert nur, wenn die Quelldatei neuer ist
`-v, --verbose`	Zeigt alle Aktivitäten auf dem Terminal an

Plus viel weitere Optionen (siehe man-page)

$ cp und die shell

```
$ cp *txt *bak
cp: target `*bak' is not a directory
```

```
$ ls -1
bar.txt
bla.txt
foo.txt
```

```
$ cp *txt *bak
cp: target `old.bak' is not a directory
```

```
$ ls -1
bar.txt
bla.txt
foo.txt
old.bak
```

Wildcards werden von der Shell ausgewertet und das Ergebnis als Argumente dem Befehl übergeben.
Siehe Kapitel 2 Abschnitt "Suchmuster" und den Befehl **echo**

$ mv

Mit dem Befel **mv** (**m**o**ve**) können Dateien verschoben und umbenennt werden.

mv <option> <datei1> <datei2>
Verschiebt die `datei1` in die `datei2`. Wenn bereits eine `datei2` existiert, so wird diese überschrieben!

mv <option> <datei1> <datei2> <directory>
Veschiebt `datei1`, `datei2` ins Verzeichnis `directory`. Existiert im Ziel-Verzeichnis schon Dateien mit denselben Namen wie `datei1`, `datei2` ... $datei_n$, so werden diese überschrieben!

<datei> kann ein relativer oder absoluter Pfad sein!

$ mv Optionen

`-b, --backup=[CTRL]`	Erzeugt ein Backup, wenn die Datei bereits vorhanden ist. `name` → `name~` oder `name[CTRL]`
`-f, --force`	wenn ein Zieldatei nicht geöffnet werden kann, so lösche diese und versuche es nochmals.
`-i, --interactive`	Frage den Benutzer, bevor ein File überschrieben wird
`-p, --preserve[=attr]`	Kopiert auch den Eigentümer, Gruppe und Zugriffsrechte der Datei
`-r, -R, --recursive`	kopiert auch die Unterverzeichnisse
`-u, --update`	Kopiert nur, wenn die Quelldatei neuer ist
`-v, --verbose`	Zeigt alle Aktivitäten auf dem Terminal an

Plus viel weitere Optionen (siehe man-page)

$ rm

Mit dem Befel **rm** (**re**m**ove**) können Dateien gelöscht werden.

rm <option> <datei1> <datei2>
Löscht die Dateien `datei1` und `datei2`. Gelöschte Dateien können **NICHT** mehr rekonstruiert werden!

Passen sie mit Wildcards auf, schnell ist eine Datei gelöscht, die man weiterhin benötigt hätte!

<datei> kann ein relativer oder absoluter Pfad sein!

$ rm Optionen

`-f, --force`	Fragt den Benutzer nie
`-i, --interactive`	Frage den Benutzer, bevor ein File gelöscht wird
`-r, -R, --recursive`	löscht auch die Unterverzeichnisse
`-v, --verbose`	Zeigt alle Aktivitäten auf dem Terminal an

Dinge, die man nicht tun soll - ausser man will es wirklich tun und man weiss was man tut:

```
$ rm -rf /
```

oder nur mit grösster Vorsicht:

```
$ rm -rf *
```

cp, mv, rm Fragen ?

Übungen

5.7 Legen Sie in Ihrem Home-Verzeichnis eine Kopie der Datei `/etc/services` unter dem Namen `myservices` an.
Benennen Sie sie um in `srv.dat` und kopieren Sie sie unter demselben Namen ins Verzeichnis `/tmp`.
Löschen Sie anschliessend beide Kopien der Datei.

Schreiben sie sich alle notwendigen Befehle so auf, dass sie unabhängig vom aktuellen Verzeichnis ausgeführt werden können. Beispielsweise:
`cp /etc/services ~/myservices`

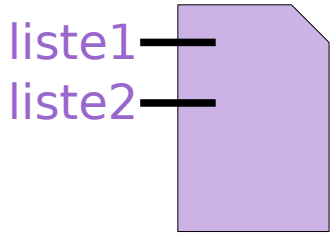
5.8 Warum hat `mv` keine `-R`-Option wie cp?

hard / symbolic link

Unix bietet die Möglichkeit Dateien zu verlinken, so das der gleich Inhalt mit unterschiedlichen Dateinamen angesprochen werden kann.

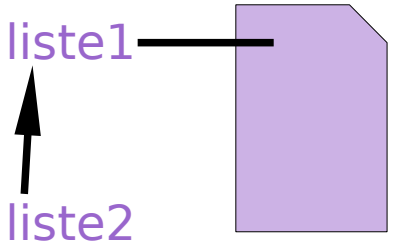
Hard Links:
nur möglich innerhalb einer Partition
nur möglich mit plain Files

```
ln liste1 liste2
rm liste1
```



Symbolische Links:
Auch Partitionsübergreifend möglich
Als Ziel sind alle Files erlaubt

```
ln -s liste1 liste2
rm liste1
```



$ ln

Mit dem Befel **ln** (**l**i**n**k) können Dateien miteinander hart verknüpft werden.

ln <datei1> <datei2>
verknüpft die Datei `datei1` hart mit der neuen Datei `datei2`

<datei> kann ein relativer oder absoluter Pfad sein. Beide müssen innerhalb dergleichen Partition liegen!

$ ln

Beispiel:

```
$ ls -l liste?
-rw-r--r-- 1 heuer staff 67584 2007-04-16 22:50 liste1
$ ln liste1 liste2
$ ls -l liste?
-rw-r--r-- 2 heuer staff 67584 2007-04-16 22:50 liste1
-rw-r--r-- 2 heuer staff 67584 2007-04-16 22:50 liste2
$ md5sum liste?
c287f19c75c6431eeaa897e5c609ea47  liste1
c287f19c75c6431eeaa897e5c609ea47  liste2
$ rm liste1
$ ls -l liste?
-rw-r--r-- 1 heuer staff 67584 2007-04-16 22:50 liste2
$ md5sum liste?
c287f19c75c6431eeaa897e5c609ea47  liste2
```

Welche Dateien gehören zusammen?

Beispiel:

```
$ ls -l liste?
-rw-r--r-- 2 heuer staff 67584 2007-04-16 23:12 liste1
-rw-r--r-- 2 heuer staff 67584 2007-04-16 23:12 liste2
-rw-r--r-- 2 heuer staff 67584 2007-04-16 23:12 liste3
-rw-r--r-- 2 heuer staff 67584 2007-04-16 23:12 liste4

$ md5sum liste?
c287f19c75c6431eeaa897e5c609ea47  liste1
c287f19c75c6431eeaa897e5c609ea47  liste2
c287f19c75c6431eeaa897e5c609ea47  liste3
c287f19c75c6431eeaa897e5c609ea47  liste4

$ ls -i1 liste?     # -i -> show inode number
302123 liste1
213090 liste2
213090 liste3
302123 liste4
```

Die Inode Nummer ist nur innerhalb eines Dateissystemes (Harddisk-Partition) eindeutig

$ ln --symbolic

Mit dem Befel **`ln -s`** (**l**i**n**k) können Dateien miteinander symbolisch verknüpft werden.

`ln --symbolic <datei1> <datei2>`
verknüpft die Datei `datei1` symbolisch mit der neuen Datei `datei2`

`<datei>` kann ein relativer oder absoluter Pfad sein. Beide müssen **NICHT** innerhalb dergleichen Partition liegen!

$ ln --symbolic

Beispiel:

```
$ ls -l liste?
-rw-r--r-- 1 heuer staff 67584 2007-04-16 22:50 liste1
$ ln -s liste1 liste2
$ ls -l liste?
-rw-r--r-- 1 heuer staff 67584 2007-04-16 22:50 liste1
lrwxrwxrwx 1 heuer staff     6 2007-04-16 22:57 liste2 -> liste1
$ md5sum liste?
c287f19c75c6431eeaa897e5c609ea47  liste1
c287f19c75c6431eeaa897e5c609ea47  liste2
$ rm liste1
$ ls -l liste?
lrwxrwxrwx 1 heuer staff     6 2007-04-16 22:57 liste2 -> liste1
$ md5sum liste?
md5sum: liste2: No such file or directory
```

$ ln Optionen

`-b, --backup=[CTRL]`	Erzeugt ein Backup, wenn die Datei bereits vorhanden ist. `name` → `name~` oder `name[CTRL]`
`-f, --force`	Löscht existierende Zieldateien
`-i, --interactive`	Frage den Benutzer, bevor ein File überschrieben wird
`-s, --symbolic`	Erstellt einen symbolischen Link
`-v, --verbose`	Zeigt alle Aktivitäten auf dem Terminal an

In Fragen ?

Übungen

5.9 Erzeugen Sie eine Datei mit beliebigem Inhalt in Ihrem Home-Verzeichnis. Legen Sie unter dem Namen link ein hartes Link auf die Datei an. Überzeugen Sie sich, dass nun zwei Namen für die Datei existieren. Versuchen Sie den Dateiinhalt mit einem Editor zu ändern. Was passiert?

5.10 Legen Sie unter dem Namen ~/symlink ein symbolisches Link auf die Datei aus der vorigen Aufgabe an. Prüfen Sie, ob der Zugriff funktioniert. Was passiert, wenn Sie die Zieldatei des Links löschen?

5.11 Auf welches Verzeichnis zeigt der ..-Link im Verzeichnis "/"?

Übungen

5.12 Betrachten Sie die folgende Ausgabe von `ls -i /`:

```
     2 .    330211 etc        1 proc   4303 var
     2 ..        2 home   65153 root
  4833 bin  244322 lib   313777 sbin
228033 boot 460935 mnt   244321 tmp
330625 dev  460940 opt   390938 usr
```

Offensichtlich haben die Verzeichnisse / und /home dieselbe Inodenummer.
Da es sich dabei offensichtlich nicht wirklich um dasselbe Verzeichnis handeln kann – können Sie dieses Phänomen erklären?

5.13 Wir haben gesagt, dass harte Links auf Verzeichnisse nicht erlaubt sind. Welchen Grund könnte es dafür geben?

5.14 Woran erkennt man in der Ausgabe von "`ls -l $HOME`", dass ein Unterverzeichnis von $HOME keine weiteren Unterverzeichnisse hat?

5.15 (Nachdenk- und Rechercheaufgabe:) Was braucht mehr Platz auf der Platte, ein hartes oder ein symbolisches Link?Warum?

$ less is more

Längere Dateien könnten mit dem Pager-Befehl **more** oder **less** seitenweise angezeigt werden.

less wird vom Befehl man verwendet, um die man-pages seitenweise zu präsentieren.

less kennt sehr viele Optionen die in der man-page Dokumentiert sind.

$ less Befehle

nächste Zeile	[↓], [e], [j],[return]
nächste Seite	[f], [space]
vorherige Zeile	[↑], [y], [k]
vorherige Seite	[b]
Anfang des Files	[g], [Home]
Ende des Files	[End], [G]
springe zur Position *n* Prozent	*n*%, *n*p
vorwärts suchen	/suchtext
rückwärts suchen	?suchtext
weitersuchen	[n]
in der anderen Richtung suchen	[N]
Befehl ausführen	!befehl optionen argument[return]
Beenden	[q]
Bildschirm refresh	[r], [ctrl]-[r], [ctrl]-[l]

$ find

find durchsucht das Filesystem nach Dateien, die bestimmte Kriterien erfüllen (Name, Eigentümer, Grösse, ...)
Zusätzlich können verschiedene Ausgangs-Orte und Bedingungen, die die Dateien erfüllen müssen angegeben werden.

```
$ find TS/.50 -type d
TS/.50
TS/.50/unix
TS/.50/unix/dateien
TS/.50/unix/linux
TS/.50/netzwerktechnologien
TS/.50/netzwerktechnologien/wireshark
```

$ find

```
find [path...] [expression]
```

path...: Gibt die Orte an, in denen find nach Dateien suchen soll. Mehrere Start-Verzeichnisse sind kein Problem.

expression: Definitionen der **Test**, die die Dateien erfüllen müssen (File ist älter als x-Tage, grösser als x Bytes, ...).

Abschliessend werden die **Actions** angegeben, was mit den gefundenen Dateien gemacht werden soll.

$ find; tests

-name Sucht nach passenden Dateinamen. Hierbei sind alle Sonderzeichen nutzbar, die von der Shell zur Verfügung gestellt werden. Mit **-iname** werden dabei Gross- und Kleinbuchstaben gleich behandelt.

-type Ermöglicht die Suche nach verschiedenen Dateitypen (siehe Kapitel 8.2).

-user Sucht nach Dateien, die dem angegebenen Benutzer gehören. Dabei ist statt des Anwendernamens auch die Angabe der eindeutigen Benutzernummer, der UID, möglich.

-group Sucht nach Dateien, die zu der angegebenen Gruppe gehören. Wie bei **-user** ist hier statt des Gruppennamens auch die Angabe der eindeutigen Gruppennummer, der GID, zulässig.

$ find; tests

-size Sucht nach Dateien bestimmter Größe. Zahlenwerte werden dabei als 512-Byte-Blöcke interpretiert, durch ein nachgestelltes **c** sind Größenangaben in Byte, durch **k** in Kibibyte erlaubt. Vorangestellte Plus- oder Minuszeichen entsprechen Unter- und Obergrenzen, womit ein Grössenbereich abgedeckt werden kann.
Das Kriterium `-size +10k` trifft z. B. für alle Dateien zu, die grösser als 10 kByte sind.

-atime (engl. **a**ccess) sucht Dateien nach dem Zeit-punkt des letzten Zugriffs. Hier sowie für die beiden nächsten Auswahlkriterien wird der Zeitraum in Tagen angegeben; . . . min statt . . . time ermöglicht eine minutengenaue Suche. (**-amin**)

-mtime (engl. **m**odification) wählt passende Dateien über den Zeitpunkt der letzten Veränderung aus.

$ find; tests

-ctime (engl. **c**hange) sucht Dateien anhand der letzten Änderung der Inodes (durch Zugriff auf den Inhalt, Rechteänderung, Umbenennen etc.)

-perm Findet nur Dateien, deren Zugriffsrechte genau mit den angegebenen übereinstimmen. Die Festlegung erfolgt mittels einer Oktalzahl, die beim Befehl chmod beschrieben wird. Möchte man nur nach einem bestimmten Recht suchen, muss der Oktalzahl ein Minuszeichen vorangestellt werden, z. B. berücksichtigt **-perm -20** alle Dateien, die Gruppenschreibrechte besitzen, unabhängig von deren übrigen Zugriffsrechten.

-links Sucht nach Dateien, deren Referenzzähler den angegebenen Zahlenwert hat. (nur für "harte" Links!)

-inum Findet Verweise auf die Datei mit der angegebenen Inode-Nummer.

$ find; logische Operatoren

		Bedeutung
!	Nicht	die folgende Bedingung darf nicht zutreffen
-a	and	die Bedingungen links und rechts vom -a müssen zutreffen
-o	or	von den Bedingungen links und rechts vom -o muss mindestens eine zutreffen
( )		Priorisieren von Verknüpfungen

$ find; actions

-delete Löscht die Datei

-exec command ;
Der Befehl command wird für jede gefundene Datei gestartet. Der Name der Datei kann mit {} dem Command übergeben werden.

-exec command {} +
Der Befehl command wird für mehrere gefundene Dateien gestartet. Die Namen der Dateien werden am Ende des command angehängt. Alternativ kann auch folgendes Konstrukt verwendet werden:
`find [path] [expression] -print | xargs command`

-print, -printf format, -print0,

-fprint file, -fprintf file format, -fprint0 file,

$ find; actions

-print Gibt jeden Filenamen auf einer eigenen Zeile aus.

-print0 Gibt jeden Filenamen getrennt mit einem 0-Zeichen aus. Diese Option ist hilfreich, wenn unübliche Filenamen vorkommen (Sonderzeichen, Umlaute, Leerzeichen, ...).

-printf format
Das Format der Ausgabe kann bestimmt werden. Die erlaubten Formatangaben können der man-page entnommen werden.

-fprint file, -fprintf file format, -fprint0 file
Die Ausgabe wird entsprechend in die Datei file geschrieben. An sonst verhalten sich diese Actions gleich wie die Pendants ohne f.

$ find; Beispiele

Beachten sie, dass die Shell auch verschiedene Zeichen selber interpretiert, die auch vom Befehl find interpretiert werden! Schützen sie diese Zeichen!

Suchen sie alle fremden Files in ihrem Homeverzeichnis:
```
$ find \! -user heuer
```

Suchen sie die Files in den Verzeichnissen /var/tmp und ihrem Homeverzeich-nis, die in den letzten 24 Stunden modifiziert wurden und ihnen gehören:
```
$ find /var/tmp ~ -mtime 0 -a -user heuer
```

Suchen sie alle symbolischen Links im Verzeichnis /bin und /usr/bin
```
$ find /bin /usr/bin -type l
```

Suchen sie alle harten Links im Verzeichnis /bin und /usr/bin, Listen sie aber keine Verzeichnisse auf!
```
$ find /bin /usr/bin -links +1 -a \! -type d
```

Finden sie alle Files, die auf id, iD, Id oder ID enden.
```
$ find -iname '*id'
```

find; Fragen ?

$ locate / slocate

find durchsucht bei jedem Aufruf das Filesystem - was seine Zeit in Anspruch nimmt. Normalerweise wird nur nach Dateinamen gesucht.
locate durchsucht nicht das Filesystem, sondern eine Datenbank, in der die Dateinamen einmal im Tag aktualisiert werden. Das Ergebnis kann so viel schneller gefunden werden

```
$ locate pinguin.png
/usr/share/icons/crystalsvg/32x32/apps/pinguin.png
```

mit `locate` dauert das keine Sekunde, mit `find` dauert das ganze mehrere Sekunden lang (mehr als 40Sekunden).

$ locate / slocate

locate versteht die gleichen Suchmuster (?,*,[...]) wie die Shell. Werden Suchmuster angegeben, so muss sichergestellt werden, das diese NICHT von der shell interpretiert werden!
Mit der Option `-r` oder `--regex` interpretiert locate das Suchmuster als regex-Ausdruck.

Findet **locate** im Suchmuster Wildcards, so muss das Suchmuster auf den ganzen Path der Datei passen!

$ locate / slocate

Die Datenbank wird vom Befehl `updatedb` erstellt. Dieses wird automatisch einmal pro Tag gestartet (Das Program `cron` sorgt dafür). Updatedb zeichnet nur die Dateina-men auf. Ob die Datei auch wirklich für den Benutzer sicht-, les- und schreib-bar ist beachtet updatedb und locate nicht.

Will man das nicht, so muss `locate` durch **`slocate`** ersetzt werden. **`slocate`** beachtet die Zugriffs Rechte und zeigt nur Dateien an, die der Benutzer mit seinen Rechten finden kann.

$ which / whereis

which durchsucht den $PATH nach dem gesuchten Programm. Wird eines gefunden, so wird der absolute Pfad zum Programm ausgegeben.
which kennt die internen Shell-Befehle nicht.

```
$ which sudo
/usr/bin/sudo
```

whereis findet neben dem Programm auch man-pages und andere Files, die mit dem Suchbegriff zusammen hängen.

```
$ whereis sudo
sudo: /usr/bin/sudo /usr/lib/sudo /usr/X11R6/bin/sudo
/usr/bin/X11/sudo /usr/share/man/man8/sudo.8.gz
```

Fragen ?

Übungen

5.16 Finden Sie alle Dateien in Ihrem System, die größer als 1 MiB sind, und lassen Sie deren Namen ausgeben.

5.17 Wie können Sie find benutzen, um eine Datei zu löschen, die einen merkwürdigen Namen hat (etwa mit unsichtbaren Kontrollzeichen oder mit Umlauten, die von älteren Shells nicht verstanden werden)?

5.18 Wie würden Sie beim Abmelden dafür sorgen, dass etwaige Dateien in /tmp, die Ihnen gehören, automatisch gelöscht werden?

Übungen

5.19 README ist ein sehr populärer Dateiname. Geben Sie die absoluten Pfadnamen aller Dateien auf Ihrem System an, die README heissen.

5.20 Legen Sie eine neue Datei in Ihrem Homeverzeichnis an und überzeu-gen Sie sich durch einen `locate`-Aufruf, dass diese Datei nicht gefunden wird.
Rufen Sie dann (mit Administratorrechten `/usr/bin/sudo`) das Programm `updatedb` auf. Wird Ihre neue Datei danach mit `locate` gefunden?

5.21 Löschen Sie die Datei wieder und wiederholen Sie die vorigen Schritte.

5.22 Wie heissen auf Ihrem System die ausführbaren Programme, die zur Bearbeitung der folgenden Kommandos herangezogen werden: fgrep, sort, mount, xterm

5.23 Wie heissen auf Ihrem System die Dateien, die die Dokumentation für das Kommando "**`crontab`**" enthalten?

Reguläre Ausdrücke: RegEx

Reguläre Ausdrücke (RegEx) können einfach komplexe Suchmuster beschrieben werden.

Angewendet werden diese Ausdrücke beispielsweise bei der Kontrolle von User-Eingaben. Ein Syntaktisch gültige Emailaddresse kann wie folgt beschrieben werden:

```
/^[a-z0-9_\.\-\+#]+@[a-z0-9\.\-]+\.[a-z]{2,}$/
```

Verschieden Programme und Programmiersprachen unterstützen reguläre Ausdrücke: **less**, **vi**, **locate**, **grep**, **PHP**, **MySQL**, ...

RegEx Definitionen

In der Regel stellt jedes Zeichen sich selbst dar. Das bedeutet, dass ein 'a' das Zeichen 'a' selektiert.

Folgenden Zeichen sind Ausnahmen:

.	Der Punkt ersetzt irgend ein Zeichen (ohne Newline)
*	Ersetzt eine beliebige Anzahl Zeichen welches ihm unmittelbar vorangestellt ist. Da das vorausgehende Zeichen auch ein Regulärer Ausdruck sein kann, ersetzt beispielsweise ".*" eine beliebige Anzahl beliebiger Zeichen.
^	Findet den folgenden Regulären Ausdruck am Anfang einer Zeile.
$	Findet den vorangegangenen Regulären Ausdruck am Ende einer Zeile.
\	Ist das Escape Zeichen, Damit können die speziell interpretierten Zeichen (.*^$[]()|{}+?) 'normalisiert' werden.

RegEx Definitionen

[ ]	Findet irgend eines der eingeschlossenen Zeichen. ^ als erstes Zeichen, kehrt die Bedeutung um. D.h. die folgenden Zeichen sollen nicht gefunden werden. - dient zur Eingabe eines Zeichenbereiches, z.B. [a-m] findet irgend einen der kleinen Buchstaben "a" bis "m" inklusive. Soll das Zeichen "]" mit in der Liste enthalten sein, so muss es am Anfang der Liste auftreten.
()	Mit der runden Klammer können Ausdrücke gruppiert werden. Beispielsweise 'a(bc)*' passt auf ein 'a' gefolgt von beliebig vielen Wiederholungen von 'bc' (auch keine Wiederholung): 'a', 'abc', 'abcbc', 'abcbcbc'
|	Alternativen können mit dem Balken getrennt werden. Beispielsweise passt '(Bahn|Last)wagen' auf 'Bahnwagen' und 'Lastwagen'.

RegEx Definitionen

{n,m}	Wiederholungsoperator für vorangegangenen Regulären Ausdruck, es gibt verschiedene Varianten: {n} findet genau n maliges Auftreten der vorherigen RegExp. {n,} findet mindestens n maliges Auftreten der vorherigen RegExp. {n,m} findet n bis m maliges Auftreten der vorherigen RegExp.
?	Das Fragezeichen als Wiederholungsoperator entspricht {0,1}
+	Ein + als Wiederholungsoperator entspricht {1,}

Die Wiederholungsoperatoren versuchen immer so viele Zeichen wie möglich abzudecken.

RegEx Definitionen

??, +?, *?	Diese Wiederholungsoperator versuchen nur so wenige Zeichen 'einzufangen' dass der Ausdruck erfüllt ist. Beispielsweise a.*?a passt auf 'aa', 'aba' nicht jedoch auf 'abba', ..
\1 .. \9	Mit \1 .. \9 können Rückbezüge auf einen vorher verwendeten RegExp, der mittels einer runden Klammer gruppiert wurde, gemacht werden.
\<	ist ein Wortanfang
\>	ist das Wortende

RegEx Definitionen

[:<class>:] Speziell definierte Klassen. Welche Klassen es gibt ist in `wctype(3)` (wide character classification) definiert. Beispielsweise [:blank:] passt auf Leerzeichen und Tabulatoren.

\d, \D d ⇨ Digit, D ⇨ non Digit (Digit = [0-9], enstpricht [:digit:])

\s, \S s ⇨ Space, S ⇨ non Space (Space = [\t\n\r\f], entspricht [:space:])

\w, \W w ⇨Word, W ⇨ non Word (Word = [a-zA-Z0-9], entspricht [:alnum:])

Die [: :] Klassen beherrschen auch die Sprachspezifischen Sonderzeichen. Dass das Funktioniert müssen die locals auf dem System entsprechend konfiguriert sein.

siehe auch **`regex(7)`**

RegEx testen

Regex zu entwickeln ist nicht einfach!

Testen sie einen Regex immer ob sie wirklich genau das selektieren wird, was sie wollen. Im Internet finden sie verschiedene Webseiten, wo sie ihre Ausdrücke testen können.

Beispielsweise ist `/(list|edit|save)/` falsch, wenn sie nur die Wörter **list**, **edit** oder **save** selektieren wollen. Dies weil auch beispielsweise 'listen', 'editieren', 'saver', ... matchen weil der Ausdruck nicht an eine bestimmte Stelle 'fixiert' wurde.

Richtig wäre `/^(list|edit|save)$/`

RegEx / Shell Suchmuster

Auf den ersten Blick sehen sich RegEx-Ausdrücke und die Suchmuster der Shell ähnlich.

Bei näherem hinsehen gibt es jedoch deutliche Unterschiede:

	Regex	Shell
Ein beliebiges Zeichen	**.**	**?**
Beliebige viele Zeichen	**.***	*
ein beliebiges oder kein Zeichen	**.?**	nicht möglich
ein beliebiges oder mehrere Zeichen	**.+**	nicht möglich

regex Fragen ?

$ grep

RegEx-Ausdrücke alleine nützen nichts. **grep** ist das bekanteste Programm das mit RegEx-Ausdrücken verwendet.

grep durchsucht Files ob ein definiertes Suchmuster vorkommt und gibt die entsprechenden Zeilen aus, in denen das Muster gefunden wurde (ausser die Optionen sagen was anderes).

```
grep [options] PATTERN [FILE...]
```

$ grep Beispiele

Gibt alle Zeilen aus, wo die Zeichen-Kombination **Frosch** vorkommt:

```
$ grep Frosch frosch.txt
Der Froschkönig oder der eiserne Heinrich
Sie sah sich um, woher die Stimme käme, da erblickte sie einen Frosch,
'Sei still und weine nicht', antwortete der Frosch, 'ich kann wohl Rat
'Was du haben willst, lieber Frosch', sagte sie, 'meine Kleider, meine
```

Gibt alle Zeilen aus, wo das Wort Frosch vorkommt:

```
$ grep '\<Frosch\>' frosch.txt
Sie sah sich um, woher die Stimme käme, da erblickte sie einen Frosch,
'Sei still und weine nicht', antwortete der Frosch, 'ich kann wohl Rat
'Was du haben willst, lieber Frosch', sagte sie, »meine Kleider, meine
Der Frosch antwortete: »Deine Kleider, deine Perlen und Edelsteine und
```

Gibt alle Zeilen aus, die mit Frosch beginnen:

```
$ grep ^Frosch frosch.txt
Frosch mag schwätzen, was er will, der sitzt doch im Wasser bei
Frosch.'
Frosch verwandelt worden war, daß er drei eiserne Bänder um sein Herz
```

$ grep; Optionen

-c, --count zählt nur die Anzahl der passenden Zeilen

--color färbt die gefunden Stellen auf dem Terminal ein

-i, --ignore-case
Ignoriert Gross- und Kleinbuchstaben

-l, --files-with-matches
gibt nur die Dateinamen der Files wo der Ausdruck vorkommt

-L, --files-without-match
gibt nur die Dateinamen der Files wo der Ausdruck nicht vorkommt

-n, --line-number
Gibt die Liniennummer, wo der Ausdruck vorkommt an

-v, --invert-match
Gibt die Zeilen aus, wo der Ausdruck nicht vorkommt

$ grep

Neben dem Befehl **grep** gibt es auch noch **fgrep** und **egrep**. **fgrep** ist ein **grep**, das nur einfach Suchmuster erlaubt dafür schneller arbeitet. **egrep** ist ein extended **grep** das zusätzliche Suchmuster erlaubt.

Heute sind **egrep** und **fgrep** in Programm **grep** integriert und können mit Optionen (-F, -E) aktiviert werden.

☞ Gleich wie bei **locate** muss bei **grep** darauf geachtet werden, dass die Shell die Suchmuster nicht selber interpretiert!

$ grep Fragen ?

Übungen

5.24 Sind die Operatoren ? und + in regulären Ausdrücken wirklich nötig?

5.25 Finden Sie in frosch.txt alle Zeilen, in denen das Wort "Tochter" oder "Königstochter" vorkommt.

5.26 In der Datei `/etc/passwd` stehen die Benutzer des Rechners (meistens jedenfalls). Jede Zeile der Datei besteht aus einer Reihe von durch Doppelpunkten getrennten Feldern. Das letzte Feld jeder Zeile gibt die Login-Shell eines Benutzers an. Geben Sie eine grep-Kommandozeile an, mit der Sie alle Benutzer finden können, die die Bash als Login-Shell verwenden.

5.27 Suchen Sie in `/usr/share/dict/words` nach allen Wörtern, die die genau die fünf Vokale "a", "e", "i", "o" und "u" in dieser Reihenfolge enthalten (möglicherweise mit Konsonanten davor, dazwischen und dahinter).

5.28 Geben Sie ein Kommando an, das im "Froschkönig" alle Zeilen sucht und ausgibt, in denen irgendein mindestens vierbuchstabiges Wort zweimal auftritt.