# Einführung in Unix

Hermann Rochholz*

14. Februar 2004

## Inhaltsverzeichnis



*Forschungszentrum Karlsruhe, Abteilung INR, Tel. 07247 / 82-6105

# 1 Einführung

Ein Betriebssystem mit seiner Kommandosprache ist wie eine Fremdsprache, die die Verbindung zwischen Mensch und Maschine herstellt. Diese muß erlernt werden. Diese Einführung wendet sich insbesondere an Anwender, die bisher Erfahrungen im Betriebssystem DOS sammeln konnten.[1] Die Beschreibung versucht, englische Fachbegriffe zu vermeiden, die häufig von Personen verwendet werden, die sehr viel Umgang mit Computern haben und so zu Mißverständnissen führen. Sie werden jedoch dort eingesetzt, wo Optionen von Kommandos z.B. den ersten Buchstaben von englischen Worten darstellen und somit das Erlernen des Betriebssystems Unix vereinfachen. Beispiele werden so gewählt, daß sie unter Analogieschlüssen direkt verwendet werden können.

Dabei ist die Vorgehensweise so, daß ein "Zurechtkommen" auf der Maschine bzw. auf dem Computer gewährleistet ist, da das System Unix sehr umfangreich ist, womit sich aber der "normale" Anwender nicht beschäftigen muß. Außerdem wird auf mögliche Stolperfallen hingewiesen.

[1] Historisch ist Unix älter als DOS. Offensichtlich aus lizenzrechtlichen Gründen wurde die Syntax anders gewählt

Weiterhin gibt es eigentlich nicht "Das Unix", sondern es existieren, wie bei jeder Sprache oder auch bei DOS ("Opendos", "Msdos") Dialekte wie z.B. "aix", "unicos", "hpux" und "linux", die sich unterscheiden. Es existieren hier 2 Linien, die als "BSD"- und "System V"-Unix bezeichnet werden.

# 2 Orientierung im System nach dem Einloggen

Da Unix ein Multiusersystem ist, können auf einer Maschine gleichzeitig mehrere Personen arbeiten. Dazu ist die Einrichtung eines Kennzeichens und eines dazugehörigen Paßwortes durch den Systemadministrator notwendig. An der Maschine sitzend erscheint der Login-Prompt und nach Angabe des Kennzeichens wird das Paßwort abgefragt. Dieses wird verdeckt eingegeben.

```
login:
password:
```

Unix unterscheidet auch hier zwischen Groß- und Kleinschreibung. Nach jeder Eingabe ist "Enter", einzugeben, was im deutschen die Wagenrücklauftaste ist. "Friß" scheint mir die treffendste Bezeichnung dieser Taste. Nach dem Einlogvorgang erscheint ein sogenannter Prompt, der je nach verwendeter Shell unterschiedlich ist. Mehr dazu etwas später.

Zur ersten Orientierung auf einer Maschine sind folgende Kommandos geeignet, die teilweise selbsterklärend sind: Die Kommandos bedeuten der Reihe nach:

| | |
|---|---|
| `whoami` | WER bin ich? |
| `pwd` | WO befinde ich mich, *P*rint *W*orking *D*irectory(engl.(Verzeichnis))? |
| `hostname` | Auf WELCHER MASCHINE (Welchem Host) befinde ich mich? |
| `df` | Wieviel Platz ist auf dem Dateisystem noch frei? (*D*isplay *F*ilesystem) |
| `df .` | Wieviel ist auf dem lokalen Dateisystem noch frei? |
| `uname -a` | WELCHES Betriebssystem ist mit welcher Version auf der Maschine? |
| `quota` | WIEVIEL darf ich belegen? (Falls quotas aktiviert sind) |
| `finger` | WER ist im Moment auf der Maschine? |

Mit dem Kommando "`logout`" oder "`exit`" verläßt man das Kennzeichen, bei den meisten Kennzeichen funktioniert auch "Control D".[2]

[2] Die Control-Taste ist die Taste mit dem "ctrl" oder "Strg" darauf.

# 3 Kommandos und Hilfe des Systems

Die Kommandosyntax unter Unix ist folgende:

`Kommando -OPTION(EN)` oder auch

`Kommando -OPTION1 -OPTION2`

Zwischen Kommando und Option(en) muß sich ein Leerzeichen befinden. Das Zeichen "–" wird als "Minus" bezeichnet. Bei vielen Kommandos schließt sich nach den Optionen ein Leerzeichen und dann eine Eingabedatei an. In vielen Fällen kann man sich Kurzinformationen über mögliche Optionen eines Kommandos mit

`Kommando -Hilfeoption`

ausgeben lassen, wobei man als Hilfeoption help, -help, '?', oder h ausprobieren muß, sodaß man auf das Lesen der Dokumentation verzichten kann. Sollte das Kommando eine Eingabedatei benötigen, so funktioniert häufig nur die Eingabe von "`Kommando`".

Genaue Informationen bekommt man mit

`man Kommando` `(z.B. man man)`

Es wird nun eine Hilfedatei angezeigt, die als "Manpage" bezeichnet wird. Die Syntax, um sich diese Datei anzuschauen, ist f(*f*orward), d(*d*own) und q(*q*uit), mit "/Buchstabenfolge" sucht man nach dieser. b(*b*ackward) und u(*u*p) funktioniert nicht auf allen Maschinen, was der Systemadministrator aber in der Regel beheben kann.

Oben befindet sich auf dieser "Manpage" die Kommandosyntax, in der Mitte die Beschreibung der Optionen und unten Pfade und verwandte Kommandos, die in der Hinsicht dieses Kommandos nützlich sein können.

Manpages sollte man nicht direkt auf den Drucker schicken, da durch Unterstreichen bzw. Hervorheben von Passagen Steuerzeichen enthalten sind.

Sehr wichtig zum Zurechtfinden in einem System ist das Kommando

`man -k Thema`

wobei dann alle Kommandos ausgegeben werden, die mit diesem Thema zu tun haben. Die Option "k" bedeutet *k*eyword, engl(Schlüsselwort). Zum Beispiel mit "`man -k debug`" findet man die Namen der auf der jeweiligen Maschine installierten Debugger (engl(Entlauser)). "`man -k`" wird häufig auch mit "`apropos`" oder "a" abgekürzt.

# 4 Bewegen von Dateien

Elementare Kommandos im Betriebssystem Unix sind folgende:

| | |
|---|---|
| `ls` | *l*ist |
| `cd Verzeichnis` | *c*hange *D*irectory (engl(Wechsele Verzeichnis)) |
| `cd ..` | Wechsele in Verzeichnis, das in der Dateistruktur höher steht |
| `mkdir Verzeichnis` | *MaK*e *DIR*ectory |
| `mv Datei/Verzeichnis` | Bewege (*MoV*e) Datei/Verzeichnis, ist gleichzeitig ein Rename |
| `rm Datei/Verzeichnis` | *ReM*ove Datei/Verzeichnis |
| `rmdir Verzeichnis` | *ReM*ove *DIR*ectory |

Dateien können mit

```
more Datei
```

, genauso wie die Manpages, angeschaut werden. Leider geht es auch hier nicht rückwärts. Das Kommando "less", das auf vielen Maschinen installiert ist, kann dies jedoch. Wie immer ist weniger mehr (less is more). Amerikaner lieben Wortspiele. Die Syntax hier ist, wie bei der Manpage, f(*f*orward), b(*b*ackward), d(*d*own), u(*u*p) und q(*q*uit).

Weiter sind die Kommandos

```
head Datei
tail Datei
```

vorhanden, die selbsterklärend sind.

Zum Editieren von Dateien stehen viele Editoren zur Auswahl. Der "xedit" ist dem "edit" unter DOS am ähnlichsten, kann allerdings genauso wenig. Gewöhnungsbedürftig sind der "vi" und der "emacs" oder "xemacs", die allerdings "Regexps" (*Reg*ular *exp*ressions, reguläre Ausdrücke) beherrschen, mit denen man recht komplexe Ausdrücke suchen und auch ersetzen kann. Während der "vi" den Vorteil hat, auf jeder Unix-Maschine installiert zu sein und er weiterhin komplett über die Tastatur bedient werden kann, spricht für den "Emacs", daß es sich bei ihm nicht um einen reinen Editor, sondern vielmehr um eine vollständige Programmierumgebung handelt, die sehr flexibel angepaßt werden kann.

# 5 Die Shell

Einige Worte zur sogenannten "Shell", was zum jetzigen Zeitpunkt zwar etwas abstrakt ist, jedoch nicht vermieden werden kann. Zunächst ist eine Shell eine Kommandoumgebung, in der Kommandos ablaufen. Eine Shell erkennt man (normalerweise) am Prompt. "%" zeigt normalerweise die C- oder Tc-Shell an, ">" steht als Voreinstellung für die Korn-Shell, die eine Weiterentwicklung der ursprünglichen bourne-Shell darstellt. Der Administrator hat als Prompt ein Lattenkreuz (#) der Bourne- oder Korn-Shell.

Dem Anfänger unter Unix wird die verwendete Shell zunächst einmal vollständig egal sein. Für ihn ist wichtig, daß die abgesetzten Kommandos "laufen", und das tun sie im allgemeinen unter jeder Shell. Nützlich wird für den Benutzer die Shell dann, wenn er z.B. riesig lange Kommandos eintippen muß. Hier halten alle gängigen Shells einen gespeicherten Satz von zuvor abgesetzten Kommandos ("History") und Namensexpansionsmechanismen bereit. Die Shells unterscheiden sich hier in der Weise, wie z.B. die Expansion gehandhabt oder auf die "History" zugegriffen wird (Unter DOS der "Doskey"). Weiterhin unterscheiden sie sich in der Programmierung von Skripten, die unter DOS "Batch-Jobs" heißen. Hier stellen Shells eine riesige Menge eingebauter Möglichkeiten zur Verfügung, wie z.B. das "Abschneiden"von Dateiendungen oder das wiederholte Ausführen von gleichen Kommandos auf viele Dateien. In Shells können auch verschiedene Variablen gesetzt werden. Dabei unterscheidet man kleingeschriebene, lokale Variablen, die recht wenig zu sagen haben. Wichtig sind jedoch die großgeschriebenen Umgebungsvariablen ("Environment-Variablen"), die auch in Prozesse exportiert werden, die aus dieser Shell gestartet werden ("Tochtershells"). Setzt man z.B. die Environment-Variable "Pager" auf "less", so wird zum Betrachten der Manpages das less-Kommando benutzt, sodaß diese auch rückwärts betrachtet werden können. In den Shells wird weiterhin die Form des Prompts definiert und die Display-Variable gesetzt. Die Variablen kann man sich mit

```
printenv
```

*Print Env*ironment anzeigen.

Programme wie "mail" oder "at" eröffnen ebenfalls eine neue Shell. Mit

```
at 23:00
./MAIN
Ctrl-D
```

wird z.B. nachts um 23:00 Uhr das Programm "MAIN" gestartet, vgl. auch nächstes Kapitel. Mit "Control-d" schließt man Shells ab. Diese Vorgehensweise macht es unter Unix möglich, auf hoch belasteten Maschinen tagsüber die Systemlast zu reduzieren.

## 5.1 Sonderzeichen

Wie unter DOS existieren unter Unix Zeichen, die von der Shell besonders interpretiert werden.

Pfade werden bei Unix mit "/" (englisch: Slash) getrennt, weitere wichtige spezielle Zeichen sind ".."(= Verzeichnis zurück), "." (= aktuelles Verzeichnis) und "~ " (= Heimatverzeichnis). Beginnt eine Pfadangabe mit "/", so bedeutet dies eine absolute Pfadangabe, was bei DOS bedeuten würde, daß man z.B. die Pfadangabe von C: beginnend meint.

Weiterhin existieren noch Umlenkungssymbole, die mit denen unter DOS teilweise identisch sind. Mit

```
Programm_mit_Bildschirmausgabe > Datei
```

werden Bildschirmausgabedaten in eine Datei umgelenkt[3]. Mit

```
Programm_mit_Bildschirmausgabe >> Datei
```

wird sie an diese Datei angehängt.[4] Mit

```
Programm_mit_Bildschirmausgabe |  lpr -Pdruckename
```

würde man die Ausgabe des Programmes direkt auf den Drucker schicken. Anstelle des "Pipe" (engl(Rohr))-Kommandos funktioniert ">" hier nicht, da hiermit eine Datei mit dem Namen lpr erzeugt werden würde. Die Pipe schickt die Ausgabe eines Kommandos, die normalerweise auf Standard-Ausgabe[5] erscheint, in das "lpr"-Kommando, welches dann von Standard-Eingabe (Standard-Input) liest.

Ein weiteres Sonderzeichen unter Unix ist der Lattenzaun "#", hinter dem in einer Zeile keine Kommandointerpretation mehr stattfindet. Es wird als Kommentarsymbol in Skripten verwendet.

Das Dollar-Zeichen bewirkt eine Interpretation von Variablen. Mit "$Variable" wird auf die Variable mit dem Namen "Variable" zugegriffen.

";" wird als Trennung eingesetzt, sodaß 2 Kommandos unabhängig in einer Zeile stehen können.

Als Sonderzeichen gelten weiterhin diverse Anführungszeichen und der Kammeraffe (@), die in anderen Kapiteln abgehandelt werden. Der Backslash "\" trennt, im Gegensatz zu DOS, keine Verzeichnisse, sondern er dient dazu, Sonderzeichen als normale Zeichen nutzen zu können. Z.B. kann man eine leere Datei mit dem Namen "Eine Datei" erzeugen, indem man

[3] U.U. muß man die Datei erst löschen oder eine Variable der Shell umsetzen, wenn die Datei existiert

[4] Für eine separate Umlenkung von normaler Ausgabe und Fehlerausgabe eines Programmes liest man die Manpages der Shell und fragt den Administrator nach Beispielen.

[5] Bei Arbeiten am Bildschirm ist dieser die Standard-Ausgabe (Standard-Output)

```
touch Eine\ Datei
```

eingibt. Hier wird dann das Leerzeichen als normales Zeichen benutzt.

Kein Sonderzeichen stellt der Unterstrich (Underscore) "_" dar, der wie ein normaler Buchstabe gehandhabt wird.

# 6 Programme

## 6.1 Übersetzen von Programmen

Wie bereits erwähnt, kann man mit "man -k compile" ermitteln, welche Übersetzer (compiler) auf einer Maschine installiert sind, anderweitig ist die Befragung des Systemadministrators notwendig. Um die Systemlast zu reduzieren und auch viel Zeit zu sparen ist unter Unix die Verwendung eines "Makefiles" zu empfehlen, das bewirkt, daß bei Änderung einzelner Programmteile auch nur diese neu übersetzt werden müssen und nicht das gesamte Programm. Sollte das Quellprogramm in einem Stück vorliegen, existiert für Fortran 77 in Unix sogar ein Makefilegenerator[6], der die Quelldatei in einzelne Module zerlegt und gleichzeitig das Makefile schreibt. Lediglich die Übersetzungsoptionen "FFLAGS" muß man im Makefile anpassen.

Die Übersetzung startet man dann einfach mit dem Kommando "make". Voreinstellungsmäßig wird dann das Programm "a.out" erzeugt, das man jedoch mit einem vernünftigen Namen umbenennen sollte. [7] Im Folgenden wird das ausführbare Programm mit "`MAIN`" bezeichnet.

## 6.2 Ausführen von Programmen

Programme, Skripten und auch Kommandos können dann ausgeführt werden, wenn sie mit dem Kommando "ls -l" z.B. folgende Ausgabe bringen:

```
-rwxr-xr-x   1 Benutzer   Gruppe     195340 Nov  2 17:23 MAIN
```

MAIN ist hier für den Benutzer (Zeichen 2-4) e*X*ecutable (=ausführbar), *W*ritable und *R*eadable. Dieses Programm kann einfach mit

```
./MAIN
```

[6] Auf der Cray heißt dieser "fmgen"

[7] Auf vielen Maschinen werden Dateien wie "a.out" oder "core" nach wenigen Wochen vom Systemadministrator gelöscht, da diese teilweise sehr groß sind und man nach kurzer Zeit sowieso nicht meht weiß, welchem Quellprogramm sie zuzuordnen sind

laufen. Dieser Befehl bedeutet, präzise ausgedrückt, daß das Programm "MAIN", das sich in dem "Verzeichnis, in dem man sich im Moment befindet" (Current Directory), ausgeführt wird. In den allermeisten Fällen kann man auf das "./" verzichten, wobei dann das Programm "MAIN" verwendet wird, was zuerst im Suchpfad für Programme auftaucht. Dies ist jedoch unsicher, da man u.U. ein falsches Kommando "MAIN" ausführt. Gerade ein Neuling fällt darauf leicht herein, wobei gerade Profis "./" vor dem Kommando verwenden. Und bei Systemadministratoren ist das "Verzeichnis, in dem man sich momentan befindet" nicht einmal im Suchpfad, damit sie aus Versehen kein Unheil anrichten können. [8]

## 6.3 Vordergrund- und Hintergrundprozesse

Ist das Programm MAIN gestartet, läuft es im Vordergrund. Mit "ctrl"-Z und dann "bg"(*b*ack*g*round) kann man es in den Hintergrund schicken. Diese Fähigkeit von Unix kommt aus der Urzeit, als es noch kein X-Windows-System gab und auf einem Bildschirm ohne grafische Fähigkeiten mehrere Prozesse parallel laufen mußten. Bei der Benutzung von "ctrl-Z" ist zu beachten: Selbst wenn ein Programm gestoppt ist, ist es aktiv. Dieses muß *vor* dem Ausloggen abgebrochen werden, sonst blockiert es den Rechner.[9] Weiterhin kann es gleich mit "`./MAIN &`" im Hintergrund gestartet werden, tippt man "fg"(=*f*ore*g*round) ein, kommt es wieder in den Vordergrund. Mit "ctrl-C" wird der Prozeß abgebrochen. Mit

```
jobs
```

werden die Hintergrundprozesse angezeigt. Laufen mehrere Prozesse im Hintergrund, so sind sie, wie alles unter Unix, durchnummeriert. Mit

```
fg %2
```

holt man z.B. den zweiten Prozeß vom Hinter- in den Vordergrund. Startet man das Programm mit

```
nohup ./Programm &
```

(*noh*ang*up*), so läuft das Programm nach dem Ausloggen weiter.

Ergebnisse werden bei Unix voreinstellungsmäßig in die Dateien "fort.Zahl" geschrieben, d.h. Ausgabekanal 12 würde auf die Datei fort.12 schreiben. Die Kanäle 1,5 und 6 sind mit Standard-Eingabe, Standard-Ausgabe und Standard-Fehlerausgabe verknüpft.

Wichtig beim Überwachen von Prozessen ist der Befehl

[8] Es kommt bisweilen vor, daß man ein in Programm (oder Skripten) ändert, wenn man das Programm aufruft, sich aber partout keine Änderung bei der Ausführung des Programmes einstellt. Hier hat man den Pfad bei der Ausführung nicht angegeben, zwei Programme mit demselben Namen im Pfad und man führt immer das falsche aus.

[9] Das gleiche gilt für den Abbruch von xterms, in denen Prozesse weiterlaufen können

```
ps -fu Benutzername
```

wobei ps *P*rozeß*S*tatus bedeutet. Hier werden auf der Maschine alle laufenden Prozesse mit ihrer Nummer in der Spalte "PID", was die "*P*rocess-*Id*entification" (Prozeßnummer) bedeutet, angezeigt. Anhand dieser Nummer kann jeder auf dem Rechner laufende Prozeß eindeutig identifiziert, gestoppt oder gelöscht werden.

Kennt man diese Nummer,[10] so kann man den jeweiligen Prozeß mit

```
kill PID
```

oder mit Gewalt mit

```
kill -9 PID
```

stoppen.

Unter Unix bedeutet ein "Prozeß" nicht immer ein Programm (wie z.B. ein Fortranprogramm) im eigentlichen Sinne. Es zählen auch "demons" wie z.B. der Zeilendruckerdämon (*l*ine *p*rinter *d*emon, lpd) zu den Prozessen. Ein demon überprüft in bestimmten Zeitabständen, ob etwas ausgeführt werden muß. Ist z.B. der "lpd" nicht aktiv, kann man nicht drucken. Weitere demons sind z.B. der "cron"-demon, der kontrolliert, ob Kommandos mit dem Kommando "cron" abgeschickt wurden, also wiederholend zu bestimmten Zeiten arbeiten.

Um sich das Suchen nach laufenden Programmen einfacher zu machen, kann man sich mit

```
ps -aef | grep Programm_Name
```

die Suche nach der richtigen Prozeßnummer erleichtern.

Dateien, die während des Programmlaufes geschrieben und dann immer länger werden, kann man mit "tail -f Datei" anschauen, was, wie unter Unix üblich, mit "ctrl-C" abgebrochen wird.

# 7 Benutzung verschiedener Maschinen

Die Benennung von Maschinen erfolgt ebenfalls nach einer festen Syntax. Die Identifikation einer Maschine "im Netz" erfolgt über den Hostnamen und Domainnamen, also z.B. caea01.fzk.de. Ein Benutzer kann somit mit

```
Benutzer@hostname.domainname
```

[10] Übrigens ist unter Unix eigentlich alles numeriert, wie z.B. auch die Dateien (Files), die sogenannte I-Node Nummern besitzen. Da man sich Nummern schlecht merken kann, besitzen Dateien Namen.

identifiziert werden. Der Klammeraffe @ wird im englischen als "at" bezeichnet. Genauso setzt sich dann auch die Mailadresse zusammen. Innerhalb eines Netzes genügt meist die Angabe des Hostnamens "caea01" ohne Domainnamen "fzk.de", um die Maschine anzusprechen.

Die gebräuchlichsten Programme zum Einloggen auf anderen Maschinen sind

```
telnet Maschine
rlogin Maschine -l Benutzer
```

, wobei wie beim direkten Einloggen auf einer Maschine hier zunächst ein Prompt erscheint. Zum Datentransfer wird

```
ftp Maschine
```

oder

```
rcp Datei Benutzer@hostname.domainname:Pfad
```

bzw.

```
rcp Benutzer@hostname.domainname:Pfad/Datei Datei
```

benutzt. Die Grundvoraussetzung dafür, daß die "*R*emote-*C*o*p*y" `rcp` funktioniert bzw. das "rlogin"-Kommando ohne Eingabe des Paßwortes durchgeführt werden kann ist die Einrichtung der Datei .rhosts auf dem eigenen Heimatverzeichnis, vgl. Kapitel 9.3.

Ein alltägliches Problem ist es, sich am Terminal von einer anderen Maschine ein Fenster schicken zu lassen. Aus Sicherheitsgründen am Günstigsten stellte sich bis jetzt die Freigabe über das Kommando "xauth" (Xauthority) heraus, was aber offensichtlich unter aix der IBM-Maschinen manchmal nicht unterstützt wird. Hier ist die Syntax "etwas länglich" und muß sogar in zwei Zeilen dargestellt werden:

```
xauth extract - $DISPLAY | rsh Andere_Maschine xauth merge - ;
xrsh Andere_Maschine xterm -display $DISPLAY
```

Mit diesem Kommando wird die Variable "DISPLAY" übergeben und auf der anderen Maschine eingeloggt.

Eine andere Methode stellt das "xhost"-Kommando dar. Grundsätzlich funktioniert dieses so, daß man auf der Maschine, an der man sitzt, und die hier beipielsweise "caea01.fzk.de" heißen soll, zunächst für die andere Maschine, die hier beispielsweise "inrirs22.fzk.de" heißen soll, die Erlaubnis geben muß, damit überhaupt ein Fenster geschickt werden darf, also z.B.

```
xhost +inrirs22.fzk.de
```

[11] Der zweite Punkt ist das Einloggen auf der anderen Maschine, also

```
rlogin inrirs22.fzk.de -l AndererBenutzer
```

Befindet man sich dann "auf der inrirs22", so muß man die richtige Anzeige setzen, wenn man sich von der inrirs Fester schicken lassen will. Unter der Korn-Shell ist dieses Kommando folgendes:

```
export DISPLAY=caea01.fzk.de:0.0
```

. Unter der C-Shell ist die Syntax zum Setzen der globalen Umgebungsvariablen "DISPLAY" anders, hier lautet es "`setenv DISPLAY caea01.fzk.de:0.0`". Die Variablen können mit "`printenv`" oder die Display-Setzung mit "`echo $DISPLAY`" angeschaut werden.
Wie man sieht, setzt sich das Display aus dem Namen des Bildschirms, an dem man sitzt und ":0.0" zusammen. Als Test kann man sich dann ein Fenster schicken lassen, also

```
xterm &
```

eine andere Möglichkeit wäre, das Display direkt beim Kommando anzugeben, was bei den meisten Anwendungen funktioniert, die ein Fenster schicken.

```
xterm -display caea01.fzk.de:0.0
```

# 8 Nützliche Kommandos

## 8.1 Suchen von Buchstabenfolgen in Dateien

Mit "grep" oder dessen Derivaten "egrep" bzw. "fgrep" kann man Buchstabenfolgen in Dateien finden, ohne sie zu editieren.

```
grep Buchstabenfolge Datei(liste)
```

z.B.:

```
grep -i xfeld *.f *.h
```

Hiermit werden alle Dateien, die auf "f" bzw. "h" enden, ausgegeben, in denen die Variable "xfeld" in Groß- oder Kleinschreibung verwendet wird, da die Option "i" *I*gnorecase bedeutet.[12]

[11] Grundsätzlich funktioniert auch das Kommando "`xhost +`", womit man allen anderen Maschinen die Erlaubnis gibt, was aber aus verständlichen Gründen aus Systemsicherheitsgründen vermieden werden soll.

[12] Bei Programmrevisionen ist der Befehl sehr nützlich, wobei zu erkennen ist, daß im vorliegenden Beispiel die Suche nach der Buchstabenfolge "x" sehr mühsam ist und das Suchergebnis sehr umfangreich wird.

## 8.2 Bestimmen des Pfades eines Kommandos

Welches Kommando des Suchpfades verwendet wird, wenn man keine Pfadangabe verwendet, (vgl. Kapitel 6.2) kann man mit dem Kommando "whence"- Kommando (Korn-Shell) bestimmen, das unter DOS kein Pendant hat:

```
whence MAIN
```

In der C-Shell lautet dieses Kommando "which". Dieses Kommando ist weiterhin bei vielen der heutigen Maschinen wichtig, wo viele Kommandos bzw. Programme mehrfach vorhanden sind und man u.U. auf eine alte Version oder auf einen falschen Compiler zugreift. Oder man kann mit "whence" ganz einfach ausprobieren, ob es ein Kommando auf einer Maschine überhaupt gibt.

## 8.3 Finden von Dateien und Kommandoanwendung auf diese

Die Syntax zum Finden ist unter Unix gewöhnungsbedürftig. Sie lautet:

```
find Verzeichnis -name Buchstabenfolge -print
```

Hiermit werden alle Dateien ausgegeben, die den Namen Buchstabenfolge enthalten. Mit dem find-Kommando kann man jedoch nicht nur etwas finden, sondern auch mit bzw. auf den/die gefundenen Dateien bzw. Verzeichnissen etwas ausführen.

```
find $HOME -name "*.ps" -exec gzip {} \;
find $HOME -name "*" -type f -exec chmod g+r {} \;
find $HOME -name "*" -type d -exec chmod g+xr {} \;
```

Das erste Kommando findet alle Postscript-Dateien unter dem Heimatverzeichnis und komprimiert diese mit gzip. Das zweite Kommando macht alle Dateien (-type f heißt: Typ (*f*ile) engl(Datei)) für "*g*roup" lesbar. Das dritte Kommando macht alle Unterverzeichnisse für "group" lesbar. Dazu muß für Verzeichnisse (-type d heißt: "Typ *d*irectory", engl(Verzeichnis)) auch e*X*ecute-Erlaubnis gegeben sein.

Man kann mit find auch Dateien löschen. Mit

```
find ~ -name core -type f -exec rm -f {} \;
```

werden alle cores vom Heimatverzeichnis ausgehend gelöscht[13]. Vorsicht! Beim Vergessen der Klammern  wird das Ergebnis unvorhersagbar. Hier sollte man vorsichtshalber sich immer die ersten gefundenen Dateien testweise anzeigen lassen (mit "Ctrl-C" kann man ja abbrechen)

[13] Das sind Dateien, die entstehen, wenn ein Programm durch einen numerischen Fehler abbricht

```
find Verzeichnis -name core -type f -exec ls -l {} \;
```

und dann das ls -l durch rm -f ersetzen. Das find-Kommando ist sehr vielseitig. Weitere Optionen dazu kann man in der Manpage nachlesen, die in diesem Fall auch Beispiele enthält.

## 8.4 Das "Linken" von Dateien

Um große Programme oder Dateien nicht doppelt halten zu müssen, existiert in Unix die Möglichkeit, Dateien zu verbinden (Linken), d.h. einmal existiert die Datei real und an einer anderen Stelle zeigt ein "Link" auf diese real existierende Datei. Mit

```
ln -s Real_existierende_Datei Link
```

kann man das Linken einer Datei durchführen. Bei der "real existierenden Datei" sollte der Pfad angegeben werden. Dabei ist egal, wie die gelinkte Datei heißt, sinnvollerweise wählt man den Namen jedoch gleich. Die Option "s" bedeutet *s*ymbolic, was bedeutet, daß dieser Link auch über Dateisysteme durchgeführt werden kann. Existiert z.B. eine Datei auf einer Workstation, bewirkt der Link auf einer Cray auf diese Datei, daß man auch auf dieser auf die Datei zugreifen und sie verändern kann. Mit dem Listing-Kommando "ls -l" wird der Link übersichtlich angezeigt. Löscht man den Link, so wird die real existierende Datei nicht gelöscht.

Ohne die Option "s" wird aus dem Link ein "Hardlink", der nur auf einer Partition einer Festplatte erlaubt ist. Sind Dateien identisch, so kann man dieses mit der Option "i" des ls-Kommandos erkennen. Haben die Dateien gleiche i-node-Nummern (1. Spalte), so sind sie gleich. Ein Hardlink wird z.B. in Systemen häufig zwischen den Editoren "vi" und "ed" durchgeführt, bei denen das Binärprogramm gleich ist, der Aufruf des Namens "vi" bzw. "ed" jedoch dazu führt, daß das Programm die Eigenschaften wechselt.

Bei der Archivierung von Dateien (vgl. nächstes Kapitel) muß man bei der Benutzung von Links aufpassen, daß dieser als Link und nicht als real existierende Datei archiviert wird. Dies kann Probleme mit dem Speicherplatz verursachen.

## 8.5 Archivierung/Kompression von Dateien

Unix besitzt eine ganze Reihe von Werkzeugen zum Archivieren und Komprimieren von Dateien. Kompression von Dateien ist z.B. sehr sinnvoll bei Postscript-Dateien, bei denen der Kompressionsgrad von etwa 75% erreicht wird. Einfach zu merken ist das Komando "compress", wie z.B.

```
compress Datei
```

, wobei "Datei" zu "Datei.Z" komprimiert wird.

```
decompress Datei.Z
```

macht die Sache wieder rückgängig. Einen weiteren Komprimierer stellt das Werkzeug "gzip" dar, das eine Datei "Datei.gz" erzeugt. Den umgekehrten Prozess erreicht man "gzip -d" (*d*ecompress). Gzip ist wesentlich flexibler wie compress, hier kann man den Kompressionsgrad angeben, komprimierte Dateien aneinanderhängen oder auf die Standardausgabe schreiben. Letzteres kann man z.B. dazu nutzen, eine komprimierte Postscript-Datei zu drucken, ohne sie zu dekomprimieren und dann wieder zu komprimieren.

```
gzip -dc Datei.ps.gz | lpr -Pdruckername
```

erledigt dies auf elegante Weise. [14] Hier wird die Standardausgabe direkt in die Druckausgabe hineingeleitet, da die Option "c" verursacht, daß die Datei nicht in eine andere entpackt wird, sondern ohne Angabe von "| lpr -Pdruckername" auf der Standardausgabe erscheinen würde.

Archivierung unter Unix erfolgt i.a. mit "tar" (*T*ape *ar*chiver, engl(Band-Archivierer)), der aber nicht nur auf Bänder schreiben kann. Sehr häufig wird dieses Werkzeug dazu benutzt, Dateistrukturen (d.h. Verzeichnisse mit Unterverzeichnissen und darin enthaltenen Dateien) in eine Datei zusammenzupacken. Hierbei findet *keine* Kompression statt.

```
tar cvf Irgendeinname.tar Verzeichnis
```

packt das Verzeichnis "Verzeichnis" in die Datei "Irgendeinnname.tar". Die Endung ".tar" ist zwar nicht notwendig, sollte jedoch logischerweise verwendet werden, damit man weiß, um was für eine Datei es sich handelt. Die Optionen "cvf" bedeuten *C*reate (erzeuge) *V*erbose (Zeige an) *F*ile, wobei hier ausnahmsweise kein Minuszeichen vor den Optionen stehen muß. Das Entpacken der Dateisysteme funktioniert mit

```
tar xvf Irgendeinname.tar
```

, wobei x für e*X*tract (extrahiere) steht.

Sehr häufig werden archivierte Dateien mit "gzip" komprimiert, womit dann Dateien mit der Doppelendung ".tar.gz" entstehen. Um solche Dateien auch unter DOS erkennen zu können, wurde diese Doppelendung zu ".tgz" zusammengefaßt.

Werden in der Unixwelt komplette Programmsysteme an Nutzer weitergegeben, so handelt es sich meistens genau um solch eine "Dateien.tar.gz", also um ein gepacktes Verzeichnis. Dieses kann man wieder entpacken und mit dem "tar"-Kommando die Dateistruktur wieder herstellen. Das benötigt jedoch einen enormen Speicherplatz, da die ".tar"-Dateien, wie bereits erwähnt, nicht komprimiert sind.

Dies kann man jedoch, genauso wie beim Drucken der Postscript-Datei, folgendermaßen umgehen:

[14] Dies ist u.a. auch deshalb sinnvoll, da die Dekompression auch um eine Größenordnung schneller als die Kompression ist.

```
gzip -dc Datei.tar.gz  | tar xvf -
gzip -dc Datei.tgz | tar xvf -
```

## 8.6 Drucken von Dateien

Dateien schickt man relativ einfach folgendermaßen auf einen Drucker:

```
lpr -Pdrucker Datei
```

Welche Drucker auf einer Maschine ansprechbar sind (oder wenn einem der Druckername entfallen ist), steht in der datei "/etc/printcap"[15], die man mit "more" oder "less" einsehen kann. Das Drucken einer Datei funktioniert i.a. so, daß die Datei in ein Verzeichnis kopiert wird, von dem aus sie gedruckt wird. Dieses Verzeichnis hat jedoch normalerweise nur begrenzt Platz. Ist der Drucker direkt an die Maschine angeschlossen, so kann man den Umstand dadurch umgehen, daß man mit

```
lpr -s -Pdrucker Datei
```

druckt, wodurch ein "Link" angelegt wird und die Größe der Datei keine Rolle spielt. Im allgemeinen sind Drucker aber über das Netz angeschlossen, sodaß dies nicht funktionieren muß.

Bei Unix-Rechnern werden meistens Postscript-Drucker verwendet. Es existieren Programme wie "enscript" oder "a2ps", die ascii-Dateien, wie z.B. Listings von Programmen, in Postscript-Format wandeln und dabei gleichzeitig Seitennummern, Zeilennummern und Datum auf die jeweilige Seite drucken. Beispielsweise kann man dann mit

```
a2ps Datei | lpr -Pdrucker
```

die Datei auf den Drucker schicken.

## 8.7 Bedienung eines Bandlaufwerkes

Wenn man den Dateinamen "Irgendeinname.tar" (vgl. voriges Kapitel) durch einen Bandnamen (bitte dazu die Manpage "tar" einsehen), ein typischer Bandname wäre z.B. "/dev/nrmt1" (*N*o *R*ewind *M*agnetic *T*ape) ersetzt, schreibt man dann aber tatsächlich auf ein Band. Die Datei "Irgendeinname.tar" entspricht also dem Band. Bänder, falls nicht vorformatiert, werden auch nur zu Beginn einmal formatiert und dann weitere Dateien angehängt oder das Bestehende überschrieben. Bei mehrfacher Formatierung können mehrere Anfangsmarken entstehen, die beim Beschreiben Probleme bereiten. Zum Spulen von Bändern gibt es das Kommando "mt" (*M*agnetic *T*ape).

[15] Unter aix heißt die Datei "/etc/qconfig"

```
mt -f /dev/nrmt1 rewind
```

spult das Band zurück, die Option "unload" veranlaßt das Bandlaufwerk, das Band herauszugeben und

```
mt -f /def/nrmt1 fsf 1
```

(fsf= *f*orward *s*kip *f*ile) springt eine Marke vor.

Hat man sehr häufig mit *einem* Bandlaufwerk zu tun, so kann man z.B. mit

```
export TAPE=/dev/nrmt1
```

ein Voreinstellungsband setzen, das immer dann angesprochen wird, wenn man beim "tar"-Kommando keinen Namen angibt.

## 8.8 awk

Hier liegt wieder eine Wortspielerei vor, awk bedeutet nämlich im Deutschen eigentlich "häßlich". Die drei Buchstaben stehen jedoch für die Anfangsbuchstaben der Autoren dieses Werkzeuges. awk ist zumindest dann das Mittel, wenn Dateien spaltenweise zu verarbeiten sind. Die Dokumentation zu awk, was im Prinzip eine komplette Programmiersprache darstellt, umfaßt mehrere 100 Seiten. awk kann Rechnen, Feldverarbeitung, Regexps's, Zeichenverarbeitung, Unterprogrammaufruf und vieles mehr. Ich möchte mich daher auf wenige Beispiele beschränken.

Will man z.B. aus einer mehrspaltigen Datei, in der die Spalten durch Leerzeichen getrennt sind, die erste und dritte Spalte extrahieren und die dritte und dann die erste in eine andere Datei schreiben, wobei die Spalten mit einem Doppelpunkt getrennt werden sollen, so lautet der Befehl folgendermaßen:

```
awk 'BEGIN {OFS = ":"};{print $3, $1}' Eingabedatei > Ausgabedatei
```

OFS bedeutet hier *O*utput *F*ield *S*eparator, der, wie erwähnt, aus Doppelpunkten bestehen soll. Sollen in der Ausgabedatei auch nur Leerzeichen als Trennung stehen, läßt man alles von "BEGIN" bis zum Strichpunkt einfach weg. Und "print $3, $1" bedeutet einfach, daß in der Ausgabe zuerst die dritte und dann die erste Spalte ausgegeben werden soll. Mit formatierter Ausgabe, Spalten zu je fünf Zeichen, sieht das Gleiche so aus:

```
awk  'BEGIN {IFS = ":"};{printf "%5s %5s\n" $3, $1}' Eing.datei > Ausg.datei
```

Hier steht zwischen den "`"..."`" das Format, wobei hier der Zeilenumbruch \n (*n*ewline) explizit angegeben werden muß. Während also in der Eingabedatei die Spalten mittels ":" getrennt sind, geschieht dies in der Ausgabe durch Leerzeichen.

## 8.9 sed

Wo awk erwähnt wird, ist der sed meist nicht weit. Das liegt daran, daß beide Werkzeuge ähnlich sind. Die Stärke der sed (*S*tream *ED*itor) liegt darin, Dateien an bestimmten Stellen zu ändern. Ein solcher Aufruf würde so aussehen:

```
sed -e "s/timestep=xx/timestep=5/g" Eingabedatei > Ausgabedatei
```

wobei hier in der Eingabedatei die Buchstabenfolge timestep=xx durch timestep=5 ersetzt wird, dieses aber in eine Datei mit dem Namen "Ausgabedatei" umgelenkt wird. Häufig steht anstelle der "5" z.B. die Variable "$nummer", wobei die Variable "nummer", auf die mit "$nummer" zugegriffen wird, in einer Schleife hoch- oder heruntergezählt wird.

# 9 Voreinstellungsdateien

Das System Unix hat für die meisten Programme und Shells systemübergreifende Voreinstellungen. Bleibt man jedoch auf einer einzigen Maschine, so ist es lästig, immer wieder die gleichen "länglichen" Befehle und die gleichen Optionen zu einzutippen. Dies kann man umgehen, indem man sich selbst Dateien einrichtet, für die das System Voreinstellungsdateien benutzt. Diese kann man dann nach Bedarf modifizieren, und ins Heimatverzeichnis legen. Wichtige Voreinstellungsdateien sind:

```
.login       .profile
.kshrc      (.cshrc)
.plan        .rhosts
.xinitrc     .mwmrc
```

Alle Dateien und Verzeichnisse mit dem Beginn "." werden normalerweise mit dem "ls"-Kommando nicht dargestellt. Sie würden unter DOS als "versteckte Dateien" bezeichnet. Um sie zu sehen, muß man "ls -a"(*a*ll) angeben. Weiterhin werden für viele Anwendungen Voreinstellungen über diese "Punktdateien" eingelesen. Die Voreinstellungsdatei für den Editor emacs hat z.B. die Bezeichnung ".emacs", Netscape legt ein ganzes Verzeichnis ".netscape" für Voreinstellungsdateien an.

## 9.1 Die Dateien .login und .kshrc (.cshrc)

Zunächst kann man in der Datei .login oder .kshrc (.cshrc) Abkürzungen (sogenannte alias) definieren, die z.B. folgendermaßen aussehen:

```
alias caea01 'xauth extract - $DISPLAY | rsh caea1 xauth merge - ;
xrsh caea01 xterm -display $DISPLAY
```

[16] Hier werden 2 Kommandos durch ein Semikolon getrennt. Die Syntax dieses alias ist i.a. geringfügig unterschiedlich, in den Dateien sind aber i.a. kurze Beispiele vorhanden, in denen man die Stellung der Kommata und Anführungszeichen sehen kann. Vorsicht: Es werden zwei verschiedene "einfache" Anführungszeichen unterschieden, dies ist das "nach rechts umfallende. "´", was im Folgenden benutzt wird und das "nach links umfallende" "`", was seltener benötigt wird und im Kapitel "Skripten" einmal auftaucht. Ganz übliche kurze aliases sind:

```
alias ll 'ls -l'
alias h  'history'
alias apropos  'man -k'
alias ls 'ls -F'
```

Falls man temporär z.B. die Zuweisung von "ls -F" auf "ls"(letztes Alias der Beispielliste) nicht benötigt, gibt man ganz einfach

```
unalias ls
```

ein. In der Korn-Shell wird weiterhin noch in der Datei .kshrc häufig ein Abbilden der Pfeiltasten (Cursortasten) vorgenommen, da auf Kommandos, die in der Kommandohistorie stehen, normalerweise nur mit gewissen Fingerverrenkungen zugegriffen werden kann. Hier gibt der Systemadministrator sicherlich ein Beispiel.

## 9.2 Die Datei .plan

Im ".plan" stehen die Adresse und Telefonnummer des Benutzers, was mit dem "finger"-Kommando innerhalb eines Systems abgefragt werden kann.

## 9.3 Die Datei .rhosts

Mit dem ".rhosts" kann man ohne Paßwort auf anderen Maschinen einloggen und mittels "rcp"(*R*emote *CoP*y) Daten zwischen Maschinen einfach kopieren. In der Datei existiert auf der "Anderen Maschine" und in dieser steht hintereinander zeilenweise der Maschinenname und der Benutzer, dem der "remote"-Zugriff auf das Kennzeichen erlaubt werden soll.

Die Funktion erprobt man am besten, indem man ein einfaches Komando wie z.B. "ls" auf der anderen Maschine absetzt, also

[16] Das Kommando "alias" entspricht in etwa dem "Doskey" unter DOS, wobei hier erreicht wird, daß bei Absetzen des Kommandos "caea01" das lange Kommando in den Anführungszeichen abgesetzt wird.

```
rsh Andere_Maschine -l Benutzername_dort ls
```

(*r*emote *sh*ell). Funktioniert dies, kann man davon ausgehen, daß dies auch für Remote-Kopien mittels "rcp" (vgl. Kapitel 7) oder das Schicken eines X-Fensters mittels "rsh" der Fall ist.

## 9.4 Die Datei .netrc

Das .netrc wird für ein automatisches ftp benötigt. In dieser Datei muß daher ein Paßwort stehen, sie darf für andere Benutzer nicht lesbar sein (`chmod go-r .netrc`). Die Benutzung der Datei .netrc ist jedoch aus Sicherheitsgründen bedenklich, der Dateitransfer mittels remote-copy ist vorzuziehen.

## 9.5 Die Datei .xinitrc

Die Dateien .xinitrc (Initialisieren) und .mwmrc (Motif Window Manager) sind für Voreinstellungen des X-Window-Systems zuständig. Ersteres enthält, wie das X-Window-System startet, zweiteres, was in den Menues enthalten ist, die erscheinen, wenn man mit den Maustasten auf den Hintergrund klickt.
Hier noch ein Tip: Wenn man mit "Ctrl" und den Maustasten auf ein X-Window klickt, erscheinen ebenfalls Menues. Hiermit kann man interaktiv z.B. die Fontgrößen einstellen.

# 10 Beispielskripten

Alle diese Skripten (was einfach eine Anzahl von hintereinandergestellten Befehlen ist und einem Script.bat unter DOS entspricht) müssen mit

```
chmod u+x Script
```

ausführbar gemacht werden. Folgende Skripten sind in der C-Shell geschrieben, wie man aus der ersten Zeile ersieht. In der ersten Zeile soll *immer* die Shell (d.h. der Kommandointerpreter) stehen, da es bei der Portierung auf andere Rechner Ärger gibt.

## 10.1 print_file

Script "print_file" zum Drucken einer Ascii-Datei auf einem Postscript-Drucker. Das Skript kann man natürlich nennen, wie man will.

# bedeutet Kommentar (außer in der ersten Zeile), "foreach" (in der Korn-Shell "for") läßt für alle Argumente in den runden Klammern bis zum Kommando "end" alles wiederholt ablaufen. Alle Argumente der Befehlszeile werden für $⋆ eingesetzt.

```
#!/bin/csh                          # Der Kommandointerpreter ist die C-Shell
#!/bin/csh -xv
# Wechsle erste and zweite Zeile -> debugging des Skriptes

# generelle Einstellungen:
set ps_printer=lp
###################################
if ($#argv < 1) then                 # muss mind. 1 Argument haben
   echo "Usage: $0 file"             # sonst Fehlermeldung
   exit 1
endif

set print_user=‘whoami‘    # Besetzt die Variable print_user mit
                           #  der Ausgabe des hinteren Kommandos
set currdir=$cwd           #  Besetze die Variable currdir mit einer
                           #  anderen Variable

foreach file ($*)
  a2ps -nn  $currdir/$file  > $currdir/$file.$$.ps   # Wandelt Textdatei
              # in PS-Datei um, $$ steht fuer eine 5-stellige Prozessnummer,
```

```
            # um nicht zufaellig etwas zu ueberschreiben
  lpr -P$ps_printer $currdir/$file.$$.ps             # Druckt
  echo "   $file   is printed on Postscript-Printer P$ps_printer"  # Meldung
  rm -f $currdir/$file.$$.ps                         # entfernt die PS-Datei
end

unset currdir
unset ps_printer
```

Dieses Skript soll nicht besonders elegant aussehen, es soll lediglich darstellen, wie so etwas aussehen kann.

## 10.2 Setzen von Variablen über ein Skript

Versucht man ein Skript zu schreiben, über das Environment-Variablen gesetzt werden sollen, funktioniert dies aus einem einfachen Grunde nicht: Startet man das Skript so wird eine neue Shell (Tochtershell) gestartet, in dem die Variablen gültig sind. Stoppt das Skript, befindet man sich wieder in der Muttershell dieses Skriptes. Wie bereits erwähnt, können keine Variablen in Muttershells exportiert werden. Das Problem kann man lösen, indem man

```
. Skript
```

bzw. in des C-Shell bzw. Tc-Shell

```
source Skript
```

eingibt.

Zur Anwendung kommt dies, wenn man z.B. in der Datei ".kshrc" herumexperimentiert und nicht jedes Mal neu einloggen will. Um sie ablaufen zu lassen gibt man `. .kshrc` ein. Analog benutzt man in der C-Shell bzw. Tc-Shell `source .cshrc`.

# 11 Das Job-Queueing-System

Queues (Warteschlangen) werden dazu benutzt, auf Hochleistungsrechnern die Prozessoren gleichmäßig und möglichst effizient auszulasten. Dieses Kapitel taucht an dieser Stelle auf, da man üblicherweise ein kleines Skript schreiben muß, um das System zu benutzen. Praktisch alle Befehle fangen mit einem "q" an, da dieses phonetisch gleich "queue" ist, was wiederum ein Wortspiel ist. Auf jeder Maschine sind die grundsätzlichen Befehle zur Bedienung der Queueing-Systems gleich, die Optionen aber unterschiedlich.

Ein Job wird grundsätzlich mit

```
qsub Skript
```

gestartet (*sub*mitted engl(abgeschickt)). Hier wird ein Skript benoetigt, um z.B. ein Programm mit dem Namen "MAIN" zu starten, das nicht ausführbar sein muß[17]:

```
#!/bin/ksh
#@$-lT 2:00:00
#@$-lM 500mb
#
ja
cd ~/Arbeit
make
./MAIN > Out
ja -st
```

Auch hier zeigt die erste Zeile den Kommandointerpreter an. Die zweite und dritte Zeile ist spezifisch und beispielhaft für irgendeine Maschine. Normalerweise bedeutet"#" in Skripten einen Kommentar. Durch das Anhängen eines maschinenspezifischen Zeichens wird das System dazu gebracht, die Zeile zu interpretieren. Hier wird die maximale Rechenzeit und der maximale Speicherbedarf angegeben. In welche "Rechenschlange" sich der Job einreiht, entscheidet dann das System aufgrund dieser zwei Angaben. "ja" bedeutet hier *J*ob *A*ccount und ist ein cray-spezifisches Kommando, wobei dieses Kommando auf jeder Maschine einen anderen Namen hat. Es gibt genau an, was innerhalb dieses Skriptes wieviel Zeit und Speicher benötigt. Weiterhin erkennt man in diesem Job-Skript, daß ein Job einen kompletten Einlogvorgang darstellt, der im Heimatverzeichnis beginnt. Man muß deshalb zunächst in das Arbeitsverzeichnis wechseln und kann dann von dort Befehle absetzen. Hier wird ein komplettes "make" durchgeführt und dann das Pragramm "MAIN" gestartet, das wie beim interaktiven Arbeiten in die Datei "Out" schreibt. Mit dem "ja -n"wird "Job Account" dazu gebracht, die Informationen auszugeben. Diese Ausgaben werden dann in "Skript.oxxxxx"geschrieben, Fehler in "Skript.exxxx", wobei xxxxx die Jobnummer darstellt, die beim

```
qstat
```

Kommando ausgegeben wird (*q*ueue *Stat*us). Mit

```
qdel -9 xxxxx
```

[17]Um so ein Skript zu testen, macht man es aber üblicherweise ausführbar und läßt es kurz einfach unter Angabe des Namens 'anlaufen'

kann man den Job abbrechen, wobei das Signal "-9" nur dann benötigt wird, wenn der Job schon läuft. Einige Programme empfangen auch andere Signale. "-15" bedeutet, daß das Programm beendet werden, aber vorher herausschreiben soll. Benötigen alle im Skript ausgeführten Operationen mehr Rechenzeit oder mehr Speicher, wie im Job-Skript angefordert, bricht das Skript bei Erreichen der Schranke mit Fehlermeldung ab. Zum angeforderten Speicherplatz ist zu bemerken, daß dieser nicht nur den vom Programm "MAIN" bzw. "make" angeforderten Speicherplatz betrifft. Zusätzlich muß der von der Shell, in der das Skript abläuft (hier ksh), angeforderte Speicherplatz berücksichtigt werden.