http://www.therealgang.de/

Titel :	UNIX - Einführung
Author :	Ulrike Krauß-Pohl
Kategorie :	Betriebssysteme-Allgemein

Eigenschaften von UNIX

- (Preemptives) **Multi-Tasking** - fähig
  (Time-Sharing-Verfahren)
- **Multi-User** - fähig
- **Multi-Processing** - fähig
- **Netzwerk** - fähig
  (TCP/IP-Protokoll)
- **Hierarchisches Dateisystem**
- Weitestgehende **Geräteunabhängigkeit**
- Hohe **Portabilität** der Kern- und Systemsoftware, da nur ein geringer Teil des Systems in Assembler programmiert ist
- **GUI** (Graphical User Interface)
  = grafische Benutzeroberfläche
  (X-Windows)

UNIX-Versionen

können in **zwei Kategorien** eingeteilt werden:

- **UNIX System V**
  entwickelt von AT&T Bell Laboratories
  mittlerweile **Standard**!!
- **BSD Unix**
  'Berkeley Software Distribution'
  entwickelt an der Universität von Kalifornien in Berkeley
  Vermarktung von Sun Microsystems

Kombination beider Systeme durch die Zusammenarbeit der beiden Firmen AT&T und Sun Microsystems bei der Entwicklung der Version **UNIX System V Release 4**!

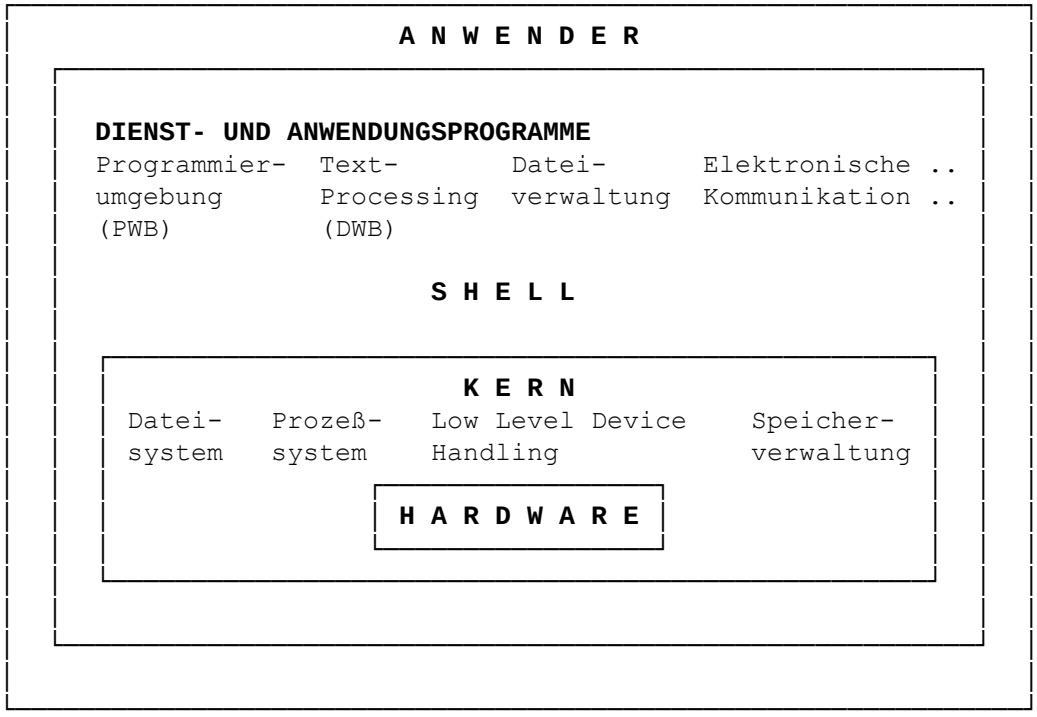
LINUX-Distributionen

Linux wurde von **Linus Torvalds**, einem Studenten der Universität von Helsinki, entwickelt. Der **Grundgedanke** Torvalds war**, Linux (= UNIX-Clone) und seinen Quellcode kostenlos verfügbar** zu machen (= **Open Source)**, so dass jeder Linux kostenfrei nutzen, seinen Quellcode ändern und ihn anderen zur Verfügung stellen kann.

Ursprünglicher Nachteil von Linux, der zunächst einen kommerziellen Einsatz verhinderte, war das Fehlen eines Supports für Unternehmen, die das Betriebssystem einsetzen wollten. Im Laufe der Zeit gab es jedoch immer mehr Firmen, die Linux vertreiben und auch Support bieten.

Als **Linux-Distributionen** werden heute u.a. angeboten:
S.u.S.E, Slackware, Red Hat, Caldera, Debian, ..

System-Struktur



Erläuterungen zur Abbildung:

- Auf der **untersten Ebene arbeitet der Kern mit der Hardware zusammen** und macht **dadurch die Dienst- und Anwendungsprogramme unabhängig von der vorhandenen Hardware (=> einfachere Portierbarkeit**!)
- Die **Shell verkehrt mit dem Kern über einen definierten Satz von Systemaufrufen**.
- Die **Shell stellt dem Benutzer die Dienst- und Anwendungsprogramme zur Verfügung** und ist der **Kommando-Interpreter des UNIX-Systems**.
- Die **Shell** kann **Systemkommandos** (= Kommandos, die vom System als Dienstleistung zur Verfügung gestellt werden) **und Benutzerkommandos** (= vom Benutzer selbst erzeugte Kommandos) ausführen.
- Als **Shells** stehen u.a. die **Bourne-Shell** (= älteste Shell), die **C-Shell** sowie die **BASH-Shell** ('Bourne Again Shell') (= Standard-Shell bei Linux) zur Verfügung.

Dateisysteme unter UNIX

Eine **Harddisk** kann in **mehrere Partitionen** aufgeteilt werden, die **jeweils mit voneinander unabhängigen Dateisystemen und von unterschiedlichen Betriebssystemen formatiert** werden können.

Kommando

⇒**Erzeugen** einer **Partition**

fdisk *laufwerk/gerätedatei*

oder bei S.u.S.E.-Linux über

yast2 -> Hardware -> Partitionieren (in der Modulliste)

Hinweise:

- **Auf einer Festplatte** können **max. 4 Partitionen** (4 primäre oder 3 primäre und 1 erweiterte) angelegt werden. Sind **weitere Partitionen erforderlich**, können **diese als logische Partitionen innerhalb der erweiterten Partition** angelegt werden.
- Die **Nummerierung der logischen Partitionen beginnt bei 5**!

Ein **Dateisystem enthält Verzeichnisse und Dateien sowie Informationen, um diese aufzufinden und zu verarbeiten**. Diese Informationen sind in der Inode des Verzeichnisses/der Datei gespeichert!! **Jedes Dateisystem** hat eine **eigene Inode-Tabelle** zur Verwaltung dieser Inodes!!

Bei der Installation von UNIX muß mindestens ein Dateisystem, das sogenannte **Wurzeldateisystem** (**Root-Dateisystem**) angelegt werden.

Für jedes Dateisystem muß ein **Mount Point** angegeben werden, in den das betreffende Dateisystem montiert wird.

Merke:

- Für das **Wurzeldateisystem** muß der **Mount Point /** festgelegt werden.
- **Weitere Dateisysteme** können **an beliebiger Stelle in die Struktur des Wurzeldateisystems eingehängt (montiert)** werden. Als **Mount Point muß** jedoch ein **leeres Verzeichnis** des Wurzeldateisystems angegeben werden!!
- Auch die **Verzeichnisstruktur auf einer CD-ROM** oder **Diskette** wird als **Dateisystem** betrachtet!! => Um Zugriff auf die CD-ROM/Diskette zu haben, muß das betreffende Dateisystem montiert werden!!
- **Definierte Dateisysteme**, Mount Point, Dateisystemtyp und einige weitere Angaben sind **in der Datei /etc/fstab** gespeichert (siehe folgende Seite).

Hinweis:
In das LINUX-Dateisystem können u.a. auch **FAT-**, **HPFS-** und **NTFS-Dateisysteme** montiert werden!

Listing der Datei /etc/fstab

```
/dev/sda1       /boot           ext2          defaults 1 2
/dev/sda2       /               ext2          defaults 1 1
/dev/sda3       swap            swap          defaults 0 2
/dev/sda5       /home           ext2          defaults 1 2
/dev/sda6       /dos            ext2          defaults 1 2
proc            /proc           proc          defaults 0 0
usbdevfs        /proc/bus/usb   usbdevfs      defaults 0 0
devpts          /dev/pts        devpts        defaults 0 0
/dev/cdrom      /cdrom          auto          ro,noauto,user,exec 0 0
/dev/fd0        /floppy         auto          noauto,user 0 0
```

Dateisystem Gerätedatei	**MountPoint**	**Typ des Dateisystems**	**Optionen** (siehe unten)

Optionen in der Datei **/etc/fstab**:

- **ro** = read only
  montiert das Dateisystem mit Nur-Lese-Recht
- **rw** = read/write
  montiert das Dateisystem mit Lese- und Schreibzugriff
- **auto**
  das Dateisystem wird automatisch beim Booten montiert
- **noauto**
  das Dateisystem wird nicht automatisch beim Booten montiert
- **user**
  erlaubt normalen Benutzern das Montieren des Dateisystems

  Hinweis:
  Nur der Benutzer, der das Dateisystem montiert hat, kann es auch demontieren.
  Sollen **alle Benutzer** das Dateisystem **demontieren** können, so muss **statt user** der Eintrag **users** in die Datei */etc/fstab* erfolgen!
- **nouser**
  verbietet normalen Benutzern das Montieren des Dateisystems
- **exec**
  erlaubt die Ausführung von Binärdateien in dem montierten Dateisystem
- **noexec**
  verbietet die Ausführung von Binärdateien in dem montierten Dateisystem
- **defaults**
  = rw, auto, nouser, exec

- **1** in **Spalte 5**
  das Dateisystem wird vor dem Aushängen gedumpt (= gesichert)
- **0** in **Spalte 5**
  das Dateisystem wird vor dem Aushängen nicht gedumpt (= gesichert)
- **1/2** in **Spalte 6**
  legt die Reihenfolge fest, in der die Dateisysteme beim Booten gecheckt werden
- **0** in **Spalte 6**
  das Dateisystem wird beim Booten nicht gecheckt

Kommandos

⇒**Montieren** eines Dateisystems:
mount [-t *dateisystemtyp***]** *gerätedatei mountpoint*

⇒**Demontieren eines** Dateisystems:
umount *gerätedatei/mountpoint*

Demontieren aller Dateisysteme:
umount -a

Hinweis:
Aktuell benutzte Dateisysteme können nicht demontiert werden!

⇒**Anzeige montierter** Dateisysteme:
df bzw. **mount**

Anzeige eines montierten Dateisystems:
df *dateisystem*

Hinweis:
Die **aktuell montierten Dateisysteme** sind in der Datei **/etc/mtab** gespeichert!

Beispielausgabe des Kommandos **df**:

Filesystem	**1k-blocks**	**Used**	**Available**	**Use%**	**Mounted on**
/dev/sda2	3099140	1273656	1668048	43%	/
/dev/sda1	15522	2757	11964	19%	/boot
/dev/sda5	5162532	177268	4723008	4%	/home
/dev/sda6	396500	13	376006	0%	/dos
usbdevfs	396500	396500	0	100%	/proc/bus/usb

Dateisystem Gerätedatei	**Anzahl Blöcke**	**belegt**	**frei**	**proz. Beleg.**	**Montier-verzeichnis**

⇒**Überprüfen** eines Dateisystems:

fsck *dateisystem*

Hinweis:
Vor dem Überprüfen sollte das betreffende Dateisystem (außer Root-Dateisystem!) demontiert werden!

⇒**Erzeugen** eines **LINUX**-Dateisystems:

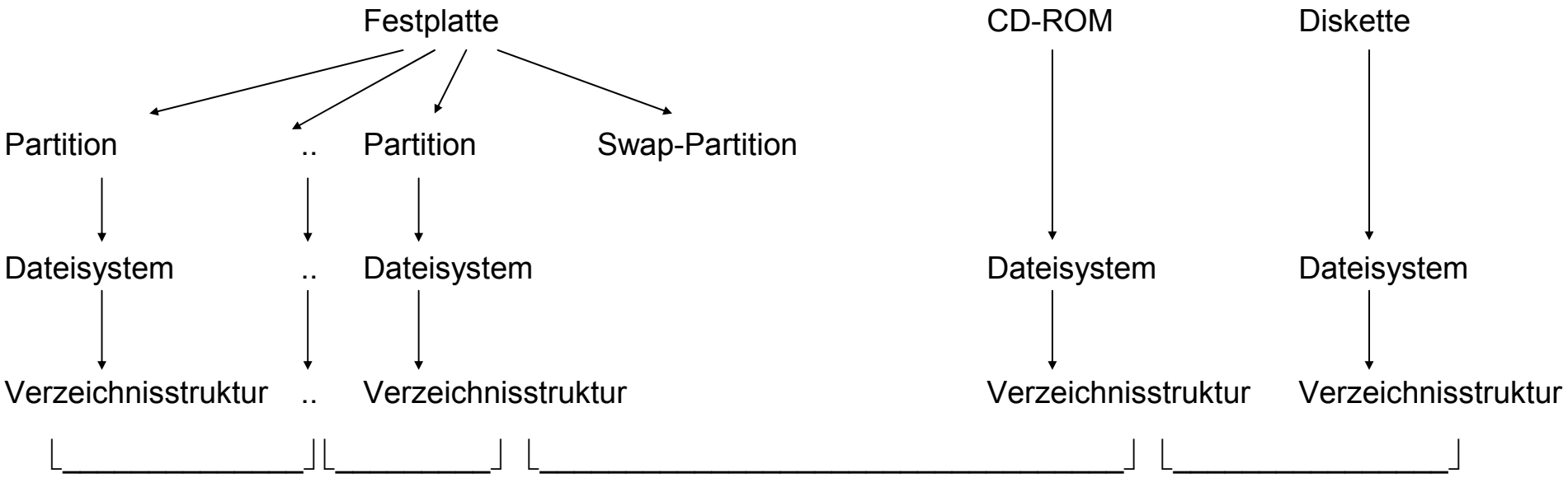
mkfs.ext2 *gerätedatei_für_dateisystem* **[***anzahl_blöcke***]**

Hinweis:

⇒Die Kommandos **fsck** und **mkfs** können **nur vom Superuser ausgeführt** werden!

⇒Zur Ausführung der Kommandos **mount** und **umount** für ein Dateisystem muß **Benutzern** in der Datei **/etc/fstab explizit Berechtigung** gegeben werden (siehe vorherige Seite!).

Graphische Darstellung der Datei(system)verwaltung



Verbindung durch Montieren eines Dateisystems in ein leeres Verzeichnis des Wurzeldateisystems

Merke:
Bei der **Installation von UNIX**
⇒wird auf jeden Fall ein Dateisystem, das **Root-Dateisystem** (Wurzel-Dateisystem) angelegt!
⇒muß eine **Swap-Partition** angelegt werden.

Benutzerverwaltung

♦ Einrichtung eines Benutzers

Die Einrichtung eines **Benutzers erfolgt durch den Systemverwalter** (in S.u.S.E.-LINUX mit Hilfe von *yast2 -> Sicherheit Benutzer*) oder mit den Kommandos **useradd** (Anlegen eines Benutzers) bzw. **userdel** (Löschen eines Benutzers)).

Als **Systemverwalter** ist in UNIX standardmäßig der Benutzer *root* angelegt. Dieser Systemverwalter wird in UNIX als **Super-User** bezeichet! **Nur der Super-User hat alle Rechte im UNIX-System**. Es ist jedoch möglich, verschiedene Rechte des Super-Users auch an "normale" Benutzer zu vergeben.

♦ An- und Abmelden als Benutzer

Anmelden (Einloggen):

Jeder UNIX-Benutzer muß sich beim System anmelden mit
- **Benutzername** und - je nach Einrichtung des Benutzers -
- **Paßwort**

Abmelden (Ausloggen):

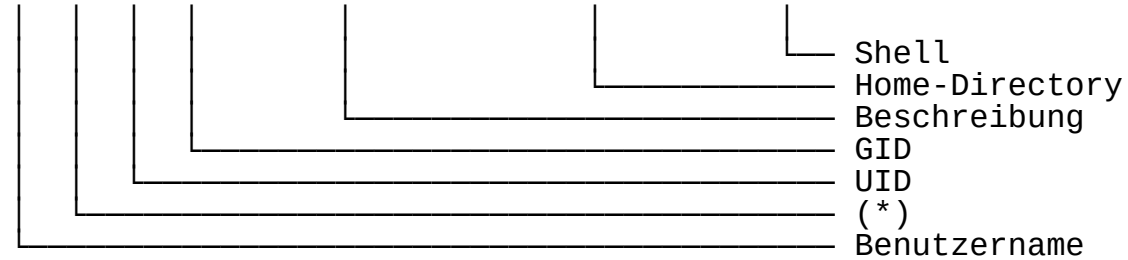
exit

♦ Identifikation eines Benutzers

Die **Identifikation eines Benutzers** beim Einloggen in das System erfolgt über seinen **Benutzernamen** - **intern** wird der Benutzer jedoch über eine Benutzernummer (= **UID**) verwaltet - **und - falls verlangt** - über ein dem Benutzer zugeordnetes **Paßwort**. Außerdem wird jeder Benutzer nach dem Einloggen einer **bestimmten Benutzergruppe zugeordnet** - auch diese Gruppe wird **intern** über eine Gruppennummer (= **GID**) verwaltet. Diese Daten über die definierten Benutzer eines UNIX-Systems sind in der **Datei /etc/passwd** gespeichert. Zusätzlich zu diesen Daten sind in dieser Datei das **Home-Directory** des Benutzers und die Shell vorgegeben, die der Benutzer nach dem Einloggen erhält (= **Login-Shell**).

Beispielhaft könnten die Daten eines Benutzers in dieser Datei wie folgt aussehen:

```
wi1:x:212:50:Erster User WI:/usr/wi1:/bin/sh
```



Erläuterung zu (*):
Das x an dieser Position zeigt an, daß das verschlüsselte Paßwort des betreffenden Benutzers (sowie weitere Informationen zu diesem Paßwort) in der **Datei /etc/shadow** gespeichert sind. Diese Datei ist - im Gegensatz zur obigen Datei **/etc/passwd** - nicht für alle Benutzer lesbar!

Eine **zweite Datei**, die der Identifikation von Benutzern dient, ist die **Datei /etc/group**. In dieser Datei sind die im UNIX-System definierten Gruppen und die zugehörigen Gruppenmitglieder gespeichert.
Hinweis: In S.u.S.E.-LINUX können Gruppen und ihre Mitglieder verwaltet werden mit Hilfe von *yast -> Administration des Systems -> Gruppenverwaltung* oder mit den Kommandos **groupadd** (Anlegen einer Gruppe) bzw. **groupdel** (Löschen einer Gruppe).

Beispielhaft könnten die Daten einer Gruppe in dieser Datei wie folgt aussehen:

```
wi:x:50:wi1,wi2,wi3,wi4,wi5,wi6
|  | |  |
|  | |  └──────────────────────── Benutzernamen der Gruppenmit-
|  | |                           glieder
|  | └─────────────────────────── GID
|  └──────────────────────────── verschlüsseltes Gruppenpaßwort
└─────────────────────────────── Gruppenname
```

Merke:
Ein **Benutzer** kann sein **eigenes Paßwort** mit dem Kommando **passwd setzen bzw. verändern!**

♦ Umgebung eines Benutzers

Durch das Einloggen des Benutzers wird für diesen Benutzer eine Shell gestartet. Die **Shell** ist eine **Benutzeroberfläche** und ein **Kommandointerpreter**, der Kommandos des Benutzers entgegennimmt und die zur Ausführung des Kommandos erforderlichen Prozesse startet. In UNIX können dem Benutzer (je nach Version und Lieferumfang) verschiedene Shells zur Verfügung stehen (z.B. **bash** = Bourne Again-Shell (**LINUX-Standard!**), sh = Bourne-Shell, ksh = Korn-Shell, csh = C-Shell). Die Shell, die beim Einloggen für den Benutzer gestartet wird, ist in der Datei /etc/passwd für den jeweiligen Benutzer festgelegt (siehe oben).

Gleichzeitig mit der Shell wird dem Benutzer eine sogenannte **(Shell-)Umgebung** zur Verfügung gestellt. Zu dieser Umgebung eines Benutzers gehören **mindestens die folgenden Informationen**, die in sogenannten **Umgebungs-Variablen** abgelegt sind:

- **Benutzter Terminaltyp** - abgelegt in **TERM**
- **Promptzeichen der Shell** - abgelegt in **PS1**

- **Home-Directory** - abgelegt in **HOME**
  Hinweis:
  Die Variable HOME wird beim Login des Benutzers mit dem in der Datei /etc/passwd festgelegten Home-Directory belegt!

- **Suchpfad für Programme** - abgelegt in **PATH**
  Hinweis: Die im Suchpfad enthaltenen Verzeichnisses werden jeweils mit : voneinander getrennt.
  Beispiel: PATH=/bin:/usr/bin:/usr/wi1/bin:.

Festlegungen dieser Benutzerumgebungen können
⇒als **Standard systemweit** in der **Datei /etc/profile** vorgegeben werden oder
⇒als **Vorgabe für einen bestimmten Benutzer** in der **Datei .profile**, die für jeden Benutzer in dessen Home-Directory existiert, gespeichert werden!

Merke:
Die **Festlegungen** in der benutzerspezifischen Datei **.profile** (bei SuSE-Linux **.bashrc**) werden bei jedem Einloggen des Benutzers ausgeführt und **haben Vorrang vor den systemweiten Festlegungen** in der Datei **/etc/profile**.

Hinweis:
⇒Der **Inhalt einer Umgebungsvariablen** kann **angezeigt** werden mit dem Kommando
echo $*umgebungsvariable*

⇒Der **Inhalt aller Umgebungsvariablen** kann **angezeigt** werden mit dem Kommando
env

♦ Informations-Kommandos über Benutzer

⇒Anzeige der **Gruppenzugehörigkeiten des Benutzers**
groups

⇒Anzeige der **UID** (Benutzeridentifikationsnummer) und **GID** (Gruppenidentifikationsnummer) **des Benutzers**
id

⇒Anzeige der **Login-Id. des Benutzers**
logname

⇒Anzeige der **eigenen Benutzerdaten**
who am i

⇒Anzeige aller **zur Zeit aktiven Benutzer mit Terminalkennung und Einlog-Zeit**

who

Zusätzliche Anzeige von Stunden und Minuten seit der letzten Aktivität des Benutzer (. bedeutet, daß der Benutzer in der letzten Minute aktiv war)

who –u

Zusätzliche Anzeige, ob das **Terminal des Benutzers für Nachrichtenempfang (**über **write** und **wall) gesperrt** ist

who –T

Nur Anzeige der Benutzerzahl und der Benutzernamen

who -q

Beispielausgabe für das Kommando **who –uT**:

```
root      +     tty1       Aug 16 14:44   00:01
ukp12     -     tty2       Aug 16 14:45   00:01
ukp11     +     pts/0      Aug 16 14:54     .     (192.168.100.5)
                                                  Client-IP-Adresse
Benutzer        Terminal   Datum/Zeit     Zeit seit der letzten
                           d. Anmeldung   Aktivität
          Terminal für Nachrichtenempfang gesperrt (-)
          Terminal für Nachrichtenempfang nicht gesperrt (+)
```

Anmerkung:

Die Terminals **tty1** und **tty2** sind **Terminalsitzungen auf der Konsole,** das **Terminal pts/0 (Pseudoterminal)** ist die **erste geöffnete Terminalleitung (0) über das Netzwerk.**

Grundlegende Kommandos

⇒**Bildschirm** löschen

clear

⇒Anzeige (Ändern nur durch Super-User!) von **Datum und Uhrzeit**

date

⇒Anzeige von **Informationen zu einem Kommando**

man *kommando*

⇒Anzeige des **Namens des aktuellen Terminals**

tty

Abbruch eines UNIX-Kommandos

<STRG> + <c>

Aufgaben

1. Lassen Sie sich das aktuelle Datum anzeigen.
date

2. Stellen Sie fest, welche Benutzer zur Zeit im System aktiv sind.
who

3. Stellen Sie fest, welches Terminal Sie benutzen.
tty

4. Stellen Sie Ihre Benutzer- und Ihre Gruppenidentifikationsnummer fest.
id

5. Informieren Sie sich über den Befehl **who**.
Man who

6. Lassen Sie sich die zur Zeit im System aktiven Benutzer mit der Dauer ihrer Aktivität/Nichtaktivität anzeigen.
Who -u

7. Lassen Sie sich nur Ihre eigenen Benutzerdaten anzeigen.
Who am i

8. Löschen Sie Ihren Bildschirm.
Clear

9. Informieren Sie sich über den Befehl **tty**.
Man tty

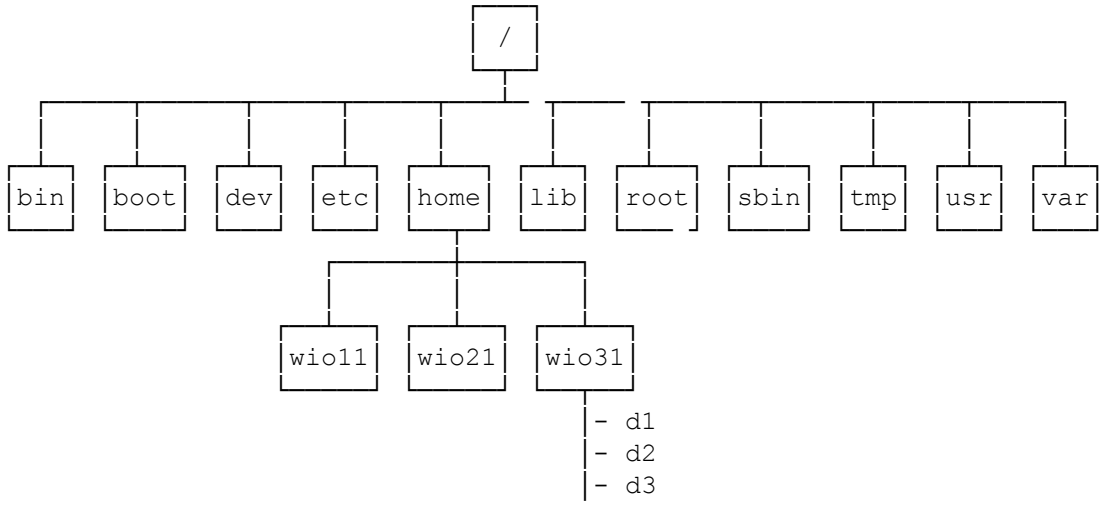
Verzeichnisverwaltung

Verzeichnisse = Dateien, die die Namen von Dateien und Unterverzeichnissen sowie Zeiger (Inode-Number!) auf diese Dateien und Unterverzeichnisse enthalten. =>

Die **Namensvergabe für Verzeichnisse** richtet **sich nach den Konventionen für Dateinamen**!! (siehe Kapital 'Dateiverwaltung')

♦ Aufbau des Wurzeldateisystems

Auszugsweise kann **die Verzeichnisstruktur des Wurzeldateisystems** wie folgt aussehen:



Aus der obigen Abbildung ist ersichtlich, daß die Verzeichnisstruktur eine hierarchische Struktur besitzt, die mit einer umgekehrten Baumstruktur vergleichbar ist. **In diesem Baum existieren Knoten** (dargestellt als **Rechtecke**!) **und Blätter**, wobei **auf der obersten Ebene der Struktur nur ein Knoten - die Wurzel** - auftreten kann.

Die **Knoten** dieses Baumes stellen **Verzeichnisse (directories, Kataloge)** dar, die **Blätter** des Baumes **Dateien** (oder ein leeres Verzeichnis). Der **Knoten auf der obersten Ebene** wird als **Root-Directory/Wurzelverzeichnis** bezeichnet.

Unterhalb des Root-Directorys werden bei der LINUX-Installation die z.T. oben aufgeführten Verzeichnisse angelegt. Dabei enthalten die einzelnen Verzeichnisse folgende Speicherungen für das System:

⇒**boot**
Kernel und Dateien zum Booten des Systems.
Bei **SuSE-Linux standardmäßig eigenes Dateisystem**!

⇒**bin** (binaries)
Kommandos zur Systemverwaltung – ausführbar durch alle Benutzer (Binärdateien).

⇒**dev** (devices)
Gerätedateien.

⇒**etc**
Systemdateien (LINUX-Konfigurationsdateien) für die Verwaltung von z.B. Benutzern, Dateisystemen, ...

⇒ **home**
Arbeitsverzeichnisse für die einzelnen Benutzer.
Empfohlen als **eigenes Dateisystem!**

⇒ **lib** (libraries)
Systembibliotheken.

⇒ **media**
mit den **Unterverzeichnissen cdrom** und **floppy** zum Mounten von CD-ROM und Diskette.

⇒ **mnt**
Mountverzeichnis

⇒ **opt**
Installation von Zusatzpaketen

⇒ **proc**
Interne Prozeßverwaltung

⇒ **root**
Home-Directory des Super-Users.

⇒ **sbin**
Kommandos zur Systemverwaltung – nur von dem Benutzer *root* ausführbar (Binärdateien).

⇒ **tmp** (temporär)
Temporäre Arbeitsdateien, die in bestimmten Zeitabständen gelöscht werden.

⇒ **usr**
Anwendungsprogramme, X-System, .. in Unterverzeichnissen.
Empfohlen als **eigenes Dateisystem!**

⇒ **var**
Dateien, die sich im Laufe der Zeit verändern, normalerweise Protokolldateien.

♦ Adressierung von Verzeichnissen/Dateien

Ein Verzeichnis/eine Datei in dem Verzeichnisbaum wird über einen sogenannten Pfadnamen angesprochen, d.h. man gibt quasi dem System den Weg durch den Verzeichnisbaum zu dem betreffenden Verzeichnis/der betreffenden Datei an. Hierbei unterscheidet man zwischen **absoluten und relativen Pfadnamen**. **Dabei beginnt der absolute Pfadname immer im Wurzelverzeichnis**, **der relative Pfadname im aktuellen Arbeitsverzeichnis**.

Merke:

- Der relative Pfadname darf nie mit einem / beginnen, da der / das Wurzelverzeichnis darstellt!!
- **In einem Pfadnamen müssen die Verzeichnisse der untereinanderliegenden Ebenen jeweils durch / getrennt werden.**

Beispiel:
Falls das aktuelle Arbeitsverzeichnis das Verzeichnis *home* ist, kann die Datei *d1* in einem Kommando angesprochen werden mit
⇒ dem absoluten Pfadnamen */home/wio31/d1*
⇒ dem relativen Pfadnamen *wio31/d1*

Anmerkung:
Es ist auch möglich, von einem untergeordneten Verzeichnis aus ein übergeordnetes Verzeichnis anzusprechen. In diesem Fall wird das übergeordnete Verzeichnis mit *..* angesprochen.

Beispiel:
Falls das aktuelle Arbeitsverzeichnis das Verzeichnis *wio11* ist, kann mit *../..* das Wurzelverzeichnis angesprochen werden!!

Hinweis:
Für jeden Benutzer wird beim Anlegen des Benutzers durch den Systemverwalter sein sogenanntes Home-Directory (üblicherweise als Subdirectory des Verzeichnisses *home*) erstellt. (In der Abbildung auf Seite 15 wären also die Verzeichnisse *wio11*, *wio21*, *wio31* Home-Directories der Benutzer *wio11*, *wio21*, *wio31*)

Ein Benutzer befindet sich nach seiner Anmeldung beim System automatisch in seinem Home-Directory, das in diesem Moment auch sein aktuelles Arbeitsverzeichnis darstellt. Alle vom Benutzer abgegebenen Befehle werden relativ zum aktuellen Arbeitsverzeichnis ausgeführt!!

♦ Kommandos zur Verzeichnisverwaltung

Symbole:

/	Wurzelverzeichnis
.	aktuelles Verzeichnis
..	dem aktuellen Verzeichnis übergeordnetes Verzeichnis

Kommandos:

⇒**Anzeige** des **aktuellen Arbeitsverzeichnisses**
pwd

⇒**Wechsel** in das angegebene Verzeichnis, d.h. dieses wird zum aktuellen Arbeitsverzeichnis
cd *verzeichnisname*

Merke:
Zwischen cd und dem Pfadnamen muß **immer ein Blank** stehen, **auch bei cd ..** und **cd /**

⇒**Wechsel** ins Home-Directory
cd

⇒**Erstellen** des angegebenen Verzeichnisses
mkdir *verzeichnisname*

Hinweise:
- Es können mehrere Verzeichnisse mit einem Befehl erstellt werden, indem die Verzeichnisse mit Blank getrennt hinter **mkdir** angegeben werden!!

- Durch Setzen des **Flags -p** kann **gleichzeitig ein noch nicht existierendes Elternverzeichnis erstellt** werden!!

⇒**Löschen** des angegebenen Verzeichnisses
rmdir *verzeichnisname*

Merke:
Das zu löschende Verzeichnis muß leer sein!!

⇒**Ändern** des Namens eines Verzeichnisses
mv *altname neuname*

⇒**Verschieben** eines Verzeichnisses in der Verzeichnisstruktur
mv *verzeichnisname pfadname*

⇒**Listen des Inhaltes** eines Verzeichnisses
ls [-flags] *verzeichnisname*

Ausführliches Listing eines Verzeichnisinhaltes
l [-flags] *verzeichnisname*

Hinweis:
Ohne Angabe des Verzeichnisses wird der Inhalt des aktuellen Verzeichnisses gelistet!

Anmerkung:
Die verfügbaren Flags für das Kommando ls werden im Kapitel 'Dateiverwaltung' behandelt!

Aufgaben zur Verzeichnisverwaltung

1. Stellen Sie fest, in welchem Verzeichnis Sie sich befinden.

```
pwd
```

2. Legen Sie als Unterverzeichnisse Ihres Home-Directorys die Verzeichnisse *u1*, *u2* und *u3* an.

```
mkdir u1 u2 u3
```

3. Kontrollieren Sie, ob das Anlegen erfolgreich durchgeführt wurde.

```
ls
```

4. Erstellen Sie das Verzeichnis *u1u1* als Unterverzeichnis von *u1*.

```
mkdir u1/u1u1
```

5. Wechseln Sie in das Wurzelverzeichnis.

```
cd /
```

6. Stellen Sie fest, ob Sie sich wirklich im Wurzelverzeichnis befinden.

```
pwd
```

7. Listen Sie den Inhalt des Verzeichnisses *etc*.

```
ls etc
```

8. Verschieben Sie das Verzeichnis *u3* in das Verzeichnis *u1u1*.

mv u3 u1/u1u1

9. Kontrollieren Sie, ob das Verschieben erfolgreich durchgeführt wurde.

ls u1/u1u1(relativ) ls ~/u1/u1u1(absolut)

10. Wechseln Sie in das Verzeichnis *u1u1*.

cd /u1/u1u1(relativ) cd ~/u1/u1u1(absolut)

11. Ändern Sie den Namen des Verzeichnisses *u2* in *u4*.

mv ~/u2 ~/u4(absolut)

mv u2 u4(relativ)

12. Löschen Sie das Verzeichnis *u3*.

rmdir u1/u1u1/u3

13. Wechseln Sie (falls nicht bereits erfolgt) in Ihr Home-Directory.

cd

Dateiverwaltung

Unter UNIX wird **jede physikalische Einheit** (z.B. Festplatten, Drucker, Terminals) **oder logische Einheit** (u.a. Verzeichnisse, normale Dateien) **als Datei im UNIX-Dateisystem** dargestellt!

♦ Aufbau eines ext2-Dateisystems

In einem **Dateisystem** sind als **Komponenten** enthalten:
- **Bootblock**
- **Superblock**
- **Inode-Liste**
- **Datenblöcke**

Das standardmäßig unter LINUX verwendete **Dateisystem ext2** enthält **desweiteren**:
- **Blockgruppen**
- **Bitmaps für belegte Blöcke bzw. Inodes**

Erläuterungen:

▪ Bootblock

⇒ liegt in Block 0 jedes Dateisystems;

⇒ enthält ein Programm zum Starten und Initialisieren des gesamten Systems.

▪ Superblock

enthält Informationen zur Verwaltung des Dateisystems, u.a.:

⇒ Größe des Dateisystems in Blöcken bzw. in Inodes

⇒ Anzahl der freien Blöcke im Dateisystem

⇒ Anzahl der freien Inodes

⇒ Datum der letzten Modifikation

⇒ Datum der letzten Prüfung

⇒ Status des Dateisystems

▪ Blockgruppe

Jede **Blockgruppe** im *ext2*-Dateisystem enthält die **Komponenten**:
- **Superblock**
  => In *ext2*-Dateisystemen existieren mehrere Superblöcke!!
- **Liste der Blockgruppenbeschreibungen**, d.h. welche Blockgruppen insgesamt vorhanden sind und an welcher Stelle der Platte sie gespeichert sind.
- **Bitmap für Blöcke**
  Größe: 1024 Bytes; pro zu verwaltender Block wird ein Bit ('belegt'/'frei') gesetzt => max. 8192 Blöcke verwaltbar!
- **Bitmap für Inodes**
  Größe: 1024 Bytes; pro zu verwaltende Inode wird ein Bit ('belegt'/'frei') gesetzt => max. 8192 Inodes verwaltbar!
- **Inode-Liste**
- **Datenblöcke**

- **Inode-Liste (Dateikopfliste)**

enthält die Inodes des Dateisystems (einer Blockgruppe! bei *ext2*-Dateisystemen). **Jede Inode enthält Informationen über eine gespeicherte Datei**.

Anmerkungen:

- Bei *ext2*-Dateisystemen werden standardmäßig 2048 Inodes in einer Blockgruppe gespeichert.
- Beim Anlegen eines ext2-Dateisystems kann die Inode-Dichte angegeben werden, d.h. die min. Byte/Inode.

- **Datenblöcke**

Die Größe eines Datenblocks ist genau die Größe eines logischen Blocks, der als Basis für das jeweilige Dateisystem gilt (im *ext2*-Dateisystem 1024 Bytes!).

Hinweis:

Die Größe eines physikalischen Blocks auf der Festplatte beträgt 512 Byte. In einem Dateisystem werden mehrere physikalische Blöcke (z.B. zwei, vier oder acht) zu einem logischen Block zusammengefaßt!

=>

Schematische Darstellung eines ext2-Dateisystems

Bootblock	
Blockgruppe_1	Superblock Blockbeschreibungen Bitmap für Inodes Bitmap für Blöcke Inode-Liste Datenblöcke
Blockgruppe_2	Superblock Blockbeschreibungen Bitmap für Inodes Bitmap für Blöcke Inode-Liste Datenblöcke
. .	

♦ Verwaltung und Speicherung einer Datei

Jede Datei in einem **UNIX-Dateisystem erhält eine eindeutige Nummer**, die sogenannte **Inode**, über die der Zugriff und die Verwaltung der Datei erfolgen. Dabei verwaltet **jedes Dateisystem seine eigenen Inodes** (siehe oben!)

In der Inode sind die **folgenden Informationen** gespeichert:

Dateityp und Zugriffsrechte
Referenzzähler
Benutzernummer + Gruppennummer
Länge in Bytes
Erstellungsdatum
Modifikationsdatum
Datum des letzten Zugriffs
Verweis auf Datenblock 1
..
Verweis auf Datenblock 12
Verweis auf 1. Indirektionsblock
Verweis auf Zweifach-Indirektionsblock
Verweis auf Dreifach-Indirektionsblock

Schematisch kann die **Verzeigerung der Datenblöcke** einer UNIX-Datei wie folgt dargestellt werden:

Annahme: 1 logischer Datenblock = 1 KB!:

1. Adresse	Block mit 1024 Zeichen			
2. Adresse	Block mit 1024 Zeichen			
3. Adresse	Block mit 1024 Zeichen			
4. Adresse	Block mit 1024 Zeichen			
5. Adresse	Block mit 1024 Zeichen			
6. Adresse	Block mit 1024 Zeichen			
7. Adresse	Block mit 1024 Zeichen			
8. Adresse	Block mit 1024 Zeichen			
9. Adresse	Block mit 1024 Zeichen			
10. Adresse	Block mit 1024 Zeichen			
11. Adresse	Block mit 1024 Zeichen			
12. Adresse	Block mit 1024 Zeichen			
13. Adresse	**einfacher indirekter Verweis auf einen Block mit**	256 Adressen von Datenblöcken der Datei		
14. Adresse	**doppelt indirekter Verweis auf einen Block mit**	256 Adressen von Blöcken mit	jeweils 256 Adressen von Datenblöcken der Datei	
15. Adresse	**dreifach indirekter Verweis auf einen Block mit**	256 Adressen von Blöcken mit	jeweils 256 Adressen von Blöcken mit	jeweils 256 Adressen von Datenblöcken der Datei
Summe	**12 x 1024 Byte = 12 KB +**	**256 x 1 KB = 256 KB +**	**256 x 256 x 1 KB = 65.536 KB +**	**256 x 256 x 256 x 1 KB = 16.777.216 KB**

♦ Dateitypen

Als Dateitypen werden **unterschieden:**
- **Normale Dateien**
- **Verzeichnisse**
- **Links**
- **Gerätedateien**
- **FIFO-Dateien (Process pipes)**

(1) Normale Datei
z.B. Texte, Programme, Shell-Skripten.

Dateityp: -

Beispiel:

```
-rw-r--r--   1   ukp22   wio22   6243   Nov 20 2001   sortbliste
```

(2) Verzeichnis
Enthält die Namen der darin gespeicherten Dateien und Unterverzeichnisse mit ihrer Inode! Nur bei Symbolischen Links (siehe unten) ist zusätzlich der Pfad zu der Originaldatei im Verzeichniskatalog gespeichert!

Dateityp: d

Beispiel:

```
drwxr-xr-x   2   ukp22   wio22   4096   Nov 13 2001   u1
```

Wichtiger Hinweis:
Nur in der Verzeichnisdatei ist der Name einer Datei/eines Verzeichnisses gespeichert! Die Inode selbst (siehe oben) enthält keinen Namen für die betreffende Datei!!

(3) Link
Ein Link ist ein Verweis auf eine Datei. UNIX **unterscheidet** zwischen **Hard-Link** und **Symbolischem Link**.
Hard-Links können nur innerhalb eines Dateisystems und nur für Dateien erzeugt werden, während
Symbolische Links über verschiedene Dateisysteme und auch für Verzeichnisse erzeugt werden können.

Nur für einen symbolischen Link existiert eine Link-Datei! mit einer eigenen Inode, zu der im Verzeichniskatalog der Pfad auf die Originaldatei gespeichert ist. Wird die Originaldatei gelöscht, so zeigen die Symbolischen Links ins Leere!!
Symbolische Links haben nicht dieselben Berechtigungen wie das Original!! (<=> Hard-Link, siehe unten!)

Dateityp: l

Beispiel:

```
lrwxrwxrwx   1   ukp22   wio22   2   Okt 16 2001   slink -> s1
```

Link name **Original**

Ein **Hard-Link** dagegen ist **nur ein Verzeichnis-Eintrag**, der einer Inode einen weiteren logischen Dateinamen zuordnet. Der **Referenzzähler** (**Link-Counter**) **in der Inode** (siehe S. 20) gibt an, wieviele logische Verweise - d.h. unterschiedliche Dateinamen - auf die betreffenden Datei existieren!

Jeder Hard-Link hat dieselben Berechtigungen auf die Datei, da alle Links auf dieselbe Inode verweisen!! Physikalisch werden die Daten erst gelöscht, wenn der letzte Hard-Link für die Datei gelöscht wird.

Beispiel:

```
-rw-r--r--   3   ukp22   wio22   25   Okt 30 2001   text3
```

Linkcounter

(4) Gerätedatei

Gerätedateien **repräsentieren Ein-/Ausgabegeräte**. Für den Benutzer ergibt sich daraus der Vorteil, daß im UNIX-System Ein-/Ausgabegeräte genau wie Dateien behandelt werden. Über eine Gerätedatei wird auf die erforderliche Geräte-Treiber-Routine (diese gehört zum UNIX-Systemkern) zugegriffen, die die Funktionen des angesprochenen Gerätes steuert. **Gerätedateien** sind standardmäßig im **Verzeichnis /dev** gespeichert. Listet man dieses Verzeichnis mit *ls -l*, so erscheint in der ersten Spalte jedes Eintrages die Geräteklasse.

Man unterscheidet als **Geräteklasse (= Dateityp)**:

- **zeichenorientierte Ein-/Ausgabegeräte**
  (z.B. Bildschirmterminal)
  Dateityp: c
  Beispiel: tty1 = Terminal
- **blockorientierte Ein-/Ausgabegeräte**
  (z.B. Diskette, Magnetband)
  Dateityp: b
  Beispiel: fd0 = 1. Diskettenlaufwerk

Erläuterungen:

⇒ Bei **blockorientierten Ein-/Ausgabegeräten** werden Ein-/Ausgabeoperationen über einen systeminternen Puffer realisiert, um die Anzahl der Ein-/Ausgabeoperationen zu minimieren. Nachteil dieser Pufferung ist jedoch, daß bei einem Systemabsturz der logische Zustand der Datei möglicherweise nicht mit der tatsächlichen physikalischen Speicherung der Daten auf dem Datenträger übereinstimmt.

⇒ **Zeichenorientierte Geräte** verarbeiten die Ein-/Ausgaben als Zeichenstrom, der sequentiell - also unstrukturiert - läuft. Bei diesen Geräten erfolgt keine Zwischenpufferung der Daten.

⇒ Für die **blockorientierten Ein-/Ausgabegeräte** Magnetplatte, Magnetband und Diskette steht **zusätzlich** eine **Schnittstelle**

zur Verfügung, die den direkten, ungepufferten Datentransfer erlaubt (**raw device**). Damit wird gearbeitet, wenn eine besondere Geschwindigkeit erreicht oder spezielle Geräteeigenschaften ausgenutzt werden sollen. Diese Schnittstelle wird als **zeichenorientiert** betrachtet.

Bei Gerätedateien werden nach Abgabe des Befehles *ls -l* anstelle der Speicherplatzbelegung in Byte zwei Nummern, die **major device number** und die **minor device number** angezeigt. Dabei gibt

- die **major device number** an, an welcher Stelle in der Geräte-Treiber-Tabelle des Kernels die erforderliche Geräte-Treiber-Routine steht.
- die **minor device number** an, welche (externe) Schnittstelle angesprochen werden soll.

Beispiellisting von **Gerätedateien** im Verzeichnis **/dev**:

```
brw-rw----   1   root   disk   2,   8   Jul 29 2000    fd0h1200
brw-rw-rw-   1   root   disk   2,  40   Jul 29 2000    fd0h1440
crw--w----   1   root   tty    4,   1   Aug 16 14:44   tty1
crw--w----   1   root   tty    4,  10   Aug 16 14:42   tty10
crw--w----   1   root   tty    4,  11   Jul 29 2000    tty11
```

Typ der Gerätedatei	major device number	minor device number	Gerät

Beispiellisting von **Gerätedateien** im Verzeichnis **/dev/pts** (Pseudoterminals!):

```
crw--w----   1   ukp22  tty  136,   0   Aug 16 15:00   0
crw--w----   1   ukp21  tty  136,   1   Aug 16 15:00   1
```

Typ der Gerätedatei	major device number	minor device number	Gerät

Auf der folgenden Seite sind die **wichtigsten standardmäßig existierenden Gerätedateien** aufgeführt.

Soll ein **neues Ein-/Ausgabegerät** angeschlossen werden, für das **keine passende Gerätedatei existiert**, so muß mit dem Befehl **mknod** und entsprechenden Parametern die erforderliche Gerätedatei angelegt werden!

(5) FIFO-Datei (Pipe-Datei)

Dienen dem Datenaustausch zwischen verschiedenen laufenden Prozessen; werden vom Betriebssystem selbständig verwaltet.

Dateityp: p

Wichtige <u>standardmäßig existierende Gerätedateien</u>

- **Festplattenlaufwerke:**

/dev/hda	erste AT-Bus Festplatte
/dev/hda1 bis /dev/hda15	Partitionen der ersten AT-Bus Festplatte
/dev/sda	erste SCSI Festplatte
/dev/sda1 bis /dev/sda15	Partitionen der ersten SCSI Festplatte
/dev/sdb	zweite SCSI Festplatte
/dev/sdc	dritte SCSI Festplatte

- **Diskettenlaufwerke:**

/dev/fd0	erstes Floppylaufwerk
/dev/fd1	zweites Floppylaufwerk

- **CD-ROM Laufwerke:**

/dev/cdrom	**Link** auf das verwendete CD-ROM Laufwerk (wird von **yast** angelegt!) wie z.B.
/dev/mcd	Mitsumi CD-ROM
/dev/scd0 bis /dev/scd1	SCSI CD-ROM Laufwerke

- **Bandlaufwerke:**

/dev/rmt0	erster SCSI Streamer "rewinding"
/dev/ftape	Floppy-Streamer "rewinding"

- **Mäuse:**

/dev/mouse	**Link** auf die verwendete Maus (wird von **yast** angelegt!)

- **Modem:**

/dev/modem	**Link** auf den com-Port, an den das Modem angeschlossen ist (wird von **yast** angelegt!)

- **Serielle Schnittstellen:**

/dev/ttyS0 bis /dev/ttyS3	Serielle Schnittstellen 0 bis 3

- **Parallele Schnittstellen:**

/dev/lp0 bis /dev/lp2	Parallele Schnittstellen **lpt1 bis lpt3**

- **Spezielle Devices:**

/dev/tty1 bis /dev/tty8	Virtuelle Konsolen
/dev/null	"Datenpapierkorb" (= Daten verschwinden im "Nichts"!)

♦ Zugriffsberechtigungen auf Dateien und Verzeichnisse

Zum Setzen/Ändern von Zugriffsberechtigungen existieren in UNIX als wichtigste Befehle für den Benutzer die Befehle **umask** und **chmod**. Dabei kann der Benutzer mit dem Befehl **umask eine Standardvorgabe für alle im folgenden neuerstellten Dateien/Verzeichnisse** festlegen, während der Befehl **chmod die Zugriffsberechtigungen für bereits existierende Dateien/Verzeichnisse** ändert. Als weiterer Befehl kann der Befehl **chgrp** von dem Besitzer einer Datei/eines Verzeichnisses angewandt werden, um die Gruppenzugehörigkeit zu verändern. Nur für den Super-User (standardmäßig, falls dieses Recht nicht auch anderen Benutzer gegeben wurde!) steht außerdem der Befehl **chown** zur Verfügung, mit dem der Besitzer einer Datei geändert werden kann.

• Anzeige der Zugriffsberechtigungen

Die **Zugriffsberechtigungen können mit dem Kommando ls -l angezeigt** werden!!
Mit diesem Befehl wird **in der ersten Spalte des Inhaltsverzeichnisses eine 10-stellige Anzeige des Dateityps und der Zugriffsrechte** erzeugt.

⇒ Die **erste Stelle** zeigt den **Dateityp**:

-	normale Datei
d	Verzeichnis
b	blockorientierte Gerätedatei
c	zeichenorientierte Gerätedatei
l	Link-Datei
p	Pipe oder FIFO special file

⇒ Die **Stellen 2-4** zeigen die **Zugriffsrechte des Besitzers** der Datei.

⇒ Die **Stellen 5-7** zeigen die **Zugriffsrechte der Besitzergruppe**.

⇒ Die **Stellen 8-10** zeigen die **Zugriffsrechte aller übrigen Benutzer**.

Beispiel:

```
-rw-r--r--   1   ukp22   wio22   24   Okt 30 2001   text1
```

Besitzer (user)
Gruppe (group)
alle anderen Benutzer (other)

Hinweis:
Die **angezeigten Zugriffsrechte gelten nicht für den Super-User**!! Dieser hat alle Rechte auf jede Datei!

Anmerkung:
Ein **Benutzer** kann **seine aktuelle Gruppenzugehörigkeit mit dem Kommando newgrp wechseln**, falls er Mitglied der gewünschten Gruppe ist oder deren Paßwort kennt!!
Um festzustellen, zu **welchen Gruppen** ein Benutzer gehört, steht ihm das Kommando **groups** zur Verfügung.

• Zugriffsrechte

Als grundlegende Zugriffsrechte können für Besitzer, Gruppe bzw. alle übrigen Benutzer die **Rechte**

x **Ausführungsrecht**
r **Leserecht**
w **Schreibrecht**

vergeben werden.

Diese **Zugriffsrechte** (r,w,x) **unterscheiden sich für Dateien und Verzeichnisse** wie in der Tabelle dargestellt:

Zugriffsrecht	**Datei**	**Verzeichnis**
read (r)	lesen	Dateinamen listen
write (w)	Inhalt verändern	Inhalt des Verzeichnisses verändern (= Erzeugen oder Löschen von Dateien)
execute (x)	Inhalt als Programm ausführen	Verzeichnis durchsuchen oder den Befehl **cd** anwenden, um in das Verzeichnis zu gelangen

• Kommandos zum Ändern/Setzen von Zugriffsrechten

⇒**Ändern von Zugriffsrechten**
chmod

Merke:

- Das Kommando **chmod** kann zum Ändern der Zugriffsrechte auf eine Datei/ein Verzeichnis **nur von dem Besitzer der Datei/des Verzeichnisses oder vom Super-User ausgeführt** werden!
- Mit dem Kommando **chmod** können die **Zugriffsrechte absolut (d.h. alle Rechte werden neu festgelegt!!) oder relativ (d.h. alle nicht geänderten Rechte bleiben erhalten)** gesetzt werden!
- Mit der **Option –R** können die Zugriffsrechte **rekursiv**, d.h. für das angegebene Verzeichnis und alle untergeordneten Verzeichnisse/Dateien geändert werden!!

* Relative Festlegung über symbolische Kürzel:

chmod wer+-=recht(e) *datei(en)* für Datei(en)

oder

chmod wer+-=recht(e) *verzeichnisname* für ein Verzeichnis

Bei diesem Aufbau des Befehles stehen die Angaben **wer, +-=, recht(e) für die folgenden möglichen Parameter**:

wer:

a	alle Benutzer
u	Besitzer der Datei
g	Besitzergruppe der Datei
o	alle übrigen Benutzer

+-=:

+	vergibt die angegebene(n) Berechtigung(en)
-	entzieht die angegebene(n) Berechtigung(en)
=	vergibt die angegebene(n) Berechtigung(en) und entzieht alle bisher existierenden Berechtigungen

recht(e):
steht für das/die auf Seite 25 beschriebene(n) Zugriffsrecht(e)

* **Absolute Festlegung über Oktalzahlen:**

chmod *nnnn datei(en)* für Dateien

oder

chmod *nnnn verzeichnisname* für ein Verzeichnis

Dabei stehen die Zeichen *nnnn* für die in der folgenden Tabelle aufgeführten Zugriffsberechtigungen:

	n	*n* **user**	*n* **group**	*n* **others**
read	0	400	40	4
write	0	200	20	2
execute	0	100	10	1

Erläuterungen:

- **Durch Summierung der entsprechenden Zahlen für User, Group bzw. Others werden die jeweiligen Zugriffsberechtigungen vergeben.**

  Beispiel:
  chmod **0755** *beispiel*
  setzt die Rechte der Datei *beispiel* auf rwxr-xr-x.

- **In der ersten Spalte kann** z.B. **für ausführbare Programme das Set-User-ID-Bit (4)** bzw. **das Set-Group-ID-Bit (2) gesetzt** werden.
  Ist dieses Bit **gesetzt, so gehört das laufende Programm dem Besitzer/der Besitzergruppe (= effektive UID/GID) der Programm-Datei, nicht wie im Normalfall dem ausführenden Benutzer (= reale UID). Hierdurch erhält der ausführende Benutzer über das Programm für die Zeit des Programmablaufes die Rechte des Programm-Besitzers/der Besitzergruppe!!**

Ein **Beispiel** hierfür ist das **Programm passwd** zum Ändern eines Paßwortes **in der Datei /etc/passwd**:

```
-rwsr-xr-x  1  root  shadow  27920  Jul 29 2000  /usr/bin/passwd*

-rw-r--r--  1  root  root     6537  Jun 20 16:45 /etc/passwd
```

In diesem Fall erhält der Benutzer durch das **Set-User-ID-Bit** bei Ausführung des **Programmes passwd** die Rechte des Super-Users. Dieser hat Schreibrecht auf die **Datei passwd**, so dass der Benutzer während der Ausführung des Programmes passwd das Schreibrecht auf die Datei passwd erhält und somit sein Paßwort ändern kann.

Hinweis:
s schließt das **x-Recht** mit ein
S schließt das **x-Recht nicht** mit ein!!

- **In der ersten Spalte kann** desweiteren z.B. **für Verzeichnisse das** sogenannte **Sticky-Bit (1), gesetzt** werden.
  Ist dieses Bit **gesetzt,** so ist **das Löschen in dem angegebenen Verzeichnis nur für den Besitzer und den Super-User erlaubt**, auch wenn andere Benutzer Schreibrecht auf dieses Verzeichnis besitzen!!

Beispiel:

```
drwxrwxr-t   2   ukp22   wio21   4096   Okt 16 2001   testz
```

Hinweis:
t schließt das **x-Recht** mit ein
T schließt das **x-Recht nicht** mit ein!!

⇒**Anzeige** der **Standardvorgabe**
umask

⇒**Festlegen** der **Standardvorgabe**
umask *nnn*

Merke:
Die Werte *nnn* **werden von 777 subtrahiert**. Aus dieser Subtraktion **ergeben sich die standardmäßigen Zugriffsrechte**!! => Das Kommando **umask** legt fest, **welche Rechte nicht vergeben** werden!!

Beispiel:

```
umask 022
```

setzt die Zugriffsrechte auf 755. (Bedeutung der Ziffern siehe Seite 28!!)

Hinweise:

* **Standardmäßig** existiert nach dem Einloggen die **Maske 022** (ist in der Datei /etc/profile systemweit gesetzt).
* Die **vom Promptzeichen aus mit umask festgelegte Maske erlischt beim Ausloggen**!
* Der **Befehl umask kann für jeden Benutzer explizit in seiner Datei .profile (.bashrc) gesetzt** werden!

⇒**Ändern der Besitzergruppe** einer Datei

chgrp *gruppname dateiname*

ordnet die Datei *dateiname* der Besitzergruppe *gruppname* zu.

Merke:
Das Kommando **chgrp** kann nur von einem Gruppenmitglied der neuen Gruppe *gruppname* ausgeführt werden!

⇒**Ändern des Besitzers** einer Datei

chown *userid dateiname*

ändert den Besitzer der Datei *dateiname* in *userid*.

Merke:
Das Kommando **chown** ist standardmäßig nur vom Super-User ausführbar!

Erforderliche Zugriffsrechte für die Ausführung der Kommandos zur Verzeichnisverwaltung

cd	**x**-Recht auf das Verzeichnis
mkdir	**w**- und **x**-Recht auf das Zielverzeichnis
mv	**w**- und **x**-Recht auf das Quell- und das Zielverzeichnis
ls	**r**-Recht auf das Verzeichnis
rmdir	**w**- und **x**-Recht auf das Zielverzeichnis

Aufgaben:

1.
Lassen Sie sich die Zugriffsrechte für die Dateien Ihres Home-Directorys anzeigen.
ls -l

2.
Lassen Sie sich die Standardvorgabe für Ihre Zugriffsberech tigungen bei Neuerstellung von Dateien und Verzeichnissen anzeigen.
umask

3.
Ändern Sie die Standardvorgabe dahingehend, daß auch Ihre Gruppe alle Rechte auf Ihre Dateien/Verzeichnisse besitzt.
umask 002
4.
Erstellen Sie in Ihrem Home-Directory ein Verzeichnis *ztest*.
mkdir ztest
5.
Kontrollieren Sie, ob dieses Verzeichnis *ztest* mit den in Aufgabe 4 festgelegten Standard-Zugriffsrechten angelegt wurde.
ls –ld ztest
6.
Entziehen Sie Ihrer Gruppe alle Zugriffsrechte für das Verzeichnis ztest. (Zwei Lösungsmöglichkeiten!!)
chmod g-rwx ztest
chmod 705 ztest
7.
Ändern Sie Ihre Standardvorgabe für Zugriffsberechtigungen dahingehend, daß Ihre Gruppe nur noch Leserecht für alle neuerstellten Dateien/Verzeichnisse besitzt.
umask 032
8.
Erstellen Sie in Ihrem Home-Directory das Verzeichnis *utest* und kontrollieren Sie, ob die Zugriffsrechte nach der Standardvorgabe vergeben wurden.
mkdir utest; ls –ld utest
9.
Ändern Sie die Gruppe des Verzeichnisses *utest* in *informix*.
chgrp informix utest
10.
Sie versuchen, eine Datei aus Ihrem Home-Directory in das Verzeichnis */dev* zu kopieren und erhalten eine Fehlermeldung. Warum kann dieses Kopieren nicht ausgeführt werden?
nur der systemverwalter hat schreibrecht. alle anderen benutzer haben kein schreibrecht auf das verzeichnus /dev und können somit nicht kopieren
11.
Wechseln Sie in das Verzeichnis */chtest*.
Warum kann das Kommando **cd** auf das Verzeichnis nicht ausgeführt werden?
für das cd kommando braucht man execute recht und das fehlt bei der datei

---

Kommandos zur Dateiverwaltung

* Dateiname:

⇒ **Max. 255 Zeichen** lang.
Es werden jedoch aus Kompatibilitätsgründen mit früheren UNIX-Versionen max. 14 Zeichen empfohlen.

⇒ **Verwendbar** sind **Buchstaben, Ziffern und Sonderzeichen**.
Es wird jedoch empfohlen, nur Buchstaben, Ziffern und den Punkt zu verwenden.

Merke:

- Das **UNIX-System unterscheidet bei Dateinamen zwischen Groß- und Kleinbuchstaben**!!

- Wird als **erstes Zeichen** in einem Dateinamen der **Punkt** verwendet, so ist diese **Datei** "**hidden**", d.h. sie wird bei einem Dateilisting mit dem Kommando *ls* nicht angezeigt!!

Zum **Ansprechen einer Gruppe von Dateien** können die folgenden **Meta-Zeichen (Wild Cards, Jokerzeichen)** bei der Angabe eines Dateinamens in den zur Dateiverwaltung aufgeführten Kommandos verwendet werden:

Meta-Zeichen	
?	Ein beliebiges Zeichen
*****	Beliebig viele (auch keine) beliebige Zeichen
[a,b,c]	Genau eines der angegebenen Zeichen a,b,c
[a-f]	Ein Zeichen aus dem angegebenen Bereich a bis f
[!a,b,c]	Keines der angegebenen Zeichen a,b,c
[!a-f]	keines der Zeichen aus dem Bereich a bis f
[^a,b,c]	Keines der angegebenen Zeichen a,b,c
****	Entwertet das folgende Zeichen als Meta-Zeichen

In der **BASH-Shell** steht desweiteren die Möglichkeit der **Zeichenkettenzusammensetzung** zur Verfügung**:**

Beispiel:
{a,b}{1,2,3} liefert a1 a2 a3 b1 b2 b3

* Kommandos:

⇒**Listen** der **angegebene(n) Datei(en)**
ls [-optionen] *datei*(en)

Auswahl möglicher Optionen für den Befehl **ls:**

Anmerkung:
Standardausführung und Optionen sind z.T. versionsabhängig!!

- **a** listet alle Einträge einschließlich versteckter Dateien
- **d** listet bei Directories nur den Namen des Directorys, nicht den Inhalt
- **i** listet die Inode-Number jeder Datei in der ersten Spalte
- **l** listet im ausführlichen Format
- **n** listet anstelle von Benutzer- und Gruppenname UID und GID
- **s** listet die Dateigröße in Blocks
- **t** listet sortiert nach dem letzten Änderungsdatum
- **F** listet Verzeichnisse mit nachfolgendem / und Programme mit nachfolgendem *
- **R** listet die Inhalte des angegebenen Verzeichnisses und aller Unterverzeichnisse

Hinweis: Für dieses Kommando ls -alF kann auch das Kommando **l** verwendet werden!

☞**Erforderliche Zugriffsrechte:** Lese- und Execute-Recht auf das Verzeichnis

Beispielausgabe für das Kommando **ls –lF:**

```
 -rw-r--r--  1 ukp22 wio22   6243   Nov 20 2001  sortbliste
 -rwxr-xr-x  1 ukp22 wio22   6243   Nov 20 2001  testprogramm*
 drwxr-xr-t  2 ukp22 wio21   4096   Okt 16 2001  testverzeichnis/
 -rw-r--r--  1 ukp22 wio22     24   Okt 30 2001  text1
(1)(2)       (3) (4)   (5)     (6)    (7)          (8)
```

Erläuterung des Listings:

(1) Dateityp
(2) Zugriffsrechte
(3) Link-Counter
(4) Besitzer der Datei
(5) Gruppe, die Rechte auf die Datei besitzt
(6) Speicherplatzbelegung in Byte
(7) Datum/Uhrzeit der Erstellung/letzten Modifikation
(8) Dateiname

Beispielausgabe für das Kommando **ls -il**:

```
352234   -rw-r--r-- ..... sortbliste
352232   -rwxr-xr-x ..... testprogramm
352432   drwxr-xr-t ..... testverzeichnis
```

Inode-Number

Beispielausgabe für das Kommando **ls -sl**:

```
0        lrwxrwxrwx ..... slink -> s1
8        -rw-r--r-- ..... sortbliste
4        -rwxr-xr-x ..... testprogramm
4        drwxr-xr-t ..... testverzeichnis
```

Belegte Datenblöcke

⇒**Kopieren** von **Datei(en)**

Kopieren einer **Datei** auf einen **anderen Namen**
cp [-optionen] *quelldatei zieldatei*

Kopieren von **Datei(en) in** ein **Verzeichnis**
cp [-optionen] *quelldatei(en) verzeichnis*

Kopieren eines **Verzeichnisses in** ein **Verzeichnis**
cp [-optionen] -**r** *quellverzeichnis zielverzeichnis*

U.a. mögliche Optionen:
f unterdrückt Warnmeldungen
i fordert eine Bestätigung bevor die Datei überschrieben wird
v listet die Dateien während des Kopiervorganges

☛**Erforderliche Zugriffsrechte:** Execute-Recht für das Quellverzeichnis, Leserecht für die Quelldatei, Schreib- und Execute-Recht für das Zielverzeichnis

Hinweis:
Die **Zieldatei** erhält **standardmäßig als Besitzer den Benutzer, der die Kopie erstellt, und dessen Gruppenzuhörigkeit**! sowie **das aktuelle Datum**. Bei Verwendung der **Option p** bleiben Besitzer, Gruppenzugehörigkeit, Zugriffsrechte und Datum der letzten Modifikation **unverändert**!!

Achtung:
Existiert die Ziel-Datei bereits, so wird diese Datei überschrieben, falls der Benutzer Schreibrecht für diese Datei hat! Falls die Ziel-Datei schreibgeschützt ist, so erfolgt eine Abfrage vor dem Kopieren.

⇒**Verschieben** von **Datei(en)** in ein **Verzeichnis**

mv [-optionen] *datei(en) verzeichnis*

U.a. mögliche Optionen:
- **f** unterdrückt Warnmeldungen
- **i** fordert eine Bestätigung bevor die Datei überschrieben wird
- **v** listet die Dateien während des Verschiebevorganges

Hinweis:
Das Kommando **mv** kann auch zum Umbenennen einer Datei wie folgt eingesetzt werden:

mv [-optionen] *altname neuname*

Achtung:
Existiert die Ziel-Datei bereits, so wird diese Datei überschrieben!! Falls die Ziel-Datei schreibgeschützt ist, so erfolgt eine Abfrage vor dem Umbenennen/Verschieben.

☞**Erforderliche Zugriffsrechte:** Schreib- und Execute-Recht für das Quellverzeichnis und das Zielverzeichnis

⇒**Löschen** von **Datei(en)**

rm [-optionen] *datei(en)*

U.a. mögliche Optionen:
- **f** unterdrückt Warnmeldungen
- **i** bewirkt eine explizite Abfrage für das Löschen jeder angesprochenen Datei
- **v** listet die Dateien während des Löschvorganges

Löschen eines **Verzeichnisses mit allen Inhalten!**

rm [-optionen] -r *verzeichnis*

☞**Erforderliche Zugriffsrechte:** Lese-und Schreibrecht für das Quellverzeichnis, in dem die Datei(en) gespeichert sind

⇒**Anzeige** des **Dateiinhaltes**

cat dateiname — **einfachste** Anzeige für kurze ASCII-Dateien
Anmerkung: Der eigentliche Zweck des Kommandos ist das Verketten von Dateien zu einer neuen Datei!

more dateiname	**seitenweise Anzeige** <enter> eine Zeile weiter <Leertaste> ein Bildschirm weiter <q> verläßt die Anzeige
less dateiname	**seitenweise Anzeige** <enter> eine Zeile weiter <Leertaste> ein Bildschirm weiter <b> ein **Bildschirm zurück**! <q> verläßt die Anzeige

☛**Erforderliche Zugriffsrechte:** Execute-Recht für das Verzeichnis, Leserecht für die Datei(en)

Aufgaben:

1.
Wechseln Sie in Ihr Home-Directory.

2.
Erstellen Sie drei verschiedene Textdateien mit den Namen *text1, text2, text3* in Ihrem Home-Directory.

Gehen Sie dabei wie folgt vor:
- Geben Sie den Befehl *cat > dateiname* ab.
- Schreiben Sie den gewünschten Text; schließen Sie dabei jede Zeile mit <enter> ab.
- Schließen Sie die Textdatei am Anfang einer neuen Zeile mit <STRG>+<d>.

cat > text1

3.
Kopieren Sie die Datei *text1* in das Unterverzeichnis *u1*.
cp text1 u1/

4.
Kopieren Sie alle drei Dateien in das Unterverzeichnis *u4*.
cp text[1,2,3] u4

5.
Verschieben Sie die Datei *text2* aus dem Unterverzeichnis *u4* in das Verzeichnis *u1u1*.
mv u4/text2 u1/u1u1

6.
Kopieren Sie die drei Dateien *text1*, *text2*, *text3* in Ihrem Home-Directory auf die Namen *atext1*, *atext2*, *atext3*. **cp text1 atext1, cp text2 atext2, cp text3 atext3**

7.
Listen Sie in Ihrem Home-Directory alle Einträge, deren Name aus zwei Zeichen besteht. **ls -d ??**

8.
Ändern Sie in dem Verzeichnis *u4* den Namen der Datei *text1* in *ztext1*. **cd u4, mv text1 ztext1**

9.

Listen Sie in Ihrem Home-Directory alle Einträge, deren Name mit *t*,*u* oder *z* beginnt. **ls -d [u,z,t]***

10.
Listen Sie Ihr Home-Directory mit allen Einträgen sowie alle Unterverzeichnisse Ihres Home-Directorys mit allen Einträgen.
ls –R $HOME
11.
Löschen Sie die Dateien im Verzeichnis *u4* (mit Abfrage für jede Datei!).
rm –i u4/*
12.
Löschen Sie das Verzeichnis *u1u1*.
rm –r u1/u1u1
13.
Listen Sie in Ihrem Home-Directory alle Einträge, in deren Name die Ziffer *1* enthalten ist.
ls –d *1*
14.
Listen Sie in Ihrem Home-Directory alle Einträge, deren Name nicht mit *a* beginnt.
ls [!a]*
15.
Listen Sie die Einträge des Verzeichnisses */bin*.
ls /bin |more
16.
Kopieren Sie das Verzeichnis *u1* auf den Namen *ku1*.
cp –r $HOME/u1 $HOME/ku1
17.
Kopieren Sie die Datei *atext1* aus Ihrem Home-Directory in das Verzeichnis */dev*.
Warum kann dieses Kopieren nicht funktionieren?
alle anderen Benutzer (other) haben kein Schreibrecht auf das Verzeichnis /dev
18.
Erzeugen Sie mit dem cat-Befehl eine Datei *kopie_test* mit dem folgenden Text: *Dies ist ein Kopierversuch*!
cat > kopietest
19.
Versuchen Sie, die Datei *kopie_test* in das Home-Directory Ihres Nachbarn zu kopieren.
Warum kann der Befehl u.U. funktionieren oder auch nicht?
ist abhängig davon, ob der Benutzer seiner Gruppe oder allen anderen Benutzern Schreibrecht auf sein Home-Directory gegeben hat

---

* Weitere Kommandos:

⇒ **Suchen** von **Datei(en) ab dem angegebenen Verzeichnis**
find *verzeichnisname* **SUCHBEGRIFF**

U.a. mögliche Einträge für SUCHBEGRIFF:

-name *dateiname*	sucht die Datei mit dem angegebenen Namen
-type *dateityp*	sucht Dateien vom angegebenen Dateityp
-inum *inodenummer*	sucht Dateien mit der angegebenen Inode-Nummer

-user *username*	sucht Dateien des angegebenen Benutzers
-group *gruppenname*	sucht Dateien der angegebenen Gruppe
-perm *nnn*	sucht Dateien mit den als Oktalzahl angegebenen Zugriffsrechten

Merke:

- Werden **im gesuchten Dateinamen Wild Cards** verwendet, so muss der **Name in Anführungsstriche** gesetzt werden.
- Es können **zwei Optionen mit Blank getrennt** angegeben werden. In diesem Fall werden beide Optionen als **logische UND-Verknüpfung** betrachtet!
- Es können **zwei Optionen mit -o getrennt in der Klammerung \(\)** angegeben werden. In diesem Fall werden beide Optionen als **logische ODER-Verknüpfung** betrachtet!
  Beispielsweise sucht das Kommando
  ```
  find / \( -perm 644 -o -perm 664 \)
  ```
  ab dem Root-Directory nach Dateien mit den Zugriffsrechten 644 oder 664.
- Auf mit dem find-Kommando gefundene Dateien kann **durch die Option -exec ein weiteres Kommando zur Ausführung gestartet** werden.
  Beispielsweise löscht das Kommando
  ```
  find . -name text1 -exec rm {} \;
  ```
  alle ab dem aktuellen Arbeitsverzeichnis gefundenen Dateien mit dem Namen text1.

☞**Erforderliche Zugriffsrechte:** Lese- und Execute-Recht für das/die Verzeichnisse.

⇒**Link** auf eine **Datei**

Hard-Link mit anderem Dateinamen
ln *quelldatei zieldatei*

Hard-Link in anderes Verzeichnis
ln *quelldatei verzeichnisname*

Symbolischer Link mit anderem Dateinamen
ln -s *quelldatei zieldatei*

Symbolischer Link in anderes Verzeichnis
ln -s *quelldatei verzeichnisname*

☞**Erforderliche Zugriffsrechte:** Execute-Recht für das Quellverzeichnis, Schreib- und Execute-Recht für das Zielverzeichnis

⇒**Anzeige** des **Speicherplatzbedarfs in Blöcken**

kumuliert für **jedes Verzeichnis ab** dem **angegebenen Verzeichnis**
du [-optionen] *verzeichnis(se)*

U.a. mögliche Optionen:

a	zusätzlich Anzeige des Speicherplatzbedarfes für jede einzelne Datei
b	Anzeige des Speicherplatzbedarfes in Byte
c	Anzeige des Gesamt-Speicherplatzbedarfs der (angegebenen) Verzeichnisse
s	Anzeige des kumulierten Speicherplatzbedarfes für die angegebenen Verzeichnisse ohne Anzeige der Unterverzeichnisse
S	keine Kumulierung des Speicherplatzbedarfs je Verzeichnis

☛**Erforderliche Zugriffsrechte:** Lese- und Execute-Recht für das/die Verzeichnis(se)

Aufgaben:

20.
Lesen Sie den Inhalt der Dateien *atext1*, *atext2* und *atext3*.
more atext*
21.
Suchen Sie ab dem Verzeichnis */usr/bin* alle Dateien des Benutzers *root*.
find /usr/bin –user root
22.
Suchen Sie ab Ihrem Home-Directory alle Dateien mit Ihrem Usernamen.
find $HOME –user $LOGNAME |more (oder mj anstatt $LOGNAME)
23.
Erzeugen Sie einen Link für die Datei *ztext1* in dem Verzeichnis *u4* mit dem Dateinamen *ltext1* ins Verzeichnis *u1*, sowie mit dem Dateinamen *ltext11* in Ihr Home-Directory.
ln u4/ztext1 u1/ltext1
ln u4/ztext1 ltext11
24.
Lassen Sie sich ab Ihrem Home-Directory die drei Dateinamen des in Aufgabe 23 erzeugten Links anzeigen.
ls –i u4/ztext1 (Feststellen der inode-Number)
find –inum 1297 (1297=inodenumber)
25.
Lassen Sie sich ab dem Hauptverzeichnis alle Dateien vom Dateityp *c* **oder** vom Dateityp *b* anzeigen.
find / \(-type c –o –type b\) |more
26.
Lassen Sie sich ab Ihrem Home-Directory alle Verzeichnisse anzeigen, in denen die Datei *text1* existiert.
find $HOME –name text1
27.
Lassen Sie sich ab Ihrem Home-Directory den Inhalt der drei durch Aufgabe 23 entstandenen Dateien *ztext1*, *ltext1* und *ltext11* nacheinander anzeigen.
find –inum 1297 –exec more {} \;
28.
Lassen Sie sich ab Ihrem Home-Directory alle Directories im ausführlichen Format anzeigen.
find –type d –exec ls –ld{} \;
29.
Suchen Sie ab dem Root-Directory alle Dateien Ihrer Benutzer-Gruppe.
find / -group wb4 |more
30.
Listen Sie den Speicherplatzbedarf Ihres Home-Directorys für jedes Verzeichnis und für jede Datei sowie den Gesamt-Speicherplatzbedarf.
du –ac |more
31.
Listen Sie den - nicht kumulierten - Speicherplatzbedarf der Verzeichnisse *u1* und *u4* und ihrer Unterverzeichnisse sowie den Gesamtspeicherplatzbedarf.
du –Sc u1 u4

32.
Listen Sie den Gesamt-Speicherplatzbedarf des Verzeichnisses */dev* ohne Anzeige der Unterverzeichnisse.

```
du –s/dev
```

Kommandoverarbeitung unter UNIX

Mit Hilfe der Shell (dem Kommandointerpreter) können **Kommandos verarbeitet** werden:

- im **Dialogbetrieb**
- im **Stapelbetrieb**
  = Abarbeitung einer Batch-Datei, die in UNIX als **Shellskript** bzw. **Shellprozedur** bezeichnet wird.

In UNIX werden **unterschieden**:

- **interne Kommdos**
  werden von der Shell direkt ausgeführt, d.h. diese Kommandos existieren nicht als Programme oder Shellskripten
- **externe Kommandos**
  existieren als Programme oder Shellskripten und werden von der Shell als Sohnprozesse ausgeführt. Dadurch können mehrere Kommandos gleichzeitig ausgeführt werden!

Anmerkung:
Die meisten Kommandos sind unter UNIX als externe Kommandos realisiert. Dadurch können diese Kommandos in den verschiedenen Shells existieren und auf gleiche Art und Weise benutzt werden!

Zur **Abkürzung** von häufig benötigten, umfangreichen Kommandos können **Alias-Kommandos** definiert werden mit:

alias *aliasname=´kommando´*

Merke:

Alias-Abkürzungen

- können mit **unalias** *aliasname* wieder **gelöscht** werden.
- **gelten nur bis zum Verlassen der Shell.**
- können **zur ständigen Verwendung** in der Datei **.profile** bzw. **.bashrc** im Home-Directory des betreffenden Benutzers **gespeichert** werden!

UNIX-Kommandos können im **Vordergrund** oder im **Hintergrund** ausgeführt werden. Die Ausführung im Hintergrund ist jedoch nur sinnvoll, wenn das Kommando keine Tastatureingaben erfordert und keine Bildschirmausgaben erzeugt bzw. wenn die Bildschirmausgaben umgeleitet werden! (siehe folgende Seite).

Soll ein **Kommando im Hintergrund** ausgeführt werden, so muß dem Kommando das Zeichen **&** (mit führendem Leerzeichen!) **nachgestellt** werden.

Anmerkung:
Nach der Eingabe eines Kommandos im Hintergrund wird auf dem Bildschirm die Nummer des Prozesses angezeigt, durch den das Kommando abgearbeitet wird. Über diese Nummer kann der **Prozeß** mit dem Kommando **kill -9** *prozessnummer* **abgebrochen** werden! Mit dem Kommando **ps** kann **ein Benutzer seine von seinem eigenen Terminal aus gestarteten und zur Zeit noch aktiven Prozesse anzeigen**.

In UNIX können nach dem Promptzeichen **in einer Zeile mehrere Kommandos eingegeben** werden.

Dabei können die folgenden Abarbeitungen unterschieden werden:

- *kommando_1* **&** *kommando_2*
  bewirkt, dass das erste Kommando im Hintergrund und das zweite Kommando im Vordergrund ausgeführt wird.
- *kommando_1* **;** *kommando_2*
  bewirkt, dass die beiden Kommandos **nacheinander** ausgeführt werden, d.h. diese Kommandos werden so ausgeführt, als ob sie jeweils in einer eigenen Zeile eingegeben worden wären.
- *kommando_1* **&&** *kommando_2*
  bewirkt, dass das **zweite Kommando nur ausgeführt** wird**, wenn zuvor das erste Kommando erfolgreich** (d.h. Rückgabewert 0) **ausgeführt** wurde!
- *kommando_1* **||** *kommando_2*
  bewirkt, dass das **zweite Kommando nur ausgeführt** wird**, wenn bei der Ausführung des ersten Kommandos ein Fehler** (d.h. Rückgabewert ≠ 0) **aufgetreten** ist!

In UNIX kann eine **verschachtelte Ausführung von Kommandos** durch **Kommando-Substitution** erreicht werden. Die Substitution eines Kommandos erfolgt durch Verwendung der folgenden Syntax

kommando_1 `` `kommando_2` ``

bzw.

kommando_1 **$(***kommando_2***)**

und bewirkt, **dass bei Ausführung von Kommando_1 das Ergebnis der Ausführung von Kommando_2 verwendet** wird.

Eine weitere Möglichkeit im Zusammenhang mit der Kommandoverarbeitung unter UNIX stellt die **Umleitung von Kommandoeingaben** und **-ausgaben** dar.

Für die Kommandoein- bzw. ausgabe existieren die folgenden Standarddateien:

- die **Standardeingabe** (stdin, Bezeichner 0)
  = das Gerät, von dem aus die Eingaben an einem Terminal erfolgen, normalerweise die Terminaltastatur.
- die **Standardausgabe** (stdout, Bezeichner 1)
  = das Programmes passwd die Rechte denes Kommandos ausgegeben werden, normalerweise der Terminalbildschirm.
- die **Standardfehlerausgabe** (stderr, Bezeichner 2)
  = das Gerät, an dem Fehlermeldungen, die bei der Ausführung eines Kommandos u.U. entstehen, ausgegeben werden, normalerweise der Terminalbildschirm.

Eine **Umleitung** der oben **genannten Standardein- bzw. -ausgaben** kann wie im folgenden dargestellt durchgeführt werden:

⇒ *kommando* **>** *ausgabedatei*

lenkt die Ausgabe des Kommandos um in eine Datei oder auf ein Ausgabegerät (z.B. Drucker, Terminal).

Achtung:
Erfolgt die Ausgabe in eine Datei, so wird - falls die Datei bereits existiert - der bisherige Dateiinhalt überschrieben!

⇒ *kommando* **>>** *ausgabedatei*

lenkt die Ausgabe um in eine Datei, wobei der bisherige Dateiinhalt jedoch nicht überschrieben wird, sondern die neue Ausgabe an den bisherigen Dateiinhalt angehängt wird.

⇒ *kommando* **2>** *ausgabedatei*

lenkt die Ausgabe der Fehlermeldungen des Kommandos um in eine Datei oder auf ein Ausgabegerät (z.B. Drucker, Terminal).

Achtung:
Erfolgt die Ausgabe in eine Datei, so wird - falls die Datei bereits existiert - der bisherige Dateiinhalt überschrieben!

⇒ *kommando* **2>>** *ausgabedatei*

lenkt die Ausgabe der Fehlermeldungen um in eine Datei, wobei der bisherige Dateiinhalt jedoch nicht überschrieben wird, sondern die neue Ausgabe an den bisherigen Dateiinhalt angehängt wird.

⇒ *kommando* **>&** *ausgabedatei*

lenkt die Ausgabe des Kommandos **und** die Ausgabe der Fehlermeldungen des Kommandos um in **eine** Datei oder auf ein Ausgabegerät (z.B. Drucker, Terminal).

Achtung:
Erfolgt die Ausgabe in eine Datei, so wird - falls die Datei bereits existiert - der bisherige Dateiinhalt überschrieben!

⇒ *kommando* **>>&** *ausgabedatei*

lenkt die Ausgabe des Kommandos **und** die Ausgabe der Fehlermeldungen um in eine Datei, wobei der bisherige Dateiinhalt jedoch nicht überschrieben wird, sondern die neue Ausgabe an den bisherigen Dateiinhalt angehängt wird.

⇒ *kommando* **<** *eingabedatei*

übergibt dem Kommando die Eingabe aus einer Datei.

Hinweise:

- Soll die **Ausgabe von Meldungen oder Fehlermeldungen unterdrückt** werden, so muß eine **Umleitung auf das Pseudo-Gerät /dev/null** erfolgen.
- Die **Ausgabe von zwei aufeinander folgenden Kommandos** kann mit **(***kommando_1* **;** *kommando_2***) >** *ausgabedatei* in **eine** Datei umgeleitet werden.

Anstelle einer Ausgabe-/Eingabeumlenkung kann in UNIX auch eine **Weiterleitung der Ausgabe eines Kommandos als Eingabe eines folgenden Kommandos** erfolgen. Diese Weiterleitung wird als **Pipelining** bezeichnet, Programme, die in dieser Weise arbeiten, als **Filter**. Zur Realisierung dieses Konzepts werden die **zu verarbeitenden Daten in einem Puffer, der Pipe, zwischengespeichert**. Beim Pipelining erfolgt die **Abarbeitung der Daten nach dem FIFO-Prinzip (= first in, first out)**, d.h. die Daten werden in der Reihenfolge, in der der erste Prozeß sie in den Puffer schreibt, von dem zweiten Prozeß aus dem Puffer gelesen. Als **Symbol (Pipe-Zeichen)** wird **zwischen zwei Befehle** das Zeichen **|** gesetzt. Es ist auch möglich, mehr als zwei Befehle durch Pipe-Zeichen miteinander zu verbinden.

Beispiel für eine Pipe:

ls / **| less**

listet den Inhalt des Wurzelverzeichnisses mit Hilfe einer Pipe seitenweise, d.h. die Ausgabe des ls-Kommandos wird als Eingabe für das Kommando less benutzt. Das Kommando less gibt dann das Inhaltsverzeichnis seitenweise aus.

Ein **wichtiges Kommando im Zusammenhang mit dem Pipe-Konzept** ist das Kommando **tee**. Mit Hilfe des tee-Kommandos erfolgt **eine Ausgabe auf die Standardausgabe** (Bildschirm) **und** es wird eine **Kopie der Ausgabe in eine Datei** erstellt.

Hinweis:
Mit der **Option -a** wird **die Ausgabe an eine bereits existierende Datei angehängt**.

Beispiel:

ls / **| tee** *wverzeichnis*

listet den Inhalt des Wurzelverzeichnisses auf dem Bildschirm und schreibt dieses Listing in die Datei *wverzeichnis*.

Aufgaben:

Hinweise:
- Führen Sie die folgenden Aufgaben in Ihrem Home-Directory aus!
- Zur Durchführung einiger der folgenden Aufgaben werden Informationen zu den Kommandos **sort** und **wc** benötigt:

⇒ **sort** *datei*
gibt den Inhalt der angegebenen Datei zeilenweise sortiert auf dem Bildschirm aus.

Hinweis: Mit der **Option -r** erfolgt eine absteigende Sortierung!

⇒ **wc -l** *datei*
zählt die Zeilen der angegebenen Datei.

1.
Erstellen Sie mit dem Befehl cat eine Textdatei *text37* mit dem Inhalt:
Dies ist die erste Zeile der Textdatei text37.

2.
Ergänzen Sie die obige Datei *text37* mit Hilfe des cat-Befehles um die Textzeile:
Dies ist die zweite Zeile der Textdatei text37.

3.
Schreiben Sie die Inhaltsverzeichnisse des Wurzelverzeichnisses und aller Unterverzeichnisse in eine Datei *verzeichnis*.

Hinweise:
- Das Kommando soll **im Hintergrund abgearbeitet** werden.
- **Eventuell auftretende Fehlermeldungen** sollen **in eine Datei** *rfehler* geschrieben werden.

4.
Kontrollieren Sie, ob Aufgabe 3 erfolgreich durchgeführt wurde.

5.
Schreiben Sie die Liste der momentan aktiven Benutzer aufsteigend sortiert in eine Datei *uaktiv*.

6.
Ergänzen Sie die obige Datei um die **Anzahl** der aktiven Benutzer.

7.
Schicken Sie mit Hilfe des cat-Befehles einem anderen angemeldeten Benutzer eine kleine Mitteilung.

Hinweise:
- Mit dem Befehl **who** kann festgestellt werden, welche anderen Terminalleitungen benutzt werden.
- Ein Terminal wird mit /dev/*terminalbezeichnung* angesprochen.

Zusatzfrage:
Unter welcher Bedingung kann Aufgabe 7 nur durchgeführt werden?

8.
Listen Sie die Befehle (= Dateien) im Verzeichnis */bin* seitenweise auf dem Bildschirm und schreiben Sie die Befehlsliste in eine Datei *bliste*.

9.
Listen Sie die Befehle (= Dateien) im Verzeichnis */usr/bin* seitenweise auf dem Bildschirm und fügen Sie die Befehlsliste in die in Aufgabe 8 erstellte Datei *bliste* hinzu.

10.
Listen Sie die in der Datei *bliste* gespeicherte Befehlsliste alphabetisch sortiert auf dem Bildschirm und speichern Sie die sortierte Befehlsliste in der Datei *sortbliste*.

Prozeßverwaltung

Alle Aktionen des Betriebssystems UNIX - z.B. Ausgabe von Daten, Kopieren von Daten - erfolgen in Form von Prozessen**. Ein Prozeß ist ein in der Ausführung befindliches Programm mit seiner Ausführungsumgebung!** Ein solcher Prozeß bildet eine abgeschlossene Einheit und wird vom Betriebssystem selbständig überwacht. **Ein Prozeß kann weitere Prozesse starten**. In diesem Fall werden der **startende Prozeß** als **Vaterprozeß** und die **gestarteten Prozesse** als **Sohnprozesse** dieses Startprozesses bezeichnet. Ein Sohnprozeß kann wiederum weitere Prozesse starten. Aus dieser Vater-Sohn-Prozeßbeziehung ist ersichtlich, daß Prozesse hierarchisch strukturiert sind. Jeder Prozeß in UNIX - mit Ausnahme des Ursprungsprozesses **init** - besitzt also einen Vaterprozeß, wobei ein Vaterprozeß durchaus mehrere Sohnprozesse besitzen kann. Ein Vaterprozeß bleibt normalerweise mindestens so lange aktiv, wie seine Sohnprozesse aktiv sind. **Jeder existierende Prozeß wird in UNIX durch eine Prozeßnummer (PID = process identification number) eindeutig identifiziert.** Sohnprozesse haben zusätzlich auch die Nummer ihres Vaterprozesses - die PPID (= parent process identification number) - gespeichert.

Jedem angemeldeten Benutzer ist mindestens ein Prozeß - normalerweise die Shell - zugeordnet!!

⇒ Das **Kommando**
pstree
zeigt die hierarchische Abhängigkeit der Prozesse!

Hinweise:

- Der **Prozeß init** ist der erste laufende Prozeß. Er wird durch Eintragungen in der Datei */etc/inittab* gesteuert. In dieser Datei ist u.a. der **Runlevel** vorgegeben, in dem der Rechner gestartet werden soll.
  SuSE unterscheidet die folgenden **Runlevel**:
  0 Halt
  S Single User mit US-Tastaturbelegung
  1 Single User
  2 Multi-User ohne Remote-Netzwerk
  3 Multi-User mit Remote-Netzwerk
  5 Multi-User mit Netzwerk und KDM
  6 Systemneustart (reboot)

  ⇒ Mit dem **Kommando**
  init *runlevel*
  kann **im laufenden Betrieb der Runlevel des Rechners geändert** werden!
- Die **Verwaltungsinformationen von Prozessen** sind **im Verzeichnis** */proc* (= **virtuelles Dateisystem**!) abgebildet. In diesem Verzeichnis existiert für jeden Prozeß ein eigenes Unterverzeichnis, das als Name die PID des betreffenden Prozesses erhält! Das Verzeichnis */proc* beansprucht **jedoch keinen Speicherplatz auf der Festplatte**, es ist nur ein Abbild betriebssysteminterner Daten!

* Die Prozeßumgebung

Zu jedem Prozeß gehört eine sogenannte Prozeßumgebung. Zu dieser Umgebung gehören **u.a.**:

- CPU-Zeit, Hauptspeicher, Festplattenplatz;
- offene Dateien;
- das aktuelle Directory, aus dem der Prozeß gestartet wurde;
- die für den Prozeß sichtbaren Umgebungsvariablen (z.B. HOME, PATH).

Ein Sohnprozeß erbt immer die Umgebung des Vaterprozesses, die jedoch verändert werden kann!

---

* Der Scheduler

Da UNIX ein Mehrbenutzersystem ist, **konkurrieren zu jedem Zeitpunkt mehrere Prozesse um die Ausführung durch die CPU**. Da der **Prozessor jedoch zu einem bestimmten Zeitpunkt nur einen Prozeß bearbeiten** kann, ist ein **Programm erforderlich**, **das die quasigleichzeitige Abarbeitung von mehreren Prozessen nach dem Time-Sharing-Verfahren realisiert - der Scheduler**. Der Scheduler **verwaltet** die sogenannte **Prozeßtabelle**, in der alle existierenden Prozesse eingetragen sind, und stellt die CPU wechselweise den konkurrierenden Prozessen zur Verfügung.

⇒ Das **Kommando**
top
zeigt **welcher Prozeß wieviel Rechenzeit beansprucht**.

Hinweis:
Das Kommando **wird nach dem Start endlos ausgeführt**, wobei standardmäßig nach jeweils 5 Sekunden eine Aktualisierung der angezeigten Informationen erfolgt. => Durch **Eingabe** des Buchstabens *q* kann das **Kommando beendet** werden.

---

* Die Prozeßpriorität

Zur **Regelung der Abarbeitungs-Reihenfolge** der einzelnen konkurrierenden Prozesse erhält **jeder Prozeß in UNIX eine Priorität**. Dabei erhalten Prozesse, die sich im Systemmodus befinden (dies sind Prozesse, die eine Systemfunktion aufgerufen haben, die noch nicht beendet ist), eine höhere Priorität als solche im Benutzermodus. Um jedoch **eine einseitige Vergabe der CPU-Zeit an Prozesse mit hoher Priorität zu vermeiden, wird die Priorität jedes Prozesses in gewissen Zeitintervallen neu berechnet**. In diese Berechnung gehen die Größe des Prozesses, die Zeit, die er bereits aktiv war, und die Zeit, die er verdrängt wurde, ein. **Durch die ständige Aktualisierung der Priorität hat ein Prozeß zwei verschiedene Prioritäten**: die **Grundpriorität**, die er **beim Start** erhält, und **die aktuelle Priorität, die er zur Ausführungszeit gerade besitzt**.

Ein **Benutzer kann bei Abgabe eines Kommandos die Grundpriorität** (Standardwert 20) **des Prozesses** zwischen 20 und 39 variieren (**herabsetzen**!) durch Voranstellen des Kommandos **nice**. Dabei ist **zu beachten**, daß **ein höherer Prioritätswert eine niedrigere Priorität bedeutet**!
Der **Systemverwalter** - **als einziger** Benutzer! - kann die Priorität eines Prozesses auch **erhöhen!**

Beispielsweise würde

⇒durch den Befehl
nice -5 *kommando*
die Priorität des angegebenen Kommandos auf 25 reduziert.

⇒durch den Befehl
nice --5 *kommando*
die Priorität des angegebenen Kommandos auf 15 erhöht!

⇒Eine **Prozeßpriorität** kann während der Prozeßabarbeitung **geändert** werden mit dem Kommando
renice *priorität PID*

---

* Hauptspeicherverwaltung

In UNIX erfolgt die **Speicherverwaltung virtuell**. Dabei wird - bei INTEL-Prozessoren - **der zur Verfügung stehende Speicher in Speicherseiten (Pages) zu je 4 KB unterteilt**, die jeweils einem Prozeß zugeordnet werden. Es werden jedoch nur diejenigen Speicherseiten einem Prozeß zur Verfügung gestellt, die er wirklich braucht. **Nicht benötigte Speicherseiten** werden bei **Hauptspeicherknappheit auf die Festplatte in einen speziellen Auslagerungsbereich (paging area) nach der LRU-Regel (Least Recently Used) ausgelagert** und **nur wieder eingelagert, wenn sie benötigt** werden (**demand paging**)!

---

* Informationen über existierende Prozesse

Informationen über existierende Prozesse können mit dem Kommando **ps** angezeigt werden.

Das Kommando **ps ohne Optionen** zeigt die folgenden Informationen zu Prozessen des betreffenden Benutzers:

- **PID**
  = Prozeßidentifikationsnummer.
- **TTY**
  = kontrollierendes Terminal des Prozesses.
  Dies ist das Terminal, das als erstes von dem Prozeß zum Lesen oder Schreiben geöffnet wurde.
  Hinweis:
  Ist bei TT ein **?** eingetragen, so ist dieser Prozeß ein Prozeß, der keinem Terminal zugeordnet ist. Ein solcher Prozeß wird als **Dämon-Prozeß** bezeichnet. Dämon-Prozesse sind Pro-

zesse, die im Hintergrund zyklisch ablaufen, d.h. sie werden in bestimmten Zeitintervallen aktiv, **z.B.** der **cron-Prozeß**!

- **STAT**
  = Prozeßstatus
  - ♦ **R** = (runnable) ausführbar
  - ♦ **S** = (sleeping) stillgelegt
  - ♦ **D** = (uninterruptible sleeping)
    nicht unterbrechbar stillgelegt
  - ♦ **T** = (stopped/traced)
    angehalten/im Einzelschrittmodus ausgeführt
  - ♦ **Z** = Zombie '<defunct>'
    = Prozeß, der beendet wurde, dessen Elternprozeß dieses Ende jedoch noch nicht realisiert hat.
  - ♦ **W** = der Prozeß hat keine residenten Seiten
- **TIME**
  = Gesamtzeit, die der Prozeß bis jetzt verbraucht hat.
- **COMMAND**
  = Kommandoname.

U.a. mögliche Optionen für das Kommando **ps:**

a	zeigt die obigen Informationen für alle Prozesse.
x	zeigt auch die Dämon-Prozesse (siehe oben!).
l	'ausführliches Listing', zeigt u.a. auch die UID, die PPID und die PRI(orität).
r	zeigt nur laufende Prozesse.
f	zeigt die Prozeßhierarchie.

Anmerkung:
Zwei wichtige Dämonen sind die **Log-Dämonen** *klogd* und *syslogd*, **die Kernel- und Systemmeldungen in Log-Dateien protokollieren**. Diese **Log-Dateien müssen jedoch in gewissen Zeitabständen gelöscht werden** (automatisch über *crontab* (siehe S. 45) oder bei jedem Systemstart bzw. manuell), da ihre Größe ständig zunimmt!

Beispielausgabe für das Kommando **ps –alx**:

F	**UID**	**PID**	**PPID**	**PRI**	**NI**	VSZ	RSS	WCHAN	**STAT**	**TTY**	**TIME**	COMMAND
100	0	1	0	0	0	344	180	do_sel	S	?	0:04	init [2]
040	0	2	1	0	0	0	0	bdflus	SW	?	0:00	[kflushd]
040	0	3	1	0	0	0	0	kupdat	SW	?	0:00	[kupdate]
040	0	4	1	0	0	0	0	kpiod	SW	?	0:00	[kpiod]

Priorität des Prozesses
Nicefaktor

---

* Hintergrundprozesse

In UNIX besteht die Möglichkeit, **Prozesse, die über eine längere Zeit ablaufen und damit ein Terminal für die weitere Arbeit blockieren, im Hintergrund ablaufen** zu lassen, so daß das **Terminal direkt nach Absenden des Befehles für andere Arbeiten**

zur Verfügung steht. Dabei sollte jedoch **beachtet** werden, daß das für den Ablauf **im Hintergrund abgegebene Kommando keine Ausgaben auf dem Bildschirm erzeugt!!**, da diese Ausgaben z.B. beim Editieren eines Textes plötzlich störend auf dem Bildschirm erscheinen, und **keine Eingaben erwartet!! (=> Ein-/Ausgabeumlenkung** S. 35**!!)**

Ein Prozeß wird in den Hintergrund gestellt durch Anfügen des Zeichen & an das entsprechende Kommando!

Hinweis:
Nach Absenden des Befehles erscheint **auf dem Bildschirm**
- eine (fortlaufende) **Jobnummer**
- die **Prozeßnummer (PID)**

des nun im Hintergrund ablaufenden Prozesses, z.B. **[2] 4711**

⇒ **Im Hintergrund laufende Prozesse** können mit Jobnummer und Status **gelistet** werden mit dem Kommando

jobs

Hinweis:
Durch die **Option -l** wird zusätzlich die PID angezeigt!!

⇒ **Im Hintergrund laufende Prozesse** können **angehalten** werden mit dem Kommando

kill -STOP %*jobnummer* oder
kill -STOP *PID*

fortgesetzt werden mit dem Kommando

kill -CONT %*jobnummer* oder
kill -CONT *PID*

⇒ **Merke:**

Normalerweise wird ein **Hintergrundprozeß abgebrochen**, **wenn sein Vaterprozeß stirbt oder abgebrochen** wird. Dies bedeutet, daß **alle Hintergrundprozesse eines Benutzers beendet** werden**, wenn dieser sich abmeldet**, da in diesem Fall seine Shell (= Vaterprozeß) beendet wird!

Soll ein **Hintergrundprozeß auch nach Abmeldung des Benutzers weiterlaufen**, so muß das Kommando mit

nohup *kommando* **&**

gestartet werden!

⇒**Wechsel** zwischen **Vordergrund- und Hintergrundverarbeitung**

Vordergrundprozeß in den Hintergrund stellen:
-> Anhalten des Vordergrundprozesses mit <STRG>+<z>
-> **bg** *%jobnummer*

Hintergrundprozeß in den Vordergrund holen:
-> **fg** *%jobnummer*

---

* Prozeßabbruch

Bei der Durchführung eines Prozeßabbruches müssen **folgende zwei Situationen unterschieden** werden:

1. **Der Prozeß läuft im Vordergrund am aktuellen Terminal:**
   (meist) <Strg> + <c>

2. **Der Prozeß läuft im Hintergrund oder an einem anderen Terminal:**
   kill -9 *PID*
   Anmerkung:
   Die Signal-Nummer 9 bewirkt, daß der Prozeß auf jeden Fall abgebrochen wird!

Merke:
Jeder "normale" Benutzer kann nur seine eigenen Prozesse killen!!

Aufgabe:

1.
Suchen Sie ab dem Wurzelverzeichnis alle Dateien des Benutzers *root*.
Beachten Sie folgende Hinweise:
- Das Kommando soll im Hintergrund abgearbeitet werden. =>
  - Ausgaben sollen in der Datei *rootliste* gespeichert werden.
  - Fehlermeldungen sollen in der Datei *rootfehler* gespeichert werden.
- Das Kommando soll mit einer um 5 verringerten Priorität gestartet werden.

2.
Kontrollieren Sie die Ausführung des in Aufgabe 1 gestarteten Kommandos mit Hilfe von zwei! unterschiedlichen Kommandos.

3.
Ändern Sie die Priorität des in Aufgabe 1 gestarteten Kommandos auf 8.

4.
Kontrollieren Sie die Änderung der Priorität.

5.
Nehmen Sie das in Aufgabe 1 gestartete Kommando in den Vordergrund.

6.
Stellen Sie das in Aufgabe 1 gestartete Kommando zurück in den Hintergrund.

---

* Ablaufsteuerung von Prozessen mit **at, batch** und **crontab**

Mit den genannten drei Kommandos **können Prozesse zu bestimmten Zeitpunkten gestartet** werden. Dabei unterscheiden sich die drei Kommandos wie folgt:

- **at**
  führt ein Programm **einmal zum angegebenen Zeitpunkt** aus.
- **batch**
  führt ein Programm aus, **wenn** das **System weniger belastet** ist, d.h. bei diesem Befehl bestimmt nicht der Benutzer den Startzeitpunkt des Programmes, sondern das System!
- **cron**
  führt ein Programm **wiederholt (periodisch) zum angegebenen Zeitpunkt** aus.

Merke:
Für alle drei Befehle werden **standardmäßig Ausgabe** und **Fehlermeldungen in die Mailbox des betreffenden Benutzers gespeichert**!! Ist dies vom Benutzer **nicht gewünscht**, so **muß** eine **Umleitung der Ausgaben/Fehlermeldungen erfolgen**!!

---

* Der Befehl **at**

Die Kommandofolge, die der Befehl **at** zu einem angegebenen Zeitpunkt ausführen soll, kann dem Befehl auf zwei verschiedene Arten zur Verfügung gestellt werden.

1. Eingabe der einzelnen Kommandos **über die Tastatur**:

at *zeit* <Enter>
  erstes Kommando <Enter>
   .
   .
  letztes Kommando <Enter>
<STRG> + <d>

2. Eingabe aus einer Datei:

at *zeit* **-f** *kommandodatei*

In diesem 2. Fall müssen die auszuführenden Kommandos zuvor in der Datei *kommandodatei* gespeichert worden sein.

Als **Angabe für** *zeit* kann eingesetzt werden:

HH:MM DD.MM.YY

Besonderheiten:

Uhrzeit: noon = 12 Uhr; midnight = Null Uhr, now = sofort

Datum: today = heute; tomorrow = morgen

Inkrement: +*zahl* minutes/hours/days/weeks/months/years

---

* Der Befehl **batch**

Die Kommandofolge, die der Befehl **batch** ausführen soll, kann dem Befehl auf zwei verschiedene Arten zur Verfügung gestellt werden.

1. Eingabe der einzelnen Kommandos **über die Tastatur**:

batch <Enter>
  erstes Kommando <Enter>
   .
   .
  letztes Kommando <Enter>
<STRG> + <d>

2. Eingabe aus einer Datei:

batch -f *kommandodatei*

In diesem 2. Fall müssen die auszuführenden Kommandos zuvor in der Datei *kommandodatei* gespeichert worden sein.

---

* **Listing** und **Löschen** von **at**- und **batch-Jobs**

Nach dem Absenden des **at-/batch**-Kommandos zeigt das System eine **Job-Nummer, die diesem Auftrag zugeordnet** wurde.

at -l **zeigt** alle zur Zeit existierenden **Jobs** des betreffenden Benutzers

Ausgabebeispiel für das Kommando **at -l**:

```
277     2002-08-24  12:00       a           ukp12
278     2002-08-23  23:30       a           ukp12
281     2002-08-23  21:40       b           ukp12
```

Jobnr	**Datum der Ausführung bei der Abgabe bei**	**Uhrzeit**	**Jobtyp at-Job batch-Job**	**Absender/Besitzer des Jobs**

atrm *jobnummer* **löscht** den angegebenen Auftrag

Anmerkung:
Festlegungen zu den Kommandos **at** und **batch** sind enthalten in:

- **/etc/at.allow**

enthält alle Benutzer, die **at** und **batch** ausführen dürfen.

- **/etc/at.deny**

enthält alle Benutzer, die **at** und **batch** nicht ausführen dürfen.

---

* Der Befehl **crontab**

Mit diesem Befehl kann eine Kommandodatei in dem Verzeichnis */var/cron/tabs* gespeichert werden, deren einzelne Kommandos von dem Prozess **cron** - der als **Hintergrundprozess im System** läuft - in den Zeitabständen ausgeführt werden, die zu dem jeweiligen Kommando angegeben sind.
Merke: Jeder Benutzer kann nur eine Kommandodatei speichern!!

Die Kommandodatei kann dem Befehl **crontab** auf zwei verschiedene Arten zur Verfügung gestellt werden.

1. Eingabe der einzelnen Kommandos **über die Tastatur**:

```
crontab -e
  erstes Kommando <Enter>
   .
   .
  letztes Kommando <Enter>
<STRG> + <d>
```

2. Kopie einer Datei:

crontab *kommandodatei*

In diesem 2. Fall müssen die auszuführenden Kommandos zuvor in der Datei *kommandodatei* gespeichert worden sein.

Eine Zeile der Crontab-Datei, d.h. ein auszuführendes Kommando, muss die folgenden 6 Spalten (jeweils mit einem Leerzeichen getrennt!) enthalten:

Minute	**Stunde**	**Tag**	**Monat**	**Wochentag**	**Kommando**
0-59	0-23	1-31	1-12	0-6, 0=Sonntag	

Als **Vorgaben für die ersten fünf Spalten** gibt es folgende **Möglichkeiten**:

- eine **einzelne Zahl** aus dem angegebenen Bereich.
- eine **Bereichsangabe**, z.B. 1-3 in der vierten Spalte bedeutet, dass das Kommando in den Monaten Januar, Februar und März ausgeführt wird.
- eine **Aufzählung** von Werten, z.B. 4,7,20 in der dritten Spalte bedeutet, dass das Kommando am 4., 7. und 20. Tag ausgeführt wird.
- das **Zeichen *** in einer Spalte bedeutet, dass das Kommando für alle möglichen Werte dieser Spalte ausgeführt wird, z.B. * in

der ersten Spalte bedeutet, dass das Kommando jede Minute ausgeführt wird.

wobei jedoch **alle fünf Spalten gleichzeitig berücksichtigt** werden**!!**

crontab -l	**zeigt den Inhalt** der Crontab-Datei des betreffenden Benutzers
crontab -r	**löscht** die Crontab-Datei des betreffenden Benutzers

Anmerkung:

Festlegungen zu dem Kommando **cron** sind enthalten in:

- **/var/spool/cron/allow**

enthält alle Benutzer, die **crontab** ausführen dürfen.

- **/var/spool/cron/deny**

enthält alle Benutzer, die **crontab** nicht ausführen dürfen.

- **/var/spool/cron/tabs**
  Verzeichnis, das die **Crontab-Dateien der einzelnen Benutzer** enthält.

Aufgaben:

1.
Legen Sie einen Job an, der in einer Datei *jan13* speichert, welche und wieviele Benutzer am 13. Januar um 11.15 Uhr im System aktiv sind.

2.
Legen Sie einen Job an, der heute um 11.30 Uhr in Ihrem Home-Directory ein Verzeichnis *attest* anlegt.

3.
Erzeugen Sie eine Datei *sumail* mit einem Text für den Super-User in Ihrem Home-Directory.

Geben Sie ein Kommando ab, so daß diese Text-Datei *sumail* zu einem Zeitpunkt geringer Systemauslastung als Mail an den Super-User gesendet wird.

4.
Lassen Sie sich Ihre zur Zeit existierenden Jobs anzeigen.

5.

Legen Sie eine Datei *testc* an, in der die folgenden Befehle abgespeichert sind:

⇒ An jedem Freitag soll um 12.00 Uhr und um 12.30 Uhr in einer Datei *benutzer* - gespeichert in Ihrem Home-Directory! - festgehalten werden, welche Benutzer aktuell im System aktiv sind, wobei nach Ablauf des Befehls die Benutzeraktivität der beiden Zeitpunkte in dieser Datei *benutzer* gespeichert sein soll.

⇒ An jedem Donnerstag jeder Woche soll um 10.00 Uhr in einer Datei *dateizahl* - gespeichert in Ihrem Home-Directory! - festgehalten werden, welche und wieviele Dateien in Ihrem Home-Directory abgespeichert sind.

6.

Speichern Sie die in Aufgabe 5. angelegte Datei *testc* als Ihre Crontab-Datei.

Drucken

Da UNIX ein Multi-User-Betriebssystem ist, kann der Fall auftreten, daß mehrere Benutzer gleichzeitig auf einen Drucker zum Ausdruck von Daten zugreifen. **Zur Verwaltung dieser gleichzeitigen Zugriffe** werden **Druckaufträge (= Jobs)** - die vom Anwender über das Kommando *lpr* abgegeben werden - **in eine Warteschlange (= Queue) für den angesprochenen Drucker eingereiht.** Einstellen in die Warteschlange bedeutet standardmäßig, daß eine Kopie der zu druckenden Datei in ein Verzeichnis, den sogenannten Spool-Bereich, geschrieben wird. Dabei wird für jeden verfügbaren Drucker mindestens ein eigener Spool-Bereich angelegt, der ein Unterverzeichnis des Verzeichnisses */usr/spool/lpd* ist. Ein **Hintergrund-Prozeß**, der **Druck-Dämon** *lpd* **überprüft in regelmäßigen Abständen diese Spool-Bereiche und sorgt für den Druck von Druckaufträgen aus der Warteschlange auf dem angegebenen Drucker**. Die Abarbeitung der Druckwarteschlange erfolgt dabei **nach dem FIFO-Prinzip**!

Merke:
Für **einen physikalischen Drucker** (= Gerät**) können mehrere Druck-Warteschlangen** mit unterschiedlichen Eigenschaften **definiert** werden. Die erforderlichen **Festlegungen** sind in der **Datei** */etc/printcap* gespeichert! Diese Datei wird von dem Druck-Dämon *lpd* ausgelesen, um festzustellen, welche Drucker im Netzwerk und lokal zur Verfügung stehen!

Beispielhaft könnte die Datei */etc/printcap* zur Definition eines lokalen Druckers wie folgt aufgebaut sein:

```
hplj1|lp|HP_Laserjet_4:\
    :sh:sd=/var/spool/lpd/hplj1:\
    :lp=/dev/lp0:\
    :if=/usr/local/lib/filter:
```

Erläuterungen:

⇒ Jede Zeile - außer der letzten - wird mit dem Zeichen \ abgeschlossen.

⇒ Einzelne Festlegungen werden mit dem Zeichen : begonnen, beendet und getrennt.

⇒ **Zeile 1:**
In der ersten Zeile, die auf Spalte 1 beginnt, werden **die Namen der verschiedenen Druckwarteschlangen**, die auf dieselbe parallele Schnittstelle (= denselben Drucker) ausgegeben werden sollen, mit dem **Trennzeichen |** angegeben.
Dabei bedeutet der **Name** *lp* in der Liste, daß **dieser Drucker** als **Standard-Drucker** bei Kommandos verwendet werden soll, in denen keine Druckwarteschlange angegeben ist!!

Merke:
Mindestens ein Drucker in der Datei */etc/printcap* sollte den Namen *lp* erhalten!!

⇒ **Zeile 2:**
sh (= suppress header) **unterdrückt die Header-Seite**, die Benutzername, Rechnername und Job-Name ausgibt.
sd legt das Spool-Verzeichnis fest. Dieses sollte ein Unterverzeichnis des Verzeichnisses */var/spool/lpd* sein.

⇒ **Zeile 3:**
lp gibt die parallele Schnittstelle für den Ausdruck an.

⇒ **Zeile 4:**
if legt den Filter für den Ausdruck fest.

Anmerkungen:

- Filterprogramme haben die Aufgabe, die Datendatei eines Jobs in das druckerspezifische Format zu überführen (Druckertreiber!). Unter SuSE-Linux steht standardmäßig als Filter das Programm **apsfilter** zur Verfügung (Aufruf über Yast zur Einrichtung lokaler Drucker! oder */var/lib/apsfilter/SETUP* (ermöglicht auch die Einrichtung von 'entfernten' Warteschlangen!!).
- Als Standard-Druckformat wird unter UNIX PostScript verwendet! Zur Überführung von PostScript-Dateien in druckerspezifische Formate von Nicht-PostScript-Druckern wird von **apsfilter** das Programm **Ghostscript** eingesetzt.

∗ **Kommandos** für den **"normalen" Benutzer:**

⇒ **Abgeben eines Druckauftrages** für Datei(en) auf den **Standard-Drucker** (z.B. festgelegt über die Variable PRINTER) bzw. [auf den angegebenen Drucker (Druckwarteschlange!)]

lpr *-optionen* [**-P***drucker*] *datei(en)*

U.a. mögliche Optionen:

m	sendet eine Mail nach Abarbeitung des Druckauftrages in die Mailbox des Benutzers.
h	unterdrückt den Ausdruck der Header-Seite.
#anzahl	druckt den Druckauftrag mit *anzahl* Kopien.

⇒**Anzeige** der **Druckaufträge** mit **Job-Id** und Priorität **in der Warteschlange** [auf dem angegebenen Drucker (Druckwarteschlange!)]

lpq [**-P***drucker*]

Ausgabebeispiel für das Kommando **lpq**:

lp is ready and printing

Rank	**Owner**	**Job**	**Files**	**TotalSize**
active	ukp12	104	testc	119bytes
1st	ukp12	117	uaktiv	419 bytes
2nd	root	118	ende	18bytes
3rd	ukp22	119	vliste	332 bytes

Position in der Warteschlange
Besitzer des Druckjobs
Jobnummer
Dateiname
Dateigröße

⇒**Löschen eines Druckauftrages** [auf dem angegebenen Drucker (Druckwarteschlange!)]

lprm [**-P***drucker*] *job-id*

Löschen aller Druckaufträge [auf dem angegebenen Drucker (Druckwarteschlange!)]

lprm [**-P***drucker*] **-**

Merke:
Ein **Benutzer** kann **nur seine eigenen Druckaufträge löschen!!**

∗ **lpc** - Das **Kommando** für den **Super-User:**

Das Kommando *lpc* steht dem Super-User mit verschiedenen Optionen zur Steuerung der Druckwarteschlangen zur Verfügung.

Hinweis für alle Kommandos:
Für den Namen der Druckwarteschlange kann auch **all** (= alle Druckwarteschlangen) - **außer bei status!!** - gesetzt werden!!

⇒**Anzeige** des Druckstatus aller [einer] Druckwarteschlange(n)

lpc status [*druckwarteschlange*]

Hinweis:
Dieses Kommando kann auch von einem "normalen" Benutzer verwendet werden!

Ausgabebeispiel für das Kommando **lpc status**:

lp: **Name der Warteschlange**

queuing is **enabled**	**Warteschlange ist aktiv und nimmt Druckaufträge an**
printing is **disabled**	**Drucker ist nicht aktiv, d.h. es werden keine Druckaufträge aus der Schlange gedruckt**
13 **entries in spool area**	**Anzahl Einträge in der Schlange**
lp is **ready and printing**	**Druck-Dämon ist gestartet**

⇒**Deaktivieren** einer Druckwarteschlange
= stoppt die Aufnahme neuer Druckaufträge in die Warteschlange
lpc disable *druckwarteschlange*

⇒**Aktivieren** einer Druckwarteschlange
lpc enable *druckwarteschlange*

⇒**Stoppen** des Ausdrucks aus einer Druckwarteschlange
lpc stop *druckwarteschlange*

Hinweis:
Der **gerade im Druck befindliche Job wird noch beendet**!

⇒**Starten** des Ausdrucks aus einer Druckwarteschlange
lpc start *druckwarteschlange*

⇒**Deaktivieren und Stoppen** des Ausdrucks einer Druckwarteschlange
lpc down *druckwarteschlange*

⇒**Aktivieren und Starten** des Ausdrucks einer Druckwarteschlange
lpc up *druckwarteschlange*

⇒**Deaktivieren und Stoppen** des Ausdrucks einer Druckwarteschlange
lpc abort *druckwarteschlange*

Hinweis:
Der **gerade im Druck befindliche Job** wird **nicht beendet, sondern sofort abgebrochen**!

⇒**Druckauftrag an die Spitze** einer Druckwarteschlange setzen
lpc topq *druckwarteschlange job-id*

Datensicherung auf Band oder Diskette

Zwei Kommandos, die zur Datensicherung auf jedem UNIX-System zur Verfügung stehen, sind die Kommandos **tar** und **cpio**.

* Das Kommando **tar**

⇒**Sichern** von **Dateien**
tar -option(en) *dateiname(en)*

⇒**Sichern** eines **Verzeichnisses mit Unterverzeichnissen**
tar -option(en) *verzeichnisname*

Wichtige Optionen:

c	legt eine neue Archivdiskette mit dem Verzeichnis/den Dateien an.
t	zeigt den Inhalt einer Archivdiskette bzw. das archivierte Verzeichnis/die archivierten Dateien.
x	schreibt das Verzeichnis/die Dateien zurück auf die Festplatte.
f *name*	erzeugt/liest ein Archiv von *name*, wobei *name* eine Datei oder eine Gerätedatei sein kann.
v	zeigt während der Befehlsausführung die Namen der archivierten Dateien.

Hinweis:
Wird beim Sichern eines Verzeichnisses dessen genauer Name angegeben, so werden die gesicherten Dateien beim Zurückschreiben auf die Festplatte wieder in dieses Verzeichnis zurückgeschrieben. Wird jedoch als Verzeichnisname **.** für aktuelles Arbeitsverzeichnis angegeben, so werden die gesicherten Dateien beim Zurückschreiben in das nun aktuelle Arbeitsverzeichnis geschrieben!!

---

* Das Kommando **cpio**

Das Kommando **cpio** ist unter LINUX zur Datensicherung **eher ungebräuchlich**. Es wird jedoch benötigt bei der Installation von Programmen, die vom Hersteller mit **cpio** auf dem Datenträger gespeichert wurden.

Datenkomprimierung

Die **Komprimierung/Dekomprimierung von einzelnen Dateien** kann mit dem Kommando **gzip** durchgeführt werden. Zur Komprimierung **mehrerer Dateien oder von Verzeichnissen** muss jedoch das Kommando **tar** (siehe oben) eingesetzt werden!

* **Komprimierung/Dekomprimierung** von Dateien

⇒**gzip** *dateiname(en)*

Optionen:

c	lässt die zu komprimierende Datei unverändert und leitet das Ergebnis auf die Standardausgabe um.
d	dekomprimiert die angegebene(n) Datei(en)
r	komprimiert/dekomprimiert auch Dateien in allen Unterverzeichnissen

Hinweis:
Die **komprimierten Dateien** erhalten als **Erweiterung** automatisch **.gz**!

---

* **Dekomprimierung** von Dateien

⇒**gunzip** *dateiname(en)*

Hinweise:

- Das Kommando **gunzip** ist ein **Link auf** das Kommando **gzip –d**
- Mit dem Kommando **gunzip** können Dateien, die **mit gzip** oder **mit compress komprimiert** wurden, **dekomprimiert** werden!

Option:

r	dekomprimiert auch Dateien in allen Unterverzeichnissen

Aufgaben:

1.
Komprimieren Sie alle Dateien in Ihrem Home-Directory, deren Dateiname mit *text* beginnt.

2.
Dekomprimieren Sie die in Aufgabe 1 komprimierten Dateien (2 Lösungen!).

Benutzerkommunikation

• Versenden von Nachrichten:

Nachrichten können mit
⇒**write** *username* an das Terminal eines eingeloggten Benutzers;
⇒**wall** an alle angemeldeten Benutzer (oft nur durch den Super-User ausführbar!);
⇒**mail** *username* in die Mailbox eines (eventuell nicht eingeloggten) Benutzers
versendet werden.

• Das Kommando **write:**

Bei Benutzung des write-Befehls erscheint jede Zeile - sobald sie mit <enter> beendet wird - auf dem eigenen Bildschirm und auf dem Bildschirm des angeschriebenen Benutzers. Der write-Befehl wird am Anfang einer neuen Zeile **mit** der Tastenkombination **<Strg> + <d> beendet**.

Ein Benutzer kann sein Terminal **gegen Nachrichtenempfang schützen** (außer bei Sendungen des Systemadministrators) mit
⇒**mesg n**

Dieser **Schutz** kann **aufgehoben** werden mit
⇒**mesg y**

Der **Schutzstatus** kann **angezeigt** werden mit
⇒**mesg**

Hinweis:
Der **Status eines Terminals** kann mit dem Kommando **who -T festgestellt** werden!!

• Das Kommando **wall:**

Die Ausführung und Arbeitsweise des Kommandos **wall** entspricht der des Kommandos **write** (siehe oben!).

• Das Kommando **mail:**

Standardmäßig wird nach Abgabe des mail-Befehles ein *Subject* abgefragt. Dieses **Subject** erscheint in der Mail-Liste des Empfängers als **Titel der Mail**.

Eine mit dem mail-Befehl geschriebene Nachricht wird **mit . <enter> am Anfang einer neuen Zeile abgeschlossen**.

⇒**Lesen der Mailbox**
Die in der Mailbox eines Benutzers liegende Post wird mit dem **mail**-Befehl gelesen. Nach Abgabe des Befehls erscheint eine Liste der Nachrichten, die sich aktuell in der Mailbox befinden. Diese Nachrichten sind fortlaufend numeriert und können über diese Nummer angesprochen werden. Unter der Nachrichtenliste erscheint das Zeichen **&**, hinter dem u.a. (nur eine kleine Auswahl!!) einer der folgenden Befehle abgegeben werden kann:

-> *Nachrichtennummer* **<enter>**
zeigt die mit *Nachrichtennummer* angesprochene Nachricht

-> **d** *Nachrichtennummer* **<enter>**
löscht die mit *Nachrichtennummer* angesprochene Nachricht

-> **h <enter>**
zeigt erneut die Nachrichtenliste

-> **r** *Nachrichtennummer* **<enter>**
ermöglicht eine Antwort auf die mit *Nachrichtennummer* angesprochene Nachricht

-> **s** *Nachrichtennummer dateiname* **<enter>**
speichert die mit *Nachrichtennummer* angesprochene Nachricht in der Datei *dateiname*
(Hinweis: es können mehrere Nachrichten in dieser Datei *dateiname* abgespeichert werden!!)

-> **q <enter>**
verläßt die Mailbox, löscht die gelesenen Nachrichten aus der Datei mailbox und speichert sie (standardmäßig) in der Datei **/home/***username***/mbox**

-> **x <enter>**
verläßt die Mailbox ohne Änderungen

Anmerkung:
Die in der Datei **mbox** bzw. in der Datei *dateiname* abgelegte Post kann mit
⇒**mail -f mbox** bzw.
⇒**mail -f** *dateiname*
gelistet werden. Nach der Listung erscheint wie nach Abgabe des mail-Befehles das Zeichen &, nach dem ebenfalls die oben aufgeführten Befehle benutzt werden können!

• Dialog mit einem anderen Benutzer

Ein Dialog mit einem anderen - eingeloggten! - Benutzer kann mit dem Kommando **talk** *username* geführt werden.

Ablauf:

⇒Zur **Aufnahme des Dialoges** muß das Kommando **talk** *username* von einem Benutzer abgegeben werden. Der mit *username* angesprochene Benutzer erhält eine Nachricht über den Dialogwunsch auf seinem Terminal. Zur Annahme des Dialogwunsches muß dieser Benutzer ebenfalls das Kommando **talk** *username* mit dem Namen des anderen Benutzers abgeben. Nach diesem gegenseitigen Verbindungsaufbau können Informationen durch Bestätigung mit der Taste <enter> auf das Terminal des jeweiligen Dialogpartners geschickt werden.

⇒**Der Dialog** kann von einem Benutzer durch die Tastenkombination **<Strg> + <c> beendet** werden.

Hinweis:
Ist ein **Terminal gegen Nachrichtenempfang geschützt** (siehe S. 58), kann der betreffende **Benutzer mit** dem Kommando **talk nicht erreicht** werden!!

Shell-Prozeduren (Shell-Skripten)

Shell-Prozeduren bieten die Möglichkeit, mehrere UNIX-Kommandos in einer Datei zu speichern und diese Kommandofolge als Programm über den Dateinamen abzurufen. Zusätzlich zur einfachen Aneinanderreihung von UNIX-Kommandos bieten diese Shell-Prozeduren die Möglichkeit, mit Variablen zu arbeiten sowie die Programmstrukturen Abfrage, Schleife und Fallunterscheidung einzusetzen.

Hinweis:
Kommentarzeilen in Shell-Prozeduren werden **mit** dem Zeichen **# eingeleitet**!

- **Aufruf einer Shell-Prozedur:**

Eine in einer Datei *dateiname* abgelegte Folge von UNIX-Kommandos (= Shell-Prozedur) kann aufgerufen werden mit:

sh *dateiname*

oder

dateiname

falls die Datei zuvor **mit** dem Befehl **chmod** auf **ausführbar** gesetzt wurde!! **Außerdem** muß für die Datei das **Leserecht** gesetzt sein, da die Shell die Datei für die Ausführung lesen muß.

Anmerkung:
Beim Aufruf der Shell-Prozedur können dieser **Prozedur zusätzlich Werte für die aktuelle Ausführung** der Prozedur übergeben werden, falls in der Prozedur die **Positionsparameter $1, $2, .., $9** verwendet wurden. **Diese Werte werden mit Leerzeichen getrennt in der erforderlichen Reihenfolge hinter dem Dateinamen aufgeführt.**
Hinweis:
Der Positionsparameter $0 enthält immer den Namen der betreffenden Shellprozedur selbst!

- **Variablen in einer Shell-Prozedur:**

In einer Shell-Prozedur können als Variablen verwendet werden:

- **selbstdefinierte Variablen**
- **vom System vordefinierte Variablen** (haben einen festen Wert und können nicht verändert werden!)
- **Umgebungsvariablen**
- **Positionsparameter**

➢ **Definition einer Variablen**

variablenname=wert

Anmerkung:
Obige Anweisung muß ohne Leerzeichen geschrieben werden! Sind im Wert Leerzeichen erforderlich, so muß der Wert in "" gesetzt werden!

Hinweise:

- Der Inhalt einer Variablen wird mit **$***variablenname* gelesen.
- Mit dem Kommando **read** *variablenname* kann eine Eingabe für diese Variable über die Tastatur erfolgen.

➢ **Beispiele für vordefinierte Variablen**

$! Prozeßnummer des zuletzt angestoßenen Hintergrundprozesses

$# Anzahl der übergebenen Werte für Positionsparameter

$$ Prozeßnummer des laufenden Prozesses

➢ **Gültigkeitsbereich von Shell-Variablen**

Ohne weitere Angaben gelten die aufgeführten Variablen nur während des Ablaufes der aktuellen Prozedur, d.h. es sind lokale Variablen. Soll eine Variable auch für von dieser Prozedur aufgerufene weitere Shell-Prozeduren (Sohn-Prozesse) Gültigkeit haben, d.h. globale Variable sein, so muß diese Variable mit dem Befehl **export** *variablenname* exportiert werden!

Hinweis:
Die Parameter $0, .., $9 können jedoch nicht exportiert werden!

- **Kommandoersetzung**

Durch Einschließen eines Kommandos in ` wird dieses Kommando durch die Shell ausgeführt und die Ausgabe des Kommandos für den Ausdruck `kommando` eingesetzt. Beispielsweise wird durch die Zeile

users=`**who | wc -l**`

in der selbstdefinierten Variablen users die Anzahl der aktuell eingeloggten Benutzer - also die Ausgabe des Kommandos who | wc -l - gespeichert.

Anmerkung:

Mit dem Kommando *echo* **$***users* kann eine Ausgabe dieser Anzahl erfolgen!

Aufgaben:

1.
Schreiben Sie eine Shell-Prozedur *ip*, die das Inhaltsverzeichnis Ihres Home-Directorys und aller Unterdirectories seitenweise ausgibt.

2.
Schreiben Sie eine Shell-Prozedur *vip*, die das Inhaltsverzeichnis eines beliebigen Directorys und aller Unterdirectories seitenweise ausgibt. Als erste Zeile soll der Name des angezeigten Directorys ausgegeben werden.

3.
Schreiben Sie eine Shell-Prozedur *w*, die Ihnen die folgenden Informationen liefert:
- Aktuelles Tagesdatum
- Alle zur Zeit angemeldeten Benutzer sowie Anzahl der angemeldeten Benutzer

4.
Schreiben Sie eine Shell-Prozedur *umgeb*, die Ihnen die folgenden Informationen (jeweils mit Informationstext) liefert:
- Benutzername
- Home-Directory
- Benutzter Terminaltyp
- Aktuelles Arbeitsverzeichnis

5.
Schreiben Sie eine Shell-Prozedur *f*,
- die es erlaubt, den Befehl **find** ab einem beliebigen Directory in beliebiger Variante im Hintergrund zu benutzen;
- die die Ausgabe des Befehles **find** in eine Datei *ausgabe* in Ihrem Home-Directory schreibt;
- die eventuell auftretende Fehlermeldungen des Befehles find in eine Datei *fehler* in Ihrem Home-Directory schreibt;
- die nach Abgabe des **find**-Befehles die Prozeßnummer ausgibt;
- die nach Ablauf des Befehles die Dateien *ausgabe* und *fehler* seitenweise anzeigt.

Testen Sie die obige Prozedur mit den folgenden Aufgabenstellungen:
- Suchen aller Dateien des Benutzers *root* ab dem Hauptverzeichnis;
- Suchen aller Dateien des Dateityps *d* ab Ihrem aktuellen Arbeitsverzeichnis.

6.
Schreiben Sie eine Shell-Prozedur *m*,
- die eine beliebige Diskette in einem beliebigen Verzeichnis montiert;
- die alle Dateien Ihres Home-Directorys auf die montierte Diskette kopiert;
- die die kopierten Dateien auf der montierten Diskette seitenweise anzeigt;
- die die Diskette wieder demontiert.