Hinweise zum Arbeiten mit UNIX

Labor "Rechnerorganisation"
Berufsakademie Stuttgart
Fachrichtung Nachrichtentechnik

Stand 1. Februar 2007

http://home.ntz.de/phleps/unix/ux_bedien.pdf

Inhaltsverzeichnis

1 UNIX-Konventionen **2**

2 UNIX-Befehle **2**
2.1 Übersicht wichtiger Befehle . . . 2
2.1.1 Online-Hilfe, allgemeine Befehle . . . 2
2.1.2 Navigation im Verzeichnisbaum . . . 2
2.1.3 Datei-Eigenschaften anzeigen und ändern . . . 3
2.1.4 Datei-Inhalt bearbeiten . . . 3
2.1.5 Verzeichnisse und Dateien verwalten . . . 4
2.1.6 Zugriff auf FAT-Disketten und Windows-Rechner . . . 4
2.2 Daten-Quellen und -Senken . . . 5

3 Shell **5**
3.1 Wichtige Syntax-Elemente, Benutzer-spezifische Konfigurationsdateien . . . 5
3.2 Wichtige Umgebungsvariablen . . . 6
3.3 Zeichen mit besonderer Bedeutung für ksh und bash . . . 6

4 Dateisystem **8**
4.1 Datei-Namen . . . 8
4.1.1 Wildcards für Datei-Namen . . . 8
4.1.2 Empfohlene Zeichen für Datei-Namen . . . 8
4.2 Legende der Datei-Typen und Zugriffsrechte . . . 9
4.3 Wichtige UNIX-Verzeichnisse . . . 10

1 UNIX-Konventionen

- UNIX unterscheidet zwischen Groß- und Kleinschreibung.
- In Namen von Umgebungsvariablen werden alle Buchstaben groß geschrieben.
- Außer den Namen der Umgebungsvariablen wird alles klein geschrieben, z. B. Verzeichnis-, Datei- und Programmnamen.

2 UNIX-Befehle

2.1 Übersicht wichtiger Befehle

2.1.1 Online-Hilfe, allgemeine Befehle

Funktion	UNIX	MS-DOS	VMS
Online-Handbuch	**man** *Befehl*	**HELP** *Befehl*	**HELP** *Befehl*
Kopfzeilen des Online-Handbuchs nach Schlüsselwort durchsuchen	**man -k** *Key*		
Text ausgeben	**echo** *Text*	**echo** *Text*	**WRITE** *Ziel Text*
Bedingung prüfen	**test** *Bedingung* [a] **[** *Bedingung* **]**	**errorlevel** *Zahl* [b] **exist** *Datei-Name*	[c]
Arbeit beenden, abmelden	**exit**		**LOGOUT**

[a] Der Befehl **test** bzw. **[Bedingung]** prüft die vorgegebene Bedingung. Das Ergebnis dieser Prüfung wird meistens mit den Befehlen **if**, **while** oder **until** ausgewertet.

[b] **errorlevel** und **exist** sind keine eigenständigen MS-DOS-Befehle, sondern Parameter des Befehls **if**. Der Befehl ist nur in Batch-Dateien erlaubt, aber nicht in der Kommandozeile.

[c] Bei VMS gibt es diverse Operatoren und Funktionen zum prüfen von Bedingungen.

2.1.2 Navigation im Verzeichnisbaum

Funktion	UNIX	MS-DOS	VMS
Arbeits-Verzeichnis anzeigen	**pwd**	**pwd**	**SHOW DEFAULT**
Arbeits-Verzeichnis wechseln	**cd** *Name*	**cd** *Name*	**SET DEFAULT** *Name*
Vollständigen Pfad eines Programms anzeigen	**which** *Name*		
Verzeichnis rekursiv nach Dateien mit bestimmten Namen, Alter, ... durchsuchen	**find** *Parameter* [1]		

[1] Nicht zu verwechseln mit dem MS-DOS-Befehl **find**, siehe S. 3!

2.1.3 Datei-Eigenschaften anzeigen und ändern

Funktion	UNIX	MS-DOS	VMS
Verzeichnis-Inhalt anzeigen	**ls** *Name*	**dir** *Name*	**DIR** *Name*
Datei-Namen und -Attribute anzeigen	**ls -l** *Name*	**attrib** *Name*	**DIR/PROT** *Name*
Datei-Typ anzeigen	**file** *Datei*	(Datei-Endung)	(Datei-Endung)
Zugriffsrechte ändern	**chmod** *Rechte Name*	**attrib** *Rechte Datei*	**SET PROTECTION**

2.1.4 Datei-Inhalt bearbeiten

Funktion	UNIX	MS-DOS	VMS
Datei-Inhalt ausgeben	**cat** *Datei* [a]	**type** *Datei*	**TYPE** *Datei*
Seiten-weise anzeigen	**more** *Datei* **less** *Datei* [b]	**more <** *Datei* ... **| more** [c]	**TYPE/PAGE** *Datei*
Sortierte Ausgabe	**sort** *Datei*	**sort <** *Datei* ... **| sort**	**SORT** *Quelle Ziel*
Datei-Inhalt drucken	{ **lp** / **lpr** } *Datei* [d]	**print** *Datei*	**PRINT** *Datei*
Ausgabe mit Seiten-Nr. usw.	**pr** *Datei* **mpage** *Datei* [e]		
Alle Zeilen mit bestimmtem Muster aus Datei extrahieren	**grep** *Muster Datei*	**find** *Muster Datei*	**SEARCH** *Parameter*
Bestimmte Spalten aus Datei extrahieren	**cut** *Spalten-Angaben Datei*		
Datei-Inhalt vergleichen	**diff** *Datei1 Datei2*	**fc** *Datei1 Datei2*	**DIFF** *Datei1 Datei2*

[a] cat = concatenate files and print to standard output

[b] **more** ist Standard. **less** ist die GNU-Variante mit mehr Funktionen, die aber nur bei manchen UNIX-Derivaten standardmäßig mitgeliefert wird.

[c] Die MS-DOS-Befehle **more** und **sort** gehören zu den drei Filter-Befehlen, die bei manchen MS-DOS-Versionen nur von *stdin* lesen können.

[d] Je nach UNIX-Dialekt: **lp** oder **lpr**

[e] **mpage** ist meist nur bei Linux-Systemen verfügbar.

2.1.5 Verzeichnisse und Dateien verwalten

Funktion	UNIX	MS-DOS	VMS
Verzeichnis anlegen	**mkdir** *Name*	**mkdir** *Name*	**CREATE/DIR** *Name*
Leeres Verzeichnis löschen	**rmdir** *Name*	**rmdir** *Name*	**DELETE** *Name*.DIR;∗
Datei löschen ... mit Rückfrage	**rm** *Datei* **rm -i** *Datei*	**del** *Datei*	**DELETE** *Datei* **DEL/CONFIRM** *Datei*
Verzeichnis einschließlich aller enthaltenen Dateien und Unterverzeichnisse löschen	**rm -r** *Name*	**deltree** *Name*	
Datei kopieren	**cp** *Quelle Ziel*	**copy** *Quelle Ziel*	**COPY** *Quelle Ziel*
Datei umbenennen	**mv** *alt neu*	**rename** *alt neu*	**RENAME** *alt neu*
Datei-Archiv verwalten	**tar** *Parameter*		

2.1.6 Zugriff auf FAT-Disketten und Windows-Rechner

FAT-Disketten:

Für den Zugriff auf Disketten mit FAT-Dateisystem ("MS-DOS"- bzw. "MS-Windows"-Disketten) gibt es **unter Linux** die Programme der *mtools*. Die wichtigsten Befehle sind:

Verzeichnis-Inhalt anzeigen:	**mdir**	*dosdatei*
Verzeichnis auf DOS-Diskette anlegen:	**mmd**	*dosverzeichnis*
Datei(en) von oder zum DOS-Laufwerk kopieren:	**mcopy**	*Quelle Ziel*

Bei Pfad- oder Datei-Namen, die das DOS-Laufwerk betreffen, muss jeweils dessen Laufwerks-Buchstabe (normalerweise "*A:*") am Anfang des Pfades stehen.
Beispiel: `mcopy unixfile.txt` **`A:`**`/dosfile.txt`

Zugriff auf Dateien von Windows-Rechnern:

Das Programm **smbmount**[2] ermöglicht (sofern installiert) unter UNIX dasselbe wie **net use** bzw. "**Netzlaufwerk verbinden**" unter MS-Windows. Als Verbindungs-Punkt auf dem UNIX-Rechner wird der Name eines vorhandenen (leeren) Verzeichnisses verwendet.
Beispiel:

Sie wollen die Freigabe *users* des Windows-Rechners *w2s1* unter dem lokalen, leeren UNIX-Verzeichnis */mnt/public/w2s1* einbinden. Ihr Benutzername auf dem Rechner *w2s1* sei *dagobert*.
Befehle:

Netzlaufwerk verbinden, nur Lese-Zugriffe erlaubt:

```
smbmount //w2s1/users /mnt/public/w2s1 -o ro,username=dagobert
```

Netzlaufwerk verbinden, Lese- und Schreibzugriffe erlaubt:

```
smbmount //w2s1/users /mnt/public/w2s1 -o username=dagobert
```

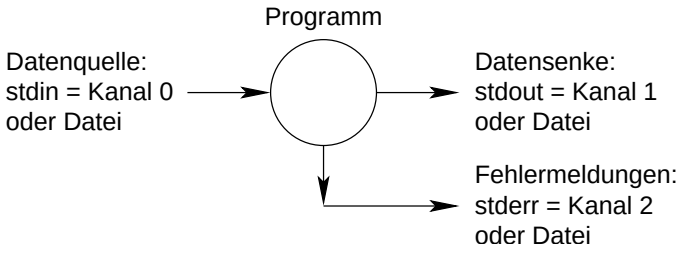
Netzlaufwerk trennen:

```
smbumount /mnt/public/w2s1
```

[2] **smbmount** ist Teil des frei verfügbaren Samba-Programmpakets. Siehe `http://www.samba.org/` bzw. `http://de.samba.org/`

2.2 Daten-Quellen und -Senken

Die meisten Programme arbeiten als Filter: Daten werden aus der Daten-Quelle gelesen, verarbeitet und in eine Daten-Senke geschrieben.



Kanal	Bezeichnung	File-Descriptor	Voreinstellung
stdin	Standard-Eingabe	0	Tastatur
stdout	Standard-Ausgabe	1	Bildschirm
stderr	Standard-Error	2	Bildschirm

Wenn der Parameter für den Datei-Namen beim Programmaufruf fehlt, lesen die meisten Programme die Daten von *stdin*.
Die meisten Programme schreiben ihre Daten nach *stdout*.

3 Shell

3.1 Wichtige Syntax-Elemente, Benutzer-spezifische Konfigurationsdateien

	UNIX	MS-DOS	VMS
Kennzeich. f. Kommando-Switches bzw. -Qualifier	–	/	/
Eingaben aus Datei lesen	**`Prog <`** *Datei*	**`Prog <`** *Datei*	
Ausgaben in Datei leiten	**`Prog >`** *Datei*	**`Prog >`** *Datei*	
Fehlermeldungen in Datei umleiten	**`Prog 2>`** *Datei*		
Ausgaben an anderes Programm umleiten	**`Prog1 | Prog2`**	**`Prog1 | Prog2`** [a]	
Typische Angabe eines Datei-Pfades	/dir/sub/file	D:\DIR\SUB\FILE.EXT	DISK:[DIR.SUB]FILE.EXT;1
Kennzeichen für Umgebungsvariablen	${VARNAME} $VARNAME	%VARNAME%	VARNAME:
Benutzer-spezifische Konfigurationsdateien	.profile .kshrc .bashrc .cshrc [b]	AUTOEXEC.BAT CONFIG.SYS	LOGIN.COM

[a] Bei MS-DOS ist die Umleitung nur an die drei Filter-Befehle **`more`**, **`find`**, und **`sort`** erlaubt.
[b] je nach verwendeter Shell

3.2 Wichtige Umgebungsvariablen

PATH Liste der Verzeichnisse, in denen nach ausführbaren Programmen gesucht wird, z. B:

`PATH=$PATH:/$HOME/bin:/usr/local/bin`

ACHTUNG: Ohne explizite Pfadangabe findet UNIX ein Programm im aktuellen Verzeichnis nur, wenn die Variable **`PATH`** einen Eintrag mit einem einzelnen Punkt als Synonym für das aktuelle Verzeichnis enthält, z. B:
`PATH=$PATH:/$HOME/bin:/usr/local/bin:.`

Aus Sicherheitsgründen sollte **`PATH`** für den System-Verwalter nie das jeweils aktuelle Verzeichnis enthalten.

PS1 Prompt, mit dem die Shell anzeigt, dass sie auf die Eingabe des nächsten Befehls wartet.

HOME *HOME*-Verzeichnis des Benutzers

DISPLAY Display für X11-Grafikoberfläche, z. B.

`DISPLAY=miraculix:0.0`

3.3 Zeichen mit besonderer Bedeutung für ksh und bash

Zusätzlich zu den Wildcard-Zeichen für Datei-Namen (siehe Abschnitt 4.1.1 auf Seite 8) haben folgende Zeichen beim Interpretieren der Zeile durch die *ksh* oder die *bash* eine besondere Bedeutung:

\# bewirkt, dass der Rest bis zum Zeilenende ignoriert wird (Kommentare in Shell-Skripten).

$ kennzeichnet das folgende Wort als Variablen-Namen:
`$VARIABLE` oder besser **`${VARIABLE}`** wird ersetzt durch den Inhalt der Variablen.

" Innerhalb von Text, der zwischen Gänsefüßchen (= doppeltem Hochkomma) steht, verlieren alle Zeichen außer *$*, "Backslash" \ und dem "Hochkomma rückwärts" ihre besondere Bedeutung.
Text-Ersatz für Shell-Variablen und für Befehle zwischen "Hochkomma rückwärts" wird ausgeführt.
Beim Zerlegen des Befehls in einzelne Argumente wird der gesamte Text zwischen Gänsefüßchen wie ein einziges Wort interpretiert.

Beispiel: `echo "Die Zeichen > und & wirken nicht, ${HOME} wurde ersetzt"`
liefert `Die Zeichen > und & wirken nicht, /home/student wurde ersetzt`

´ In Text, der zwischen einfachen Hochkommata steht, verlieren alle Zeichen mit besonderer Bedeutung diese Bedeutung. Die Zeichen werden als normale Zeichen behandelt.
Innerhalb des Textes findet keinerlei Text-Ersatz statt.
Beim Zerlegen des Befehls in einzelne Argumente wird der ganze, zwischen den Hochkommata eingeschlossene Text wie ein einziges Wort interpretiert.

Beispiel: `echo ´Die Zeichen > und & wirken nicht, ${HOME} bleibt unveraendert´`
liefert `Die Zeichen > und & wirken nicht, ${HOME} bleibt unveraendert`

` Text, der zwischen "Hochkomma rückwärts" steht, wird als Befehl ausgeführt und durch den Text ersetzt, den dieser Befehl als Ausgabe nach *stdout* schreibt.

Beispiel: `echo "Aktuelles Datum und Uhrzeit: `date` "`
liefert `Aktuelles Datum und Uhrzeit: Mon Feb 2 18:44:11 CET 2004`

& bewirkt, dass der Befehl im Hintergrund ausgeführt wird.

Beispiel: `xemacs file.c &`

; trennt aufeinander folgende Befehle innerhalb einer Zeile.

Beispiel: `cd /tmp ; ls`

( ) führt die Befehle, die zwischen den Klammern stehen, in einer separaten Shell aus.

Beispiel: `( cd /tmp ; ls )`

&& verknüpft zwei Befehle: Der Zweite wird nur ausgeführt, wenn der erste erfolgreich war, d. h. mit Return-Code 0 endete.

Beispiel: `cd verzeichnis && echo "Befehl war erfolgreich"`

|| verknüpft zwei Befehle: Der Zweite wird nur ausgeführt, wenn der erste *nicht* erfolgreich war, d. h. mit Return-Code *ungleich* 0 endete.

Beispiel: `cd verzeichnis || echo "Befehl war nicht erfolgreich"`

| verbindet *stdout* des ersten Befehls über eine Pipe mit *stdin* des zweiten Befehls.

Beispiel: `ls -l | less`

> leitet *stdout* in eine Datei um.
Wenn die Datei noch nicht existiert, wird sie automatisch angelegt.
Wenn die Datei schon existiert, wird deren bisheriger Inhalt gelöscht.

Beispiel: `ls -l > datei`

>> hängt die Ausgaben über *stdout* an eine Datei an.
Wenn die Datei noch nicht existiert, wird sie automatisch angelegt.
Wenn die Datei schon existiert, wird die Ausgabe hinter den bisherigen Inhalt angehängt.

Beispiel: `ls -l >> datei`

>& verknüpft zwei Ausgabekanäle eines Prozesses miteinander:
`2>&1` leitet die Ausgabe von File-Descriptor 2 = *stderr* auf das aktuelle Ziel von File-Descriptor 1 = *stdout*.

Beispiel: `find /tmp > liste 2>&1`
leitet Standard-Ausgabe und Fehlermeldungen von `find /tmp` in die Datei *liste* um.

< liest die Daten für *stdin* aus einer Datei statt von der Tastatur.

<< liest die Daten für *stdin* aus der selben Quelle, aus der auch die Befehlszeile gelesen wurde.

Beispiel:

```
cat << EOF
Die folgenden Zeilen werden gelesen und an stdin des
Befehls 'cat' geschickt.
Die Eingabe endet, wenn am Anfang einer Zeile das Wort steht,
das oben hinter '<<' stand. In diesem Fall ist das also 'EOF':
EOF
```

4 Dateisystem

4.1 Datei-Namen

4.1.1 Wildcards für Datei-Namen

Das Ersetzen von Angaben mit Wildcards durch passende Dateinamen erfolgt durch die Shell.

∗ Ein Stern wird ersetzt durch kein oder beliebig viele beliebige Zeichen, jedoch nicht durch einen Punkt am Anfang eines Dateinamens.
Im Gegensatz zu MS-DOS kann der Stern an beliebigen Stellen des Namens stehen, z. B. ∗*ab*∗

? Ein Fragezeichen wird ersetzt durch genau ein beliebiges Zeichen, jedoch nicht durch einen Punkt am Anfang eines Dateinamens.

[...] Zeichen zwischen eckigen Klammern werden ersetzt durch genau eines dieser Zeichen. Die Klammern können Aufzählungen und Bereichsangaben enthalten.
Beispiel: *a[b–ek]x* paßt auf folgende Namen: *abx acx adx aex akx*
Das Muster paßt aber nicht zu *aax afx* und *abbx*

4.1.2 Empfohlene Zeichen für Datei-Namen

UNIX unterscheidet zwischen Groß- und Kleinschreibung.
Dateinamen sollen nur aus den Zeichen a–z, Unterstrich, Punkt, Komma und Ziffern bestehen. Bindestrich (= Minus-Zeichen) ist ebenfalls erlaubt, sollte jedoch nicht am Anfang des Namens stehen[3]. A–Z sind möglich, widersprechen aber der Konvention, nur Kleinbuchstaben für Dateinamen zu verwenden.
Dateien, deren Namen mit einem Punkt beginnt – z. B. *.profile* – sind *versteckte Dateien*: Der Befehl **`ls`** zeigt solche Dateien nur beim Aufruf mit der Option **`-a`** an. Auch die Wildcard-Zeichen '∗' und '?' am Anfang eines Dateinamens werden nicht durch Namen ersetzt, die mit Punkt beginnen.
Der Schrägstrich '/' trennt Verzeichnis- bzw. Dateinamen in Pfadangaben.
Folgende Zeichen sollten nicht in Dateinamen verwendet werden:

- Tabulator- und Leerzeichen [4]
- alle Zeichen, die von der Shell besonders behandelt werden [4]
- Wildcard-Zeichen für Dateinamen [4]
- deutsche Umlaute und 'ß' [5]

[3] Dateinamen, die mit einem Bindestrich beginnen, werden von Programmen als Programm-Option behandelt.
[4] Dateinamen mit solchen Zeichen sind zwar möglich, ihre Verwendung erfordert jedoch besondere Maßnahmen durch den Anwender.
[5] Deutsche Sonderzeichen in Dateinamen verursachen oft Ärger beim Übertragen auf andere Rechnerplattformen.

4.2 Legende der Datei-Typen und Zugriffsrechte

Ausgabe des Befehls **`ls -l`** in einem Beispielverzeichnis:

```
-rw-r--r--   1 mueller  users           4 Feb  6  1999 datei1.txt
-rwxr-xr-x   1 mueller  users           9 Jul 24 19:47 datei2.sh
-rw-r--r--   1 schulze  guest          35 Jul 24 19:48 datei3.txt
-rw-rw-r--   1 maier    users          22 Jul 24 19:48 datei4.txt
-rw-rw-rw-   1 mueller  users          27 Mar 28  2000 file
lrwxrwxrwx   1 maier    users          10 Dec 29 11:53 symlink -> datei4.txt
drwxr-xr-x   2 schulze  guest        1024 Dec 29 11:50 verzeichnis
```

Bedeutung:

```
|Zugriffs-   |   Eigentümer:       Datei-  Zeitstempel   Name
| rechte     | User     Gruppe     größe
|            |
|            +- Link-Count
+-------------- Dateityp
```

Legende:

Dateityp:

- **–** "normale" Datei, z. B. Text, Daten, Programm
- **b** *Block-Devicefile*: Gerätedatei[6] mit blockorientierter, d. h. gepufferter Ein- / Ausgabe
- **c** *Character-Devicefile*: Gerätedatei mit zeichenorientierter, d. h. sequentieller Ein- / Ausgabe
- **d** *Directory*: Dateiverzeichnis
- **l** *Symbolischer Link*: Verweis auf anderen Verzeichniseintrag
- **p** *Named Pipe*: FIFO-Datei[7]
- **s** *Socket*

Zugriffsrechte sind in 3 abgestuften Berechtigungsklassen vergeben:

- `rwx------` Zugriffsrechte für den *User*, d. h. Prozesse mit gleicher User-ID wie der Dateieigentümer
- `---rwx---` Zugriffsrechte für die *Group*, d. h. Prozesse mit gleicher Gruppen-ID wie die Gruppe, der die Datei gehört
- `------rwx` Zugriffsrechte für *Others*, d. h. Prozesse mit anderer User-ID und anderer Gruppen-ID

[6] Weitere Informationen zu Gerätedateien finden Sie im Abschnitt "4.4.2.2 Block- und zeichenorientierte Ein- und Ausgabe" auf Seite 121 des Skripts *"Rechnerorganisation", Nachrichtentechnik 5. + 6. Hj.*

[7] Mehr zu FIFO-Dateien finden Sie im Abschnitt "4.5.3 FIFO-Dateien" auf Seite 134 des Skripts *"Rechnerorganisation", Nachrichtentechnik 5. + 6. Hj.*

Für jede Berechtigungsklasse werden folgende Zugriffsrechte unterschieden:

`r--` *read*: Lesezugriff erlaubt

`-w-` *write*: Schreibzugriff erlaubt.
Schreibrecht auf ein Verzeichnis erlaubt, Dateien in dem Verzeichnis anzulegen und zu löschen!

`--x` *execute*: Ausführen der Datei erlaubt. Dies ist nur für Programme oder Skripts sinnvoll.
Bei Verzeichnissen erlaubt `x`, das Verzeichnis zu durchsuchen sowie in das Verzeichnis zu wechseln.
Alternativ zu `--x` gibt es noch die Rechte `--s` und `--t`. Details siehe **`man 1 chmod`** oder **`info chmod`** oder **`man ls`**

4.3 Wichtige UNIX-Verzeichnisse

/	*root*-Verzeichnis = *"Wurzel"* des Dateibaumes
/bin	wichtige, elementare UNIX-Programme
/dev	Gerätedateien
/etc	Rechner-spezifische Konfigurationsdateien
/etc/opt	Konfigurationsdateien für Anwendungen in /opt: /etc/opt/*applikation* gehört zu /opt/*applikation*
/home	Basis für benutzerspezifische Verzeichnisse: Jeder Benutzer hat hier ein eigenes *HOME*-Verzeichnis, z. B. */home/dagobert* für Benutzer *dagobert*.
/lib	wichtige Systembibliotheken
/opt	Programme und statische Dateien von Anwendungen. Jede Anwendung hat ein eigenes Unterverzeichnis.
/sbin	Programme für den Systemverwalter
/tmp	temporäre Dateien
/usr/bin	allgemeine UNIX-Programme
/usr/include	Header-Dateien für C-Compiler, z. B. stdio.h
/usr/lib	allgemeine Bibliotheken
/var	Dateien, deren Größe sich im Lauf der Zeit ändert
/var/opt	Daten zu Anwendungen in /opt: /var/opt/*applikation* gehört zu /opt/*applikation*