SOFTWARE

WEGA

DIENSTPROGRAMME
BAND B

EAW *electronic*

P8000

Version 1.1 (2008-04-01)

EAW-electronic P8000

W E G A - S o f t w a r e

Dienstprogramme

(Band-D)

Diese Dokumentation wurde von einem Kollektiv des Kombinates

VEB ELEKTRO-APPARATE-WERKE
BERLIN-TREPTOW "FRIEDRICH EBERT"

erarbeitet.

Herausgeber:

Kombinat
VEB ELEKTRO-APPARATE-WERKE
BERLIN-TREPTOW "FRIEDRICH EBERT"
Hoffmannstrasse 15-26
BERLIN
1193

WAE/03-0205-02

Ausgabe: 12/88

Die vorliegende Dokumentation unterliegt nicht dem Aenderungsdienst.

Spezielle Hinweise zum aktuellen Stand der Softwarepakete befinden sich in README-Dateien auf den entsprechenden Vertriebsdisketten.

Dieser Band-D enthaelt folgende Unterlagen:

Teil 1: CSHELL
WEGA-Kommandointerpreter

Teil 2: C-ISAM
Indexsequentielle Zugriffsmethode

Teil 3: SCCS
Source Code Control System

Gesamtinhaltsverzeichnis Seite

Teil 1: CSHELL-Kommandointerpreter

Teil 2: C-ISAM Indexsequentielle Zugriffsmethode

Hinweise des Lesers zu diesem Dokumentationsband

Wir sind staendig bemueht, unsere Unterlagen auf einem qualitativ hochwertigen Stand zu halten. Sollten Sie deshalb Hinweise zur Verbesserung dieses Dokumentationsbandes bzw. zur Beseitigung von Fehlern haben, so bitten wir Sie, diesen Fragebogen auszufuellen und an uns zurueckzusenden.

Titel des Dokumentationsbandes: WEGA-Dienstprogramme (Band-D)

Ihr Name / Tel.-Nr.:

Name und Anschrift des Betriebes:

Genuegt diese Dokumentation Ihren Anspruechen? ja / nein
Falls nein, warum nicht?

Was wuerde diese Dokumentation verbessern?

Sonstige Hinweise:

Fehler innerhalb dieser Dokumentation:

Unsere Anschrift: Kombinat VEB ELEKTRO-APPARATE-WERKE
BERLIN-TREPTOW "FRIEDRICH EBERT"
Abteilung Basissoftware
Hoffmannstrasse 15-26
BERLIN
1193

Notizen:

Notizen:

C-SHELL

WEGA-Kommandointerpreter

Vorwort

Diese Unterlage ist eine Zusammenfassung und Erweiterung von Informationen des WEGA-Programmierhandbuchs und verschiedener unterschiedlicher Quellen.

Inhaltsverzeichnis Seite

1. Einfuehrung in C-Shell

1.1. Was ist eine Shell?

Eine Shell ist ein interaktives Programm, das Kommandos interpretiert und ausfuehrt. Sie ist ein Software-Interface zwischen den auf dem Terminal geschriebenen Kommandos und den Funktionen des Computers. Die Shell bestimmt ebenfalls die Eigenschaften jeder arbeitenden Umgebung. Die fuer die Shell gesetzten Optionen und Variablen werden die fuer jedes Kommando festgesetzten Optionen und Variablen. Auf login initiiert das arbeitende WEGA-System einen C-Shell-Prozess fuer den Nutzer. Dieser interaktive Prozess ist die Login-Shell des Nutzer, er ist der Ausgangsprozess fuer alle nachfolgenden Prozesse (als Child-Prozesse bekannt). Die Login-Shell ist eine Umgebung, die die grundlegenden Parameter der Interaktion zwischen dem Benutzer und dem arbeitenden System genau bezeichnet.

Die Login-Shell definiert die Home-Directory fuer den Nutzer; sie definiert den Suchpfad fuer jedes Kommando; das dem Nutzer gegebene Prompt, um anzuzeigen, dass das System fuer ein weiteres Kommando bereit ist; die zu verwendende Shell und den in Gebrauch befindlichen Terminaltyp.

Wenn an der Umgebung keine Modifikation ausgefuehrt wurde, dann werden Standardwerte fuer die Eigenschaften dieser Umgebung verwendet. Jeder Nutzer kann sich diese Umgebung je nach individuellem Geschmack, Bedarf und/oder Verwendung einrichten. Die Eigenschaften der Umgebung koennen durch das Festlegen neuer Shell-Variablen oder das Aendern der existierenden Shell-Variablen veraendert werden.

WEGA unterhaelt 2 Shell-Programme, jedes mit seinen eigenen Kommandos und Variablen. Das Standard-Shell-Programm fuer WEGA ist die C-Shell. Das zweite Shell-Programm ist die Bourne-Shell (bekannt als "die Shell"; siehe sh(1) und Einfuehrung in Shell in "WEGA-Dienstprogramme".

Die C-Shell ist wegen ihrer erhoehten Kommandstruktur und ihrer konventionellen C-Programmiersyntax etwas wirksamer als die Bourne-Shell. Beide Shell-Programme koennen als Login-Shell dienen und beide koennen vom Terminal interaktiv aufgerufen werden.

1.2. Konventionen dieser Dokumentation

command(1)

Ein Wort, das von einer einzelnen in Klammern stehenden Ziffer gefolgt wird, wie in ls(1), ist ein Kommando; das Wort ist das Kommando und die in Klammern stehende Ziffer "(1)" verweist auf den Dokumentationsteil im WEGA-Programmierhandbuch. In diesem Fall wird auf Teil 1 des WEGA-Programmierhandbuches verwiesen. Die Kommandos sind im Programmierhandbuch

innerhalb jedes Teils alphabetisch geordnet.

Anfuehrungszeichen

Spezielle Zeichen werden wie bei "?" in Anfuehrungszeichen ausgegeben, um sie vom Dokumententext abzuheben.

Beispiele:

Jeder Nutzer kann in Abhaengigkeit von der Struktur unterschiedliche Daten von den Beispielen erhalten. Ein Beispiel eines Kommandos nimmt folgendes Format an:

```
command operator filename
```

Das ist vom Hauptteil des Textes eingerueckt und von dem darueber und darunter stehenden Text durch eine Leerzeile getrennt. In einigen Faellen kann ein einzelnes Beispiel im Hauptteil des Textes erscheinen.

Variablennamen

Wenn Variablen entweder auf dem Terminal oder in einem Shell-Script aufgerufen werden, geht ihnen ein Dollarzeichen "$" wie im Kommando

```
echo $PATH
```

voraus; wenn Variablen im Text erlaeutert werden, wird auf sie ohne das Praefixzeichen (Dollar) verwiesen (siehe echo(1) und Teil 10).

Die Syntax

Die Syntaxangabe demonstriert die Syntax eines gegebenen Kommandos, indem sie veranschaulicht, wo die Optionen, Flags und Schluesselwoerter (falls vorhanden) plaziert und wo der Dateiname (falls vorhanden) plaziert ist. Das folgende Beispiel demonstriert einen "Syntax"-Teil:

Syntax

```
date > filename
```

Wenn mehr als eine Zeile erscheint, bedeutet das, dass das erlaeuterte Objekt auf mehr als eine Art und Weise verwendet werden kann.

Eckige Klammern []

In der Syntaxzeile zeigen eckige Klammern an, dass die von ihnen eingeschlossenen Informationen optional sind; sie koennen im Kommando erscheinen, sind jedoch nicht obligatorisch.

In dem Beispiel

```
echo [-n] string
```

ist das "-n"-Flag optional. Es kann erscheinen, aber auch weggelassen werden. Im eigentlichen eingegebenen Kommando werden nur die Optionen, nicht die eckigen Klammern geschrieben.

Drei Punkte ...

Drei Punkte in einer Reihe "..." zeigen an, dass das einleitende Element beliebige Male wiederholt werden kann. Im folgenden Beispiel:

```
command {item1, item2, ....}
```

zeigt die Auslassung an, dass innerhalb der geschweiften Klammern eine beliebige Anzahl von items erscheinen kann.

Standardteil:

Im Standardteil wird der fuer eine Variable implizit gesetzte Wert angegeben.

Siehe auch:

Wenn ein Hinweis auf Teil 3 gegeben wird - Dateinamensubstitution, verweist er auf Teil 3 dieser Dokumentation. Der "Siehe auch:"-Teil verweist auch auf andere Dokumentationen in "WEGA-Software-Dienstprogramme" und andere Handbuecher der P8000-Dokumentation.

Grosser Anfangsbuchstabe

Grosse Buchstaben werden bei Eigennamen und am Anfang eines neuen Satzes verwendet. Wenn ein Satz mit einem Kommandonamen beginnt, wird dieser Kommandoname gross geschrieben, auch wenn alle in WEGA vorhandenen Kommandos mit kleinen Buchstaben geschrieben werden muessen. Der Name der Umgebungsvariablen wie PATH wird durch Konventionen nur mit Grossbuchstaben geschrieben.

Ausdruckunterbrechung

Einige Ausdruecke bestehen aus zwei Woertern, muessen auf Grund der Art und Weise, wie der Computer die Freiraeume interpretiert, aber in einem Wort geschrieben werden. In solchen Faellen kann der Ausdruck als zwei durch einen Punkt oder einen Unterstrich (anstelle eines Leerzeichens) getrennte Woerter dargestellt werden, wie bei

```
command.1 oder READ_ME
```

2. C-Shell-Kommandoeingabbe

Kommandos werden in den Computer nach einem "prompt" geschrieben.

2.1. Das WEGA-Prompt

Ein "prompt" ist ein vom Computer gegebenes Zeichen, dass er bereit ist, Kommandos anzunehmen. Im WEGA-System ist das Standardprompt eines Nutzers ein Prozentzeichen "%".

2.2. Kommandosyntax

Viele Kommandos im WEGA-System bestehen aus einem einzelnen Wort, das von einem "RETURN" Zeichen beendet wird. Diese sind als einfache Kommandos bekannt und werden mit der folgenden Syntax eingegeben:

```
command
```

Ein Beispiel fuer ein einfaches Kommando stellt das date-Kommando dar, das wie folgt eingegeben wird:

```
date
```

und ein Ergebniss, wie

```
TUE NOV 23 14:14:35 MEZ 1982
```

erzeugt.

Die meisten Kommandos koennen modifiziert werden, um mehr und bessere Informationen zu liefern. Die Modifikation erfolgt durch mehrere Argumente in Form von Optionen, Flags, Schluesselworten oder Dateinamen. Kommandos mit Argumenten werden mit folgender Syntax angegeben:

```
command option flag Schluesselwoerter filename
```

Jedes Kommando bestimmt seine eigenen syntaktischen Anforderungen, d.h., der Programmierer des Programms schreibt die syntaktischen Anforderungen in den Hauptteil des Programms. Einige Programme erfordern das Beginnen einer Option, eines Flags oder Schluesselwortes mit einem Minuszeichen "-". Das ls(1) Programm verlangt, dass die Optionen mit einem Minuszeichen beginnen. Die "l" Option liefert eine lange Auflistung der Dateien; sie wird geschrieben als

```
ls -l
```

Andere Programme machen das Minuszeichen optional. Die tar(1) Programmoptionen verwenden das am Anfang stehende Minuszeichen nicht. Die "t" Option liefert das Inhaltsverzeichnis

fuer die Dateien im Archiv; sie wird geschrieben als

```
tar t
```

Einige Programme, wie das tar-Programm, verlangen, dass eine Option, ein Flag oder ein Schluesselwort das zweite Argument des Kommandos ist; andere, wie das ls-Programm, machen die Argumente optional. Mehrere Woerter in einem Kommando sind durch Freiraeume (Leerzeichen oder Tabs) oder Semikolon getrennt, wobei das erste Wort die Handlung anzeigt und die restlichen Woerter als Argumente dienen, wie bei

```
ls -l
```

wo das "-l" Flag ein Argument des ls-Kommando darstellt, das dem Programm mitteilt, eine lange Auflistung zu liefern.

2.3. Einfache Kommandos

Syntax:

```
command
```

Ein Kommando ist eine Instruktion an den Computer. Ein einfaches Kommando besteht aus einem oder mehreren Zeichen, die auf das Computerterminal nach dem "prompt" geschrieben werden. Das Kommando wird mit einem "RETURN"-Zeichen beendet. Ein Kommando besteht mindestens aus einem Wort, das eine auszufuehrende Handlung spezifiziert. Z.B. ist

```
ls
```

das Kommando, das eine Liste der Dateien der aktuellen Directory erzeugt. Siehe ls(1). Das ls-Kommando erzielt Ergebnisse von folgendem Format:

```
csh.01  csh.03  csh.05  csh.07  csh.9A  csh.9T
csh.02  csh.04  csh.06  csh.08  csh.9B  temp
```

Jeder Name verweist auf eine Datei oder eine Directory in der aktuellen Directory.

2.4. Zusammengesetzte Kommandos

Syntax:

```
command1; command2
```

Kommandofolgen koennen durch ein Semikolon getrennt und dann aufeinanderfolgend ausgefuehrt werden, wie z.B.

```
ls; who; pwd; date
```

Das erzielte Ergebniss ist dann:

```
csh.01  csh.03  csh.05  csh.07  csh.9A  csh.9T
csh.02  csh.04  csh.06  csh.08  csh.9B  temp
patty   tty0    Nov 23 08:04
deck    tty2    Nov 23 09:38
carol   tty8    Nov 23 08:17
craig   tty9    Nov 23 08:36
/z/deck/Util/New.csh
Tue Nov 23 14:14:35 MEZ 1982
```

Siehe ls(1), who(1), pwd(1) und date(1).

2.5. Ein im Hintergrund laufendes Kommando

Syntax:

```
command &
```

Da einige Kommandos mehrere Minuten bis zur Beendigung benoetigen, stellt das WEGA-System einen Mechanismus fuer das gleichzeitige Laufen mehrerer Kommandos bereit; das ist bekannt als das Laufen der Kommandos im Hintergrund. Die Steuerung des Terminals wird an den Nutzer zurueckgegeben, waehrend die Kommandos ihre Ausfuehrung fortsetzen. Ein Kommando laeuft im Hintergrund, wenn es vom Ampersand ("&") gefolgt wird. Fehlerdiagnostiken, sofern nicht anderes festgelegt ist, werden an die Standard-Fehlerausgabe - das Terminal - ausgegeben. Z.B. kann die Uebersetzung eines C-Programms, test.c durch

```
cc test.c &
```

erreicht werden. (siehe cc (1) im WEGA-Programmierhandbuch bezueglich weiterer Informationen ueber den C-Compiler).

Die Ueberpruefung, ob der Uebersetzungsprozess laeuft, kann mit dem ps(1)-Kommando erfolgen. Der gesamte Ablauf wuerde folgende Form annehmen:

```
% cc test.c &
2999
% ps
 PID TTY TIME CMD
1309 2   0:29 csh
2999 2   0:00 cc
3002 2   0:03 ps
```

Das cc test.c & Kommando startet einen Uebersetzungslauf. Eine Prozessidentifikationsziffer erscheint auf dem Bildschirm, unmittelbar vom naechsten Prompt gefolgt. Das ps-Kommando wird eingegeben, sobald das Prompt erscheint, auch wenn der vorangegangene Prozess noch laeuft, und es werden die gegenwaertigen Prozesse angezeigt.

2.6. Ein in einer Subshell laufendes Kommando

Syntax:

```
(command)
```

In runden Klammern stehende Kommandos werden stets in einer Subshell ausgefuehrt. Im folgenden Beispiel hindert das in einer Subshell laufende Kommando cd an der Beeinflussung einer gegenwaertigen Shell. Folglich gibt das Kommando

```
(cd; pwd)
```

den Namen der Home-Directory aus, ohne die gegenwaertig aktuelle zu veraendern, indes veraendert das Komando

```
cd; pwd
```

die gegenwaertig aktuelle Directory in die Home-Directory und gibt dann den Namen jener Directory aus. Diese Kommandostruktur ist als voruebergehende Flucht vor der gegenwaertig aktuellen Directory nuetzlich.

Siehe auch:

```
cd(1)
```

2.7. Konditionale Kommandooperatoren

Syntax:

```
command.1 && command.2
command.1 || command.2
```

Ein Operator ist ein Symbol, das die Arbeitsweise eines Kommandos veraendert. In mathematischen Kommandos (wie bc(1) und dc(1)) sind "+", "-", "*" und "/" die mathematischen Standardoperatoren fuer die "Addition, Subtraktion, Multiplikation bzw. Division". Die beiden folgenden Operatoren sind "logische" Operatoren, der logische "and"-Operator (&&) und der logische "or"-Operator (||). Diese Operatoren trennen zwei Kommandos auf einer einzelnen Zeile und stellen fest, ob das eine oder das andere, beide oder kein Kommando ausgefuehrt wird. Die Bestimmung basiert darauf, ob das erste Kommando erfolgreich ausgefuehrt wurde oder nicht. Wenn das erste Kommando ohne einen Fehler beendet wird, heisst dies, dass es erfolgreich ausgefuehrt wurde und gibt einen Statuscode "0" zurueck. Wenn es nicht erfolgreich ausgefuehrt wurde, gibt es einen Exitstatus ungleich Null, normalerweise eine "1" zurueck.

Ungluecklicherweise bedeutet in C-Shell "0" "false" und "1" "true"; folglich scheint die Syntax der Operatoren irgendwie

umgekehrt zu sein, wenn sie auf Kommandos angewendet werden.

Die folgende Tabelle veranschaulicht die Ergebnisse dieser Operatoren:

erstes auszufuehrendes Kommando	Operator	zweites auszufuehrendes Kommando
yes	|| "or"	= yes
no	|| "or"	= no
no	&& "and"	= yes
yes	&& "and"	= no

In dem Kommando

```
ls || date
```

fuehrt die C-Shell sowohl ls als auch date aus. In dem Kommando

```
bogus.command || fake.command
```

versucht die C-Shell bogus.command auszufuehren, unternimmt bei Misslingen nicht den Versuch, fake.command auszufuehren. In dem Kommando

```
bogus.command && ls
```

versucht die C-Shell bogus.command auszufuehren und fuehrt bei Misslingen ls aus. Schliesslich fuehrt die C-Shell im Kommando

```
ls && bogus.command
```

das ls-Kommando aus und versucht bei Gelingen nicht

```
bogus.command auszufuehren.
```

Siehe auch: Abschn. 7: Shell-Variablen - die Statusvariable Abschn. 8.4.1. - C-Shell-Skripte - And und Or-Operatoren

2.8. Die Kommandosubstitution

Syntax:

```
`command`
```

Ein Kommando in "`" wird ausgefuehrt und die Ausgabe des Kommandos ersetzt das Kommando selbst. Z.B. erzeugt das Kommando

```
echo "Today is `date`"
```

eine entsprechende Ausgabe wie:

```
Today is Fri Dec 10 17:16:00 MEZ 1982
```

2.9. Ein-/Ausgabesteuerung

Das System hat drei Kommunikationskanaele zwischen dem Nutzer und dem Computer, einen Standard-Eingabekanal und zwei Ausgabekanaele, die Standard-Ausgabe und die Fehlerausgabe. Implizit (sofern nicht anders spezifiziert) erfolgt die Eingabe durch die Terminaltastatur. Das ist die Standard-Eingabe. Die Ausgabe zum Terminal-Bildschirm und ist die Standard-Ausgabe. Jeder aus der Ausfuehrung eines Programms resultierende Fehler erzeugt eine Fehlernachricht auf dem Terminalbildschirm. Das ist die Standard-Fehlerausgabe.

Es gibt Situationen, in denen die Eingabe aus einigen anderen Quellen in ein Programm gelangen muss (z.B. aus einer Datei). Desgleichen kann ein Beduerfnis nach Umlenkung der Ausgabe und der Fehlernachrichten bestehen. Obwohl die Standard-Eingabe, Ausgabe und Fehlerkanaele implizit der Tastatur und dem Terminalbildschirm zugeordnet sind, koennen sie durch die Verwendung von groesser als (">") und kleiner als ("<") veraendert werden. Die folgenden Abschnitte erlaeutern diese Umlenkung.

2.9.1. Die Umlenkung der Eingabe - <

Syntax:

```
command < com.list
```

Die Datei com.list wird eroeffnet und ihr Inhalt wird fuer command als Eingabe verwendet. Bei

```
wc < text.file
```

wird text.file als Eingabe fuer das wordcount-Kommando wc(1) verwendet. Dies erzielt Ergebnisse in folgendem Format:

```
474 2055 12623
```

Die erste Zahl ist die Anzahl der Zeilen, die zweite Zahl ist die Anzahl der Woerter und die letzte Zahl ist die Anzahl der Zeichen in der Datei.

Ein anderes Beispiel ist die Bildung einer com.file genannten Datei mit ex(1)-Editor-Kommandos, z.B. mit Kommandos, die alle fuehrenden Freiraeume und alle Leerzeilen entfernen. Das Kommando

```
ex test < com.file
```

ruft den ex-Editor fuer die Datei test auf, doch anstatt die Editor-Kommandos von der Standard-Eingabe (der Tastatur) zu nehmen, werden die Kommandos von der Datei com.file gelesen.

Siehe auch: wc(1) ex(1) und WEGA-Programmierhandbuch

2.9.2. Interne Dateneingabe - <<

Syntax:

```
command << label
```

Ein Shell-Script ist eine Datei von Kommandos, die von der Shell einzeln ausgefuehrt werden, so als wuerden sie auf dem Terminal eingegeben werden. In den meisten Faellen entnehmen die Kommandos in einem Shell-Script die Eingabe einem Terminal oder anderen Dateien, doch in einigen Faellen kann es auch notwendig sein, die Eingabe dem Shell-Script selbst zu entnehmen. (Siehe Teil 3 - Das Csh-Kommando und die C-Shell-Scripte). Das doppelte kleiner-als-Symbol gestattet es einem Shell-Script, die Daten vom Innern seines eigenen Textes zu entnehmen. Das ist im Zusammenhang mit Editor-Scripts am gebraeuchlichsten; beachte den folgenden Shell-Script:

```
% deblank -- remove blank lines
ex test << 'EOF'
g/^S/d
w
q
'EOF'
```

In dem Beispiel bedeutet die Zeile

```
ex text << 'EOF'
```

dass die Datei test mit dem ex-Editor bearbeitet wird und dass die Kommandoeingabe fuer ex vom Hauptteil des Shell-Scripts stammt. Die "<< 'EOF'"-Bezeichnung bedeutet, dass die Daten "bis zu 'EOF'" als Eingabe genommen werden. Die einfachen Anfuehrungsstriche um " 'EOF' " verhindern jegliche Variablenexpansion. (Siehe Teil 7 - Shell-Variablen)

Siehe auch: ex (1) und WEGA-Programmierhandbuch

2.9.3. Umlenkung der Ausgabe - >

Syntax:

```
command > test1
```

Die Datei test1 wird als Ausgabe verwendet. Sollte die Datei nicht existieren, wird sie erzeugt. Existiert sie, wird sie ueberschrieben und ihre vorheriger Inhalt wird geloescht. Das Kommando

```
ls -l > test2
```

steckt die Ausgabe des ls -l Kommandos in die Datei test2.

Anmerkung: Eine Datei wird stets geloescht (falls sie existiert), bevor eine neue Information in sie geschrieben wird.

Das Kommando

```
cat file1 > file2
```

loescht alle Informationen in file2, bevor es den Inhalt von file1 in sie steckt.

Anmerkung: Um das versehentliche Loeschen einer Datei zu verhindern, kann die Noclobber-Variable mit dem Kommando:

```
set noclobber
```

gesetzt werden.

Siehe auch: Abschnitt 29.6. "Ueberschreiben Noclobber" fuer Beispiele und Abschnitt 7.1. fuer mehr Details ueber vordefinierte C-Shell-Variablen (noclobber) und cat(1)

2.9.4. Anhaengen an eine Datei >>

Syntax:

```
command >> file
```

Die doppelte groesser-als-Zeichenkonstruktion (">>") haengt die Ausgabe von command an das Ende der Datei an, anstatt die Datei zuerst zu loeschen. Wenn file nicht existiert, wird sie automatisch erzeugt. Wenn z.B. file1 aus den 3 Zeilen besteht:

```
Now is the time
for all good people
to come to the aid of their party
```

und file2 aus der einen Zeile

```
The quick brown fox jumps over the lazy dog
```

besteht, so erzeugt das Kommando

```
cat file1 >> file2
```

eine neue Datei file2, die die 4 folgenden Zeilen enthaelt:

```
The quick brown fox jumps over the lazy dog
Now is the time
for all good people
to come to the aid of their party
```

Beachte, dass der Inhalt von file1 an das Ende von file2 angehaengt wird.

2.9.5. Umlenkung der Standard-Fehlerausgabe - >&

Syntax:

```
command >& file
```

Das von einem Ampersand gefolgte groesser-als-Zeichen (">&") leitet die Fehlernachrichten zusammen mit der Standard-Ausgabe in die spezifizierte Datei. Gegeben sei das Kommando

```
cat bogus.file > new.file
```

wenn bogus.file nicht existiert, so ergibt es den Fehler

```
cat: cannot open bogus.file
```

Unter Nutzung des groesser-als-Zeichens plus Ampersandkonstruktion

```
cat bogus.file >& new.file
```

weden alle Fehlernachrichten zur Datei new.file umgelenkt und diese kann wie jede andere Textdatei bearbeitet werden.

2.9.6. Ueberschreiben Noclobber - >!

Syntax:

```
command >! file
```

Wenn file existiert und die C-Shell-Variable noclobber gesetzt ist, misslingt ein Kommando, das die einfache Form der Ausgabe (">") verwendet und hat eine Fehlernachricht zur Folge. Im Kommando

```
cat file1 > file2
```

wird in diesem Fall (wenn file2 existiert) die Fehlernachricht

```
file2: File exists.
```

erzeugt. Die noclobber-Variable verhindert die unbeabsichtigte

Zerstoerung von Dateien. In diesem Falle kann das groesser-als-Zeichen zusammen mit dem Ausrufungszeichen (">!") zur Unterdrueckung dieser Kontrolle verwendet werden.

Das Kommando:

```
cat file1 >! file2
```

gelingt nun, file2 wird von file1 ueberschrieben, auch wenn file2 existiert und noclobber gesetzt ist. Das Kommando erzeugt keine Fehlernachricht.

2.9.7. Ausgabe, Fehler und Noclobber - >&!

Syntax:

```
command >&! file
```

Diese Form vereint die Ampersand- ("&") und Ausrufungszeichen-("!")-Konstruktionen. Wie oben erwaehnt, lenkt sie die Fehlerausgabe zur Datei file und ueberschreibt den Inhalt von file unabhaengig vom Wert der noclobber-Variable.

2.9.8. Anhaengen und Standard-Fehlerausgabe - >>&

Syntax:

```
command >>& file
```

Diese Form vereinigt das "Anhaengen an das Ende einer Datei" (die doppelte groesser-als-Konstruktion) mit der "Umlenkung" der Standard-Fehlerausgabe (Ampersand) und haengt die Ausgabe von command und alle Fehlernachrichten an das Ende der Datei file an.

2.9.9. Anhaengen und Noclobber - >>!

Syntax:

```
command >>! file
```

Diese Form verknuepft das "Anhaengen an das Ende einer Datei" (die doppelte groesser-als-Konstruktion) mit dem "Uebergehen der Noclobbervariablen" (Ausrufungszeichen) und haengt die Ausgabe von command ungeachtet der noclobber-Variable (ob sie gesetzt ist) an das Ende der Datei an.

2.9.10. Anhaengen, Noclobber und Fehlerausgabe - >>&!

Syntax:

```
command >>&! file
```

Haengt die Ausgabe an das Ende der Datei an. Falls die noclobber-Variable gesetzt ist, wird sie ignoriert und die Standard-Fehlerausgabe wird ebenfalls angehaengt.

2.10. Pipes

Syntax:

```
command | command
```

Eine Folge von einfachen Kommandos, durch einen vertikalen Balken "|", auch als ein pipe bekannt, getrennt, bildet eine Pipeline. Die Ausgabe jedes Kommandos in einer Pipeline wird zur Eingabe des naechsten.

Das Beispiel

```
who|grep chuck
```

nimmt die Ausgabe des who(1) Kommandos und leitet es durch das Kommando grep(1), um die Zeilen mit der Zeichenkette chuck zu extrahieren.

Dieses Kommando entspricht der Umlenkung der Ausgabe des who-Kommandos zu einer vorlaeufigen Datei, dem darauffolgenden Laufen des Kommandos grep chuck mit jener vorlaeufigen Datei als Eingabe und dem Entfernen der vorlaeufigen Datei, wie in der Abfolge

```
who > temp
grep chuck temp
rm temp
```

Das Kommando

```
who
```

erzeugt eine Liste in folgendem Format:

```
karen   tty0    Nov 23 08:04
chuck   tty2    Nov 23 09:38
mike    tty6    Nov 23 14:50
carol   tty8    Nov 23 08:17
george  tty9    Nov 23 08:36
```

Die Ausgabe wird mit dem Kommando

```
who > temp
```

zu einer Datei umgelenkt.

Die Zeile mit dem Wort cruck wird mit dem Kommando

```
grep chuck temp
```

extrahiert, um die Ausgabe

```
chuck tty2 Nov 23 09:38
```

zu erzeugen und die vorlaeufige Datei wird mit dem Kommando

```
rm temp
```

entfernt. All das kann mit dem Pipe-Mechanismus leichter ausgefuehrt werden, wie im Kommando

```
who|grep chuck
```

das die gewuenschte Ausgabe erzeugt:

```
chuck tty2 Nov 23 09:38.
```

Die folgenden Tabellen stellen eine Zusammenfassung der Kommandostrukturen und der I/O-Umlenkungszeichen dar:

Zusammenfassung der Kommandostruktur:

command	Simple command
command flag	Command with an option argument
command filename	Command with a filename argument
command;command	Compound command
command &	Running a command in background
(command)	Running a command in a subshell
`command`	Command substitution

Zusammenfassung der I/O-Umlenkung:

Symbol	Bedeutung
<	Take input from
<<	Take input up to
>	Redirect output
>&	Redirect output and error
>!	Redirect output -- override noclobber if set
>&!	Redirect output and error; override noclobber
>>	Append output
>>&	Append output and error
>>!	Append output and override no clobber
>>&!	Append output and error; override noclobber
|	Output from first command is input for second

3. Die Substitution von Dateinamen

3.1. Zeichen fuer Dateinamen

Die C-Shell bietet eine Methode der abgekuerzten Kommunikation. Im Falle der Dateinamen bietet die Shell eine Anzahl von speziellen Zeichen (bekannt als Metazeichen, Magiczeichen oder Wild-Card-Zeichen), die zur Bildung von Datei- und Directorynamen entsprechend spezifischer Regeln benutzt werden koennen.

Der Prozess bezieht sich auf Mustervergleiche und Dateinamenexpansion. Wenn ein Metazeichen verwendet wird, werden die Dateinamen und Directories abgetastet, um zu sehen, ob die durch das Metazeichen gesetzte Muster zu jenen Datei- und/oder Directorynamen passt.

Die wahren Eigenschaften eines Metazeichens offenbaren sich mit dem echo Kommando. Das Kommando:

```
echo metacharacter
```

wird das Muster, das durch das Metazeichen vertreten wird, zurueckgeben. Der Mustervergleich findet nach folgenden Regeln statt:

Stern - *

Syntax:

```
command *
```

Der Stern ist ein sehr wirksames Zeichen. Es ist die Kurzschrift fuer "jedes Muster" in Datei- und Directorynamen. Z.B. listet das Kommando

```
ls *
```

alle Dateien und Directories aus. Das Kommando

```
ls a*
```

gibt alle Dateien und Directories die mit dem Buchstaben "a" beginnen aus. Schliesslich listet das Kommando

```
ls /z/deck/U*/*
```

alle Dateien und Directories unter dem Directory oder den Directories in /z/deck, die mit dem Buchstaben "U" beginnen aus.

Fragezeichen - ?

Syntax:

```
command ?
```

Das Fragezeichen ist die Shell-Kurzschrift fuer "alle Einzelzeichen". Folglich registriert das Kommando

```
ls ???
```

alle Dateien und Directories mit Namen aus genau drei Zeichen. Die Dateien

```
abc dog ps1
```

z.B. passen zum String "???", waehrend es bei den folgenden nicht der Fall ist:

```
file1 jj make.p test.c
```

Die Zeichen koennen Buchstaben, Zahlen oder irgendein anderes legitimes (Nicht-Metazeichen) Dateinamenzeichen sein. Entsprechend wird das Kommando

```
ls csh.??
```

eine Liste von Dateien und Directories erzeugen, die mit "csh." beginnen und mit zwei Zeichen enden. Z.B. passen die Dateien

```
csh.01 csh.02 csh.03
```

waehrend die Dateien

```
csh.1 csh.test csh.A
```

nicht passen.

Muster/Bereich - [A-Z]

Syntax:

```
command [begin of range - end of range]
```

Die eckigen Klammern definieren einen Zeichenbereich, der zu jedem Einzelzeichen passt, das in den angegebenen Bereich (alphabetisch oder numerisch) faellt.

```
ls csh.0[1-9]
```

listet alle Dateien die mit "csh.0" beginnen und mit der Ziffer 1 bis 9 enden; d.h. csh.01 csh.02 csh.03 csh.04 csh.05 csh.06 csh.07 csh.08 csh.09. Das Kommando

```
ls csh.[1-3][1-9]
```

wird alle Dateien von "csh.11" bis "csh.39" auslisten. Andere Zeichen koennen ebenfalls im Bereich spezifiziert

sein. Die Einordnung erfolgt nach dem ASCII-Numerierungsschema. Mit Ausnahme spezieller Zeichen laeuft die ASCII-Anordnungsfolge von 0-9, A-Z und a-z. Folglich wird der gesamte Bereich alphanumerischer Zeichen (und einiger nicht-alpha-numerischer Zeichen) vom Ausdruck 0-z eingenommen.

Siehe auch: ascii(7)

Abkuerzung - {A,B,C}

Syntax:

```
command {item.1,item.2,...}
```

Die geschweiften Klammern verweisen auf eine Auswahl von Zeichen oder Strings - jedes von ihnen kann zu einer Datei oder Directory passen oder nicht. Das Kommando

```
ls file.a{b,c,d}e
```

listet die Dateien

```
file.abe file.ace file.ade
```

aus, falls sie existieren. Dementsprechend passt das Kommando

```
ls /usr/man/man1/{csh,ls,dog}.1
```

zu den Dateien

```
/usr/man/man1/csh.1
/usr/man/man1/ls.1
/usr/man/man1/dog.1
```

Die Auswahl der Zeichen oder Strings muss nicht in einem Bereich oder in einer bestimmten Reihenfolge sein. Sie muessen nicht von der selben Laenge sein, sie muessen durch Kommas (ohne Leerzeichen) getrennt sein.

Tilde - ~

Syntax:

```
command ~
command ~user.name
```

Das Tilde dient als Abkuerzung fuer die Home-Directory des Nutzers. Das Kommando

```
ls -l ~
```

wird auf die Home-Directory des Nutzers ausgedehnt:

```
ls -l /z/deck
```

Wenn der Tilde ein Name folgt, sucht die Shell nach einem Nutzer mit jenem Namen und substituiert deren Home-Directory; Folglich wird das Kommando

```
ls -l ~carol
```

auf:

```
ls -l /z/carol
```

ausgedehnt. Wenn der Tilde "~" ein anderer als ein in der Kennwort-Datei befindlicher Name oder ein Schraegstrich "/" folgt, wird es als buchstabengetreue Tilde durch die Shell verwendet. Im Kommando

```
cat ~filename
```

z.B. sucht die Shell nach einer Datei mit dem genauen Namen "~filename".

Eine Zusammenfassung der Zeichen der Dateinamenexpansion erscheint in der folgenden Tabelle.

Siehe auch: Teil 10.1 - Umgebungsvariablen

Zusammenfassung der Substitutionszeichen zur Dateinamensbildung:

*	Any string
?	Any single character
[A-Z]	Any character in the range A to Z
{A,B,C}	Any element from the set A, B, or C
~	Home directory or user name.

3.2. Der Metazeichensatz

Die Dateinamenexpansion ist ein Beispiel dafuer, wie Shell die Sonderzeichen benutzt. Jedes der folgenden Zeichen besitzt fuer Shell und/oder das Betriebssystem eine spezielle Bedeutung.

Die folgende Aufzaehlung beschreibt die Sonderzeichen in der Reihenfolge ihres Erscheinens im ASCII-Zeichensatz und ihrer Bedeutung in C-Shell, der History-Funktion und dem Betriebssystem.

Leerzeichen ' '

Syntax:

```
command space arguments
command tab arguments
```

Das Leerzeichen trennt Woerter in Kommandos. Wenn ein zusammengesetztes Kommando geschrieben wird, verwendet die Shell Freiraeume - space oder tab Zeichen, um die verschiedenen Komponenten auseinanderhalten zu koennen. Das Kommando

```
ls -l /tmp /z /usr/spool
```

wird von der Shell verstanden, da die Komponententeile durch die abgrenzenden Leerzeichen in erkennbare Teile zerfallen.

Ausrufungszeichen ' ! '

Syntax:

```
!character, number or string
```

Das Ausrufungszeichen wird in der Shell verwendet, um einen Aufruf des history-Mechanismus der Shell zu initiieren (siehe Teil 4). Zuvor geschriebene Kommandos werden von eins beginnend numeriert und in der History-Liste aufbewahrt. Von dort koennen sie auf verschiedene Weise wieder aktiviert werden. Das Kommando

```
!ls
```

durchsucht die History-Liste rueckwaerts, um das juengste Kommando, das mit dem String ls beginnt, zu finden und fuehrt es aus. Das Kommando

```
!3
```

durchsucht die History-Liste rueckwaerts, um das Kommando Nummer "3" zu finden und auszufuehren. Das Ausrufungszeichen wird ebenfalls in einer Vielzahl von Programmen zum Aufruf der Shell verwendet.

Siehe auch: Teil 4 - Die History-Funktion

Doppeltes Anfuehrungszeichen ' " '

Syntax:

```
command "string"
```

Das doppelte Anfuehrungszeichen (") wird auf beiden Seiten eines Ausdrucks verwendet, um die Expansion verschiedener anderer spezieller Zeichen zu verhindern. Das Kommando

```
echo *
```

dehnt das Stern-Metazeichen (*) auf die gesamten Datei- und

Directory-Namen in der gegenwaertigen Directory aus, waehrend das Kommando

```
echo "*"
```

bloss den Stern ausgibt.

Siehe auch: Teil 3.3 - Das Setzen von Anfuehrungszeichen - Unterdruecken der Expansion von Metazeichen.

Das Doppelkreuz ' # '

Syntax:

```
# comment
```

Das Doppelkreuz wird als erstes Zeichen in einem Shell-Script verwendet, um anzuzeigen, dass die C-Shell zur Ausfuehrung des Skripts verwendet werden soll. Das Doppelkreuz wird ebenfalls im Hauptteil eines Shell-Skripts verwendet, um einen Kommentar zu beginnen - das Doppelkreuz sagt der C-Shell, dass sie den Rest der Zeile ignorieren soll. Innerhalb des Hauptteils eines Shell-Skripts wird die Zeile

```
# this is a comment line
```

von der Shell ignoriert.

Siehe auch: Teil 8 - Shell-Skripte

Das Dollarzeichen ' $ '

Syntax:

```
command $variable
command !$
```

Das Dollarzeichen besitzt in einer Reihe von Umstaenden besondere Bedeutung. Wenn es mit einem Variablennamen, wie in

```
echo $prompt
```

verwendet wird, verweist es auf die Shell-Variable "prompt". Wenn es mit dem History-Mechanismus, wie im Kommando

```
!$
```

verwendet wird, verweist es auf das letzte Element des letzten Kommandos. Sollte das letzte Kommando

```
ls -l /z/joe/file.1
```

sein, erzeugt das Kommando

```
cat !$
```

die Ergebnisse, als waere es Kommando

```
cat /z/joe/file.l
```

gewesen.

Siehe auch: Teil 7 - Shell-Variablen
Teil 8 - Die History-Funktion

Ampersand ' & '

Syntax:

```
command &
!N:s/x/&/
command && command
```

Im ersten Fall veranlasst das Ampersand, das zum Ende des Kommandos angegeben wird, dass das Kommando im Hintergrund lauft. Das Kommando

```
cc test.c &
```

setzt einen Kompilationsprozess in Gang und gibt sofort ein prompt zurueck. Dann kann ein anderes Kommando eingegeben werden, auch waehrend der Kompilationsprozess noch laeuft.

Im zweiten Fall vertritt das Ampersand den "gerade substituierten String" in einer History-Substitution.

Im dritten Fall wird das Ampersand als logischer "and" Operator in konditionalen Kommandos verwendet.

Siehe auch:
Abschn. 2.5. - Ein im Hintergrund laufendes Kommando
Abschn. 4. - Die History-Funktion
Abschn. 2.7. - Konditionale Kommandooperatoren.

das einfache Anfuehrungszeichen " ' "

Syntax:

```
command 'string'
```

Das einfache Anfuehrungszeichen ist ein weiterer Kennzeichnungsmechanismus, der von Shell verwendet wird, um die Expansion spezieller Zeichen zu hemmen oder zu unterdruecken. Das Kommando

```
echo $prompt
```

erzeugt

```
%
```

waehrend die Verwendung von doppelten Anfuehrungszeichen, wie im Kommando

```
echo "$prompt"
```

ebenfalls

```
%
```

erzeugt.

Um die Expansion des String "$prompt" durch die Shell zu unterdruecken, muessen einfache Anfuehrungszeichen verwendet werden. Das Kommando

```
echo '$prompt'
```

erzeugt

```
$prompt.
```

Siehe auch: Abschn. 3.3 - Unterdruecken der Expansion von Metazeichen

Linke runde Klammer ' ('

Syntax:

```
( command )
foreach variable ( list )
```

Im ersten Fall wird ein in Klammern stehendes Kommando stets in einer Subshell ausgefuehrt. Es ist fast wie ein vorlaeufiger Wechsel in eine andere Arbeitsumgebung, z.B., wenn die gegenwaertige Arbeitsdirectory /tmp ist und das folgende Kommando eingegeben wird:

```
(cd; pwd)
```

erzeugt die C-Shell eine neue C-Shell und fuehrt das Kommando innerhalb jener neuen Shell aus. Die Subshell stirbt und die Steuerung kehrt zur Stammshell zurueck, von der das Kommando gegeben wurde. Die gegenwaertige Arbeitsdirectory bleibt /tmp. Hierin besteht ein wesentlicher Unterschied zum Kommando

```
cd; pwd
```

das die gegenwaertige Arbeitsdirectory in die Home-Directory wechselt.

Im zweiten Fall werden die Klammern verwendet, um eine Wortliste in Shellschleifen abzugrenzen, wie in der anweisung

```
foreach i (1 2 3 4 )
```

Die Klammern zeigen der Shell an, dass die Liste "1 2 3 4" als Steuerungsmechanismus der Schleife verwendet werden soll. Das ist ebenfalls fuer die if, while und switch-Anweisungen, die spaeter behandelt werden, verwendbar.

Siehe auch:

Abschn. 2.6 - Das in einer Subshell laufende Kommando
Abschn. 6 - Struktur der C-Shell-Programmiersprache

Rechte runde Klammer ') '

Syntax:

```
( command )
foreach variable ( list )
while ( expression )
```

Die rechte Klammer beendet den Steuerungsmechanismus einer Schleife oder ein Subshell-Kommando.

Stern ' * '

Syntax:

```
command *
```

Der Stern ist ein Zeichen der Dateinamenexpansion, es passt zu allen Mustern.

Siehe auch: Abschnitt 3.1. Zeichen fuer Dateinamen.

Das Pluszeichen ' + '

Syntax:

```
number + number
variable++
```

In den on-line Rechnern (dc(1) und bc(1)) und den mathematischen Funktionen der C-Shell-Skripte wird das Pluszeichen als Additionsfunktion verwendet. Im Hauptteil eines Shell-Skript weist die Zeile

```
@ x = (6 + 6)
```

der Variablen "x" den Wert 12. Das Pluszeichen wird ebenfalls fuer die Erhoehung des Variablenwertes verwendet, wie in

```
        @ i++
```

die den Wert i jedesmal um 1 erhoeht, wenn die Anweisung ausgefuehrt wird. Im folgenden Shell-Skript wird die Variable i innerhalb einer Schleife erhoeht:

```
        # the name of this file is "test.file"
        @ i=1
        while (1)
                echo $i
                @ i++
        end
```

Es wird mit dem Kommando

```
        csh test.file
```

ausgefuehrt und erzeugt die folgende Ausgabe

```
        1
        2
        3
        4
        5
        6
        7
        .
        .
        .
```

bis eine Unterbrechung (DEL-Taste) erfolgt.

Siehe auch: Abschn. 5.3.9. - Das "At Sign" @

Komma ' , '

Syntax:

```
        command {item1,item2}
```

Das Komma wird zur Abgrenzung von Elementen innerhalb der geschweiften Klammern verwendet. Im Kommando

```
        cat csh.0{3,5,7}
```

sind die Kommas notwendige Trennzeichen der Ziffern 3, 5 und 7.

Siehe auch: bc(1), dc(1)
Abschn. 7 - Shellvariablen

Minus ' - '

Syntax:

```
        number - number
        @ variable--
```

```
command -(flag, option, or key)
```

Wie das Pluszeichen, wird das Minuszeichen als Substraktionsoperator innerhalb der Shell-Skripte verwendet. Es wird ebenfalls zum dekrementieren von Variablen verwendet.

```
# the name of this file is test.file.2
@ i = 15
while (1)
        echo $i
        @ i--
end
```

Der Shell-Skript wird mit dem csh-Kommando, wie im obrigen Beispiel, ausgefuehrt und erzeugt die folgende Ausgabe:

```
15
14
13
12
11
10
9
.
.
.
```

bis einer Unterbrechung (DEL-Taste) erfolgt. Die Verwendung des Minuszeichens ist fuer die Shell von besonderer Wichtigkeit, um Flags, Optionen oder Keys fuer viele Shell-Kommandos anzuzeigen, wie in:

```
ls -l
```

Siehe auch: Abschn. 5.3.9. - Das "At Sign" @
Abschn. 2 - Kommandos der C-Shell

Der Punkt ' . '

Syntax:

```
command .
.filename
```

Obwohl der Punkt keine Funktion der C-Shell im besonderen ist, wird er vom Betriebssystem zur Markierung der gegenwaertigen Arbeitsdirectory verwendet. Das Kommando zum Kopieren (cp(1)) einer Datei in die aktuelle Directory hat folgende Syntax:

```
cp /tmp/karen .
```

Dieses Kommando ist die Kurzschrift des Kommandos

```
cp /tmp/karen (current_working_directory)
```

Wenn der Punkt als erstes Zeichen in einem Dateinamen verwendet wird, laesst er den Dateinamen fuer ein Standard-ls-Kommando unsichtbar werden ("Punkt"-Dateien werden mit der "-a" Option sichtbar, wie in "ls -a").

Im zweiten Fall gibt es mehrere "Punkt-Dateien", die fuer die Shell von besonderer Wichtigkeit sind. Die .login-Datei wird immer dann gelesen, wenn eine neue C-Shell aufgerufen wird und die .exrc-Datei wird vom ex-Editor gelesen, um grundlegende Optionen festzusetzen, etc.

Die mit einem Punkt beginnenden Dateinamen werden nicht mit dem Standard ls-Kommando erfasst, sondern mit der -a (all) Option, wie im Kommando ls -a.

Siehe auch: Teil 9 - C-Shell-Dateien

Punkt-Punkt ' .. '

Syntax:

```
command ..
```

"Punkt-Punkt" wird vom Betriebssystem verwendet, um die parent-directory zu kennzeichnen. Wenn die gegenwaertige Arbeitsdirectory "/z/deck" ist, verwandelt das Kommando

```
cd ..
```

die aktuelle Arbeitsdirectory in "/z". Sowohl Punkt ".", als auch Punkt-Punkt ".." erscheinen mit dem Kommando "ls -a" als Directorynamen.

Siehe auch: Abschn. 9 - C-Shell-Dateien

Schraegstrich ' / '

Syntax:

```
command /path
```

Das Schraegstrich-Zeichen wird als Pfadabgrenzer zur Lokalisierung von Dateien verwendet. Im Pfad

```
/usr/spool/mail/user.name
```

trennen die Schraegstriche Directory- und Dateinamen. Wenn das erste Zeichen in dem Pfadnamen einer Datei ein Schraegstrich ist, beginnt die Shell von der Wurzel des Dateisystems, um die Datei zu lokalisieren. Wenn die gegenwaertige Arbeitsdirectory z.B. /tmp ist, wird das Kommando

```
ls -l /z/paula/temp
```

die Datei mit genau jenem Pfad lokalisieren, waehrend das Kommando

```
ls -l z/paula/temp
```

nach einer Datei namens /tmp/z/paula/temp suchen wird. Wenn das erste Zeichen in einem Pfadnamen kein Schraegstrich ist, beginnt die Shell von der aktuellen Arbeitsdirectory, um die Datei zu ermitteln.

Doppelpunkt ' : '

Syntax:

```
!identifier:modifier
```

In Verbindung mit dem History-Mechanismus wird der Doppelpunkt zur Modifizierung vorangegangener Kommandos verwendet. Das Kommando

```
!1:s/who/date/
```

wird das Kommando Nummer 1 wiederholen und das Wort who durch das Wort date ersetzen.

Siehe auch: Abschn. 4 - Die History Funktion

Semikolon ' ; '

Syntax:

```
command; command
```

Das Semikolon ist ein Kommandobegrenzer. Das Kommando

```
ls; who; pwd; date
```

kann in einer Zeile eingegeben werden, es wird durch Shell zerlegt.

kleiner als ' < '

Syntax:

```
command < file
if (variable < variable)
```

Im ersten Fall wird das kleiner-als-Zeichen zur Umlenkung der Eingabe von file zu command verwendet. Im zweiten Fall verwenden mathematische Operationen dieses Zeichen als Vergleichsoperator "kleiner als", wie im Ausdruck

```
if ($a < $b) then
```

```
        ...
```

Siehe auch:
Abschnitt 2.9.1 - Umlenkung der Eingabe und
Abschnitt 2.9.3 - Umlenkung der Ausgabe

Gleichheitszeichen ' = '

Syntax:

```
        set variable=value
        if (variable == variable) then
```

Shell-Variablen erhalten mit dem set-Kommando einen Wert, dabei wird die im ersten Fall dargestellte Syntax verwendet. Im zweiten Fall verwenden mathematische Operationen innerhalb der Shell-Skripte das doppelte Gleicheitszeichen mit der Bedeutung "ist gleich", wie im Ausdruck

```
        if ($a == $b) then
        ...
```

Siehe auch: Abschn. 7 - Shell-Variablen

Groesser als ' > '

Syntax:

```
        command > file
        if (variable > variable) then
```

Im ersten Fall lenkt das "groesser als"-Zeichen die Ausgabe von command zu file um. Im zweiten Fall ist dieses Zeichen der mathematische Operator "groesser als" innerhalb eines Shell-Skripts, wie in der Zeile:

```
        if ($a > $b) then
        ...
```

Siehe auch:
Abschnitt 2.9.1 - Umlenkung der Eingabe und
Abschn. 2.9.3 - Umlenkung der Ausgabe
Abschn. 7 - Shellvariablen

Fragezeichen ' ? '

Syntax:

```
        command ?
        !?string?
```

Im ersten Fall wird das Fragezeichen als ein Dateinamensubstitutions-Zeichen verwendet und passt zu allen Einzelzeichen in einem Dateinamen. Im obigen Beispiel wird das command Dateinamen mit einem einzigen Zeichen beeinflussen.

Das Fragezeichen wird ebenfalls zur Abgrenzung von Strings im History-Mechanismus verwendet. Das Kommando

```
!?string?
```

extrahiert das juengste string enthaltene Kommando.

Siehe auch: Abschn. 3.1. - Zeichen fuer Dateinamen
Abschn. 4. - Die History-Funktion

At-Sign ' @ '

Syntax:

```
@ variable=number
```

Dieses Zeichen gestattet es Variablen numerische Werte anstelle von string-Werten zu geben, so dass mit ihnen mathematische Operationen ausgefuehrt werden koennen. Das Kommando:

```
@ x=( 6 + 6 )
```

weist der Variablen "x" den Wert 12 zu, waehrend

```
set x=( 6 + 6)
```

"x" die Zeichenkette "6+6" als Wert zuweist.

Ein Leerzeichen muss das At-Sign vom Rest der Variablenzuweisung trennen.

Siehe auch: Abschn. 5.3.9 - Das "At-Sign" @

Linke eckige Klammer ' ['

Syntax:

```
command [range]
$variable[subscript]
```

Die linke und rechte Klammer wird zur Abgrenzung eines Zeichenbereichs verwendet, die zur Musterbildung in Dateinamenexpansionen verwendet werden.

Sie werden ebenfalls verwendet, um eine Komponente einer Variablen mit mehreren Elementen zu isolieren. Wenn die Variable "X" mit dem Kommando

```
set X = (a b c d e)
```

den Wert "a b c d e" erhielt, so wird das dritte Element "c" mit dem Kommando

```
echo $X[3]
```

adressiert.

Siehe auch: Abschn. 3.1. - Zeichen fuer Dateinamen
Abschn. 7.4. - Nutzerdefinierte Variablen-substitutionen

Rechte eckige Klammer '] '

Syntax:

```
command [range]
$variable[subscript]
```

Die rechte eckige Klammer wird zum Abschliessen eines Bereichs- oder Subscriptwerts benutzt.

Siehe auch: Abschn. 3.1. - Zeichen fuer Dateinamen
Abschn. 7.4. - Nutzerdefinierte Variablen-substitutionen

Backslash ' \ '

Syntax:

```
command \metacharacter
```

Das Backslash-Zeichen verhindert die spezielle Bedeutung eines Metazeichens.

Siehe auch:

Abschn. 3.3 - Unterdruecken der Expansion von Metazeichen

Pfeil nach oben ' ^ '

Syntax:

```
!identifer:^
^string1^string2^
```

Der Pfeil nach oben wird durch die History-Funktion verwendet und ist die Kurzschrift fuer "das erste Element". Das Kommando

```
!5:^
```

verweist auf die Argumentnummer 1 im fuenften Kommando.

Im zweiten Fall ist der Pfeil nach oben auch eine Substitutionsmoeglichkeit im History-Mechanismus. Ist ein Kommando

```
ls -l /z/sisan
```

mit einem Tippfehler eingegeben worden, so erzeugt

```
^i^u
```

das naechste richtige Kommando

```
ls -l /z/susan
```

Dieser Mechanismus ist analog zum ":s"-Substitutionsmechanismus der History-Funktion.

Siehe auch: Abschn. 4 - Die History-Funktion

Back quotes ' ` '

Syntax:

```
command `command`
```

In back quotes stehende Kommandos werden ausgefuehrt und die Ausgabe des Kommandos ersetzt das Kommando in back quotes. Z.B. erzeugt das Kommando

```
echo `date`

Wed Dec 8 15:01:48 MEZ 1982
```

Siehe auch: Abschn. 2.8 - Kommandosubstitution

linke geschweifte Klammer '{'

Syntax:

```
command {string1,string2}
command ${variable}word
```

Die linke und rechte geschweifte Klammer grenzen die Abkuerzungen bei der Dateinamenexpansion ab.

Wenn die Variable "X" auf den Wert "4" wurde, hat das folgende Kommando

```
echo ${X}9ers
49ers
```

zur Folge, im Gegensatz zum Kommando

```
echo $X9ers
```

das einen Fehler zur Folge hat:

```
X9ers: Undefined variable
```

Die geschweiften Klammern hindern den umgebenden Text an der Beeinflussung der "X"-Variable.

Siehe auch: Abschn. 3.1 - Zeichen fuer Dateinamen
Abschn. 7. - Shellvariablen

Vertikaler Balken (pipe) ' | '

Syntax:

```
command | command
command || command
```

Der einfache vertikale Balken verhaelt sich wie eine Pipeline, die die Ausgabe von command auf der linken Seite mit der Eingabe von command auf der rechten Seite verknuepft.

Im zweiten Fall wird der Doppelbalkenmechanismus als logischer "or"-Kommandooperator verwendet.

Siehe auch: Abschn. 2.10. - Pipes
Abschn. 2.7. - Konditionale Kommando-operatoren

Tilde ' ~ '

Syntax:

```
command ~
command ~user.name
```

Die Tilde ist ein Dateinamenexpansionszeichen. Sie wird durch die Home-Directory ersetzt.

Siehe auch: Abschn. 3.1. - Zeichen fuer Dateinamen

Die folgende Tabelle ist eine Zusammenfassung der C-Shell Metazeichen.

Zeichen:	Bedeutung:	Kontext:
space	Delimits words	Commands
!	Accesses history	History
"	Quoting mechanism	Commands
#	Comment line	Shell scripts
$	Last element	History
%	String isolation	History
&	Background command	Prompt
&	Pattern substitution	History
&&	Logical "and" operator	Commands
(	Begins string	Command loops
(	Begin subshell	Commands
)	Ends string	Command loops
)	End subshell	Commands
*	All characters	Filenames
+	Addition	Shell scripts
++	Variable incrementer	Shell scripts
,	Range delimiter	Commands
-	Flag	Commands
-	Subtraction	Shell scripts
--	Variable decrementer	Commands
.	Current Directory	Filenames
.	"dot" files	Filenames
/	Path delimiter	Filenames
:	History modifier	History
;	Command delimiter	Commands
<	Input redirect	Commands
=	Equals	Shell scripts
>	Output Redirect	Commands
?	Any single character	Filenames
@	Math operations	Shell scripts
[	Begins range	Filenames
]	Ends range	Filenames
\	Escapes metacharacters	Commands
^	First argument	History
`	Quoting device	Commands
`	Command Substitution	Command
{	Begins abbreviations	Filenames
}	Ends abbreviations	Filenames
|	Pipe mechanism	Commands
||	Logical "or" operator	Commands
~	Home	Filenames

3.3. Unterdruecken der Expansion von Metazeichen

Es gibt Situationen, in denen Metazeichen nicht ausgedehnt werden sollten. Ein Shell-Skript, z.B. das sowohl das Editor-Kommando mit dem Dollarzeichen $ als auch ein Variablenname mit einem Dollarzeichen einschliesst. In diesen Faellen muss das Dollarzeichen im Editor-Kommando in Anfuehrungszeichen gesetzt werden, so dass seine Bedeutung buchstabengetreu von der Shell genommen und nicht ausgedehnt wird.

Es gibt 4 Kennzeichnungsarten, die im WEGA-System verfuegbar sind, wie die folgende Tabelle zeigt:

The backslash	\
Double quotes	"
Right quote	'
The noglob option	set noglob

Fuer die Kennzeichnungsarten gelten folgende Regeln:

Zeichen:	Symbol:	Quotes Variables:	Quotes Filenames:
The backslash	\	yes	yes
Double quotes	"	no	yes
Right quote	'	yes	yes
Noglob	n/a	no	yes

Die folgende Tabelle zeigt die Wirkung der verfuegbaren Kennzeichnungsarten:

Kommando:	\	"	'	noglob
echo *	*	*	*	*
echo $HOME	$HOME	/z/deck	$HOME	/z/deck

Die folgende Tabelle zeigt, welche Zeichen gekennzeichnet werden muessen, wenn sie in Kommandos verwendet werden und falls ihre Bedeutung buchstabengetreu genommen werden soll. Nicht gekennzeichnet werden sie als Kommandooperatoren verwendet oder auf Datei- und Directorynamen ausgedehnt.

ampersand	&
asterisk	*
backslash	\
dollar sign	$
exclamation point	!
greater than sign	>
left brace	{
left bracket	[
left parenthesis	(
single quote mark	'
less than sign	<
question mark	?
quote mark	"
right brace	}
right bracket	]
right parenthesis	)
back quote	`
semi-colon	;
tilde	~
up arrow	^
vertical bar (pipe)	|

4. Die History-Funktion

4.1. Die Kommando-History

Die vom Nutzer eingegebenen Kommandos werden von eins beginnend nacheinander numeriert und im Speicher in einer History-Liste aufbewahrt. Die Groesse der History-Liste wird von der history-Variablen gesteuert (siehe 7.1.4. - History).

Das Kommando

```
set history=15
```

das normal eingegeben werden kann oder in der .cshrc- oder .login-Datei spezifiziert werden kann, legt fest, dass die letzten 15 Kommandos gespeichert werden. Es werden keine Kommandos gespeichert, wenn die history-Variable nicht gesetzt ist. Die history-Variable ist implizit nicht gesetzt.

Das Kommando

```
history
```

zeigt den Inhalt der History-Liste in folgendem Format an.

```
5       vi temp
6       ls
7       more temp
8       history
9       cat csh.01
10      cat csh.01 > temp.2
11      more temp.2
12      who
13      vi temp.2
14      echo $prompt
15      who > temp.3
16      cat temp.3
17      ls -la
18      vi temp.3
19      history
```

Beachte, dass die Liste in diesem Fall die letzten 15 Kommandos enthaelt. Wenn die Anzahl der Kommandos auf der Liste 15 ueberschreitet, faellt das aelteste Kommando (unwiederbringlich) vom Ende der Liste ab. Mit anderen Worten, wenn die history-Variable auf 15 gesetzt ist, stoesst die Kommandonummer 16 die Kommandonummer 1 von der Liste.

Die Kommandos von der History-Liste koennen wieder aufgerufen und manipuliert werden. Das Ausrufungszeichen "!" wird zur Initiierung eines Aufrufs der History-Funktion verwendet.

4.2. Verwendungsformen der History-Funktion

Die folgende Tabelle veranschaulicht die 7 gebraeuchlichsten Anwendungen der History-Funktion. Da diese Funktionen unter verschiedene Kategorien fallen, wird die Erlaeuterung jeder Form in nachfolgenden Abschnitten wiederholt.

Syntax	Erklaerung	Beispiel
!!	Repeat the last command	!!
!n	Repeat command number n	!6
!string	Repeat command starting with string	!ls
!$	Last argument of previous command	ls !$
!*	All arguments except #0	ls !*
^x^y^	Substitute y for x	^wrong^right^
!!:n	Argument number n	ls !15:2

Doppeltes Ausrufungszeichen !!

Syntax:

```
!!
```

Das doppelte Ausrufungszeichen bedeutet "wiederhole das letzte Kommando noch einmal". Das letzte Kommando (der obigen Liste) ist History, das Kommando !! erzeugt den folgenden Austausch:

```
%!!
history
    6       ls
    7       more temp
    8       history
    9       cat csh.01
    10      cat csh.01 > temp.2
    11      more temp.2
    12      who
    13      vi temp.2
    14      echo Sprompt
    15      who > temp.3
    16      cat temp.3
    17      ls -la
    18      vi temp.3
    19      history
    20      history
```

Ausrufungszeichen; Ziffer - !n

Syntax:

```
        !n
```

Ein Ausrufungszeichen und irgendeine Ziffer bedeutet "wiederhole das Kommando Nummer n". Das Kommando

```
        !6
```

wiederholt das Kommando Nummer 6 der History-Liste. In diesem Fall (von der obigen Liste) das Kommando ls.

Ausrufungszeichen; string - !string

Syntax:

```
        !string
```

Ein Ausrufungszeichen und eine Zeichenkette bedeutet "wiederhole das Kommando, das mit string beginnt". Das Kommando

```
        !v
```

wird das letzte Kommando ausfindig machen, das mit dem Buchstaben "v" beginnt. In diesem Fall ist es das Kommando:

```
        vi temp.3
```

Ausrufungszeichen; Dollarzeichen - !$

Syntax:

```
        command !$
```

Zusaetzlich zu den vollstaendigen Kommandozeilen koennen auch Teile davon, die Argumente des Kommandos manipuliert werden. Die Kombination Ausrufungszeichen, Dollarzeichen verweist auf das letzte Argument des vorangegangenen Kommandos. Wenn das vorausgehende Kommando

```
        ls -l /z/hank/temp
```

war, so erzeugt

```
        cat !$
```

das Kommando

```
        cat /z/hank/temp
```

Ausrufungszeichen; Stern - !*

Syntax:

```
        command !*
```

Die Kombination Ausrufungszeichen, Stern bedeutet "vom zweiten Argument bis zum letzten Argument"; wenn das vorausgehende Kommando

```
ls /z/joe/work /z/zubes/art
```

war, so erzeugt

```
cat !*
```

das Kommando

```
cat /z/joe/work /z/zubes/art
```

Doppelter Pfeil - ^string1^ string2^

Syntax:

```
^string1^string2^
```

Der Pfeil funktioniert als ein string-Substitutions-Mechanismus fuer vorangegangene Kommandos. Wird ein falsches Kommando, z.B.

```
cat /z/curl/letter
```

eingegeben, so erzeugt

```
^curl^carol^
```

das gewollte Kommando

```
cat /z/carol/letter
```

Der nachgestellte Pfeil kann weggelassen werden, wenn das letzte Zeichen des string ein RETURN ist.

Doppeltes Ausrufungszeichen; Ziffer - !!:n

Syntax:

```
[command] !identifier:n
```

Der Aufruf der History-Funktion (ein Ausrufungszeichen gefolgt von einem mit dem gewuenschten Kommando verbundenen identifier) gefolgt von einem Doppelpunkt und einer Ziffer ist der Aufruf des numerierten Arguments jenes Kommandos. Z.B. wenn das Kommando

```
ls /z | wc
```

gegeben ist, das alle Dateien und Directories in der Directory /z erfasst und die Ergebnisse durch das word count-command leitet, so erzeugt

```
cd !!:1
```

das Kommando

```
cd /z
```

4.3. Zugriff auf vorangegangene Kommandos

Gespeicherte Kommandos koennen von der History-Liste durch eine Vielzahl von Kommandos wieder aufgerufen und ausgefuehrt werden. All diese Kommandos beginnen mit einem Ausrufungszeichen. Kommandos oder Ereignisse werden in der Reihenfolge ihrer Nummern aufgezeichnet. Jedem Ereigniss kann nachgegangen werden, indem man die Nummer zu einem Teil des prompt macht. Das geschieht durch das Einsetzen eines Ausrufungszeichens (das gekennzeichnet sein muss) "\!" in den prompt-string.

Durch die Zeile

```
set prompt="%\!"
```

(die in die ~/.cshrc- oder ~/.login-Datei eingetragen werden kann) wird fuer das erste Kommando das folgende Prompt erzeugt

```
%1
```

Die Zahl wird bei jedem Kommando um eins anwachsen.

!! wiederholt das unmittelbar letzte Kommando. Im unten stehenden Fall ist das letzte Kommando (Kommando Nummer 5) history.

!n ist die Kurzschrift fuer "wiederhole das Kommando Nummer n".

Wenn die History-Liste z.B. ist:

```
1 ls
2 who
3 pwd
4 date
5 history
```

so kann das pwd(1)-Kommando noch einmal durch Eingabe von

```
!3
```

ausgefuehrt werden.

!-n bedeutet "Fuehre das Kommando aus, das n Kommandos vor dem gegenwaertigen Kommando ist. pwd (das dritte Kommando vor dem gegenwaertigen Kommando) kann auch durch

```
!-3
```

ausgefuehrt werden.

!string
bedeutet "Fuehre das letzte Kommando mit dem Praefix string aus". Die History-Funktion wird die History-Liste zurueck untersuchen und nach dem juengsten Auftreten eines Kommandos, das mit dem Buchstaben (oder dem Buchstabenstring) string beginnt, suchen. Das Kommando

```
!p
```

wird ebenfalls das pwd-Kommando von der oben gezeigten History-Beispielliste erzeugen.

!?string?
wird das letzte Kommando das string enthaelt ausfuehren, das nachgestellte "?" kann weggelassen werden, wenn nichts folgt.

```
!?wd?
```

wird ebenfalls das pwd-Kommando von der History-Beispielliste erzeugen.

Zugriff auf vorangegangene Kommandos

Syntax	Erklaerung	Beispiel
!!	Repeat the last command	!!
!n	Repeat command number n	!3
!-n	Repeat the command n commands back	!-4
!text	Repeat the command starting with the string text	!p
!?text?	Repeat the command containing the string text	!?wd?

4.4. Modifizierung vorangegangener Kommandos

Teile vorangegangener Kommandos koennen isoliert und manipuliert werden. Es ist moeglich, sowohl auf ein einzelnes Kommando, als auch auf einzelne Argumente eines Kommandos Zugriff zu erhalten. Argumente sind von Null beginnend aufeinanderfolgend numeriert und koennen durch die Folge "!n:", gefolgt von einem Argument-Bezeichner, die nachfolgend erklaert werden, ausgewaehlt werden. Z.B. ist das Kommando Nummer 1 gegeben:

```
%1 ls; who; pwd; date
```

Die Argumente werden wie folgt numeriert:

```
0       1       2       3       4       5       6
ls      ;       who     ;       pwd     ;       date
```

Die folgenden Beispiele demonstrieren die Verwendung des History-Mechanismus zum Zugriff auf Argumente eines Kommandos.

!1:n verweist auf das n-te-Argument des Kommandos Nummer 1. Das Kommando

```
!1:1
```

erzeugt und versucht das erste Kommandoargument auszufuehren (in diesem Fall das Semikolon):

```
;
```

!1:^ verweist auf das erste Argument im Kommando Nummer 1. Das Kommando:

```
!1:^
```

erzeugt und versucht auszufuehren das erste Semikolon des Kommandostrings:

```
;
```

!1:$ verweist auf das letzte Argument des Kommandos Nummer 1. Das Kommando

```
!1:$
```

erzeugt und fuehrt aus

```
date
```

!1:n-m
ist ein Bereich von Argumenten von Nummer n bis Nummer m. Das Kommando

```
!1:2-4
```

erzeugt und fuehrt aus das 2.; 3. und 4. Kommandoargument; who, das Semikolon und pwd:

```
who; pwd.
```

!1:-n
kennzeichnet den Argumentenbereich von Nummer 0 bis Nummer n. Das Kommando

```
!1:-3
```

erzeugt und fuehrt aus das nullte, 1., 2. und 3. Argument des ersten Kommandos:

```
ls; who;
```

!1:* kennzeichnet den Argumentenbereich von Nummer 1 bis zum letzten Argument oder Nichts, wenn es nur ein Argument gibt. Das Kommando

```
!1:*
```

erzeugt und fuehrt das 1., 2., 3., 4., 5. und 6. Argument des Kommandos Nummer 1 aus:

```
; who; pwd; date
```

!1:n*
kennzeichnet den Argumentenbereich von Nummer n bis zum letzten Argument. Das Kommando:

```
!1:2*
```

erzeugt und fuehrt das 2., 3., 4., 5. und 6. Argument aus

```
who; pwd; date
```

!1:n-
wie !1:n*, nur wird das letzte Argument (das Argument "$") weggelassen. Es verkleinert den Argumentenbereich von Nummer n bis zum vorletzten Argument. Das Kommando

```
!1:2-
```

erzeugt und fuehrt das 2., 3., 4. und 5. Argument des Kommandos Nummer 1 aus.

```
who; pwd;
```

Zugriff auf vorherige Kommandowoerter

Syntax	Erlaeuterung	Beispiel
!N:n	Command N, argument n	11:2
!N:^	Command N, first argument	!1:^
!N:$	Command N, last argument	!1:$
!N:n-m	Command N, argument n through m	!1:3-5
!N:-n	Command N, argument 0 through n	!1:-3
!N:*	Command N, argument 1 tho the last	!1:*
!N:n*	Command N, argument n to the last	!1:3*
!N:n-	Command N, argument n to last-1	!1:3-

4.5. Modifizierung vorangegangener Kommandowoerter

Zusaetzlich zum Aufrufen und Modifizieren von Kommandos der History-Liste und zum Aufrufen und Modifizieren von Argumenten innerhalb einzelner Kommandos koennen Argumententeile auch getrennt aufgerufen und manipuliert werden. Ist das Kommando

```
%1 ls -l /z/deck/util/cshell/csh.01
```

gegeben, so sind die folgenden Modifizierer wiefolgt definiert:

:h Entferne eine nachgestellte Pfadnamenkomponente, belasse den Kopfteil. Das Kommando

```
!1:h
```

erzeugt

```
ls -l /z/deck/util/cshell
```

:r Entferne eine nachgestellte Punkt-Komponente (".xxx"), belasse den Namen des Wurzelwortes

```
!1:r
```

erzeugt

```
ls -l /z/deck/util/cshell/csh
```

:t Entferne alle fuehrenden Pfadnamenkomponenten, belasse das hintere Ende. Das Kommando

```
!1:t
```

erzeugt

```
ls -l csh.01
```

:p Schreibe das neue Kommando, doch fuehre es nicht aus. Das Kommando:

```
!1:p
```

erzeugt

```
ls -l /z/deck/util/cshell/csh.01
```

doch fuehrt es nicht aus.

:s/string1/string2
Setze string2 anstelle von string1 ein; das nachgestellte "/" kann weggelassen werden, wenn eine neue

Zeile folgt; "/" ist kein eindeutiger Abgrenzer. Das Kommando

```
!1:s/cshell/shell
```

erzeugt und fuehrt aus:

```
ls -l /z/deck/util/shell/csh.01
```

:q Kennzeichnet die substituierten Argumente, um weitere Substitutionen zu unterdruecken. Gegeben ist das Kommando 1

```
ls -l /z/deck/util/shell/csh.01
```

und gegeben ist die Substitution

```
!1:s/csh.01/csh*/
```

Das folgende Kommando

```
!1:q
```

erzeugt

```
ls -l /z/deck/util/shell/csh*
```

jedoch wird in diesem Fall der Stern buchstabengetreu genommen, er wird nicht ausgedehnt. Beachte den folgenden Austausch (der Ausgabe wurden Leerzeichen hinzugefuegt, um die Lesbarkeit zu erhoehen):

```
% 1 ls -l csh.01

-rw-r--r-- 1 deck system 15459 Oct 13 12:49 csh.01

% 2 !1:s/csh.01/csh.*/

ls -1 csh.*

-rw-r--r-- 1 deck system 15459 Oct 13 12:49 csh.01
-rw-r--r-- 1 deck system 18238 Oct 13 12:50 csh.02
-rw-r--r-- 1 deck system 13347 Oct 13 12:50 csh.03
-rw-r--r-- 1 deck system  3316 Oct 13 12:50 csh.04
-rw-r--r-- 1 deck system 30814 Oct 13 12:51 csh.9A
-rw-r--r-- 1 deck system  2395 Oct 13 12:51 csh.9T
-rw-r--r-- 1 deck system 45153 Nov  9 17:59 csh.ref

% 3 !2:q

ls -l csh.*

csh.* not found
```

:x wie q, doch unterteilt die Argumente bei Leerzeichen Tabs und Newlines.

:& Wiederhole die vorangegangene Substitution.

```
! 1:&
```

Modifizierung vorheriger Kommandowoerter

Syntax	Erlaeuterung	Beispiel
!n:h	take the head of the pathname	!1:h
!n:r	leave the root of the filename	!1:r
!n:t	leave the tail of the pathname	!1:t
!n:p	print but don't execute	!1:p
!n:s/X/Y/	replace X with Y	!1:s/unix/wega/
!n:q	quote substituted arguments	!1:q
!n:x	quote, break substituted arguments	!1:x
!n:&	repeat previous substitution	!1:&

Sofern nicht durch ein ":g" eingeleitet, wird die Modifikation nur am ersten modifizierbaren Argument vorgenommen. Ein backslash-Zeichen "\" muss verwendet werden, um ein "/"-Zeichen zu kennzeichnen, wenn es auf der linken Seite des Substitutionsstring verwendet wird - d.h. wenn es ein Teil des string1 im untenstehenden Beispiel ist:

```
!1:s/string1/string2
```

4.6. Magiczeichen in der History-Funktion

& Das Ampersand-Zeichen "&" auf der rechten Seite einer Substitutionsaussage wird durch den Text auf der linken Seite der Aussage ersetzt. Wenn z.B. das erste Kommando

```
% 1 ls /z/deck/util
```

ist, kann eine das Ampersand benutzende Substitution wie folgt verwendet werden:

```
!1:s/util/&.plus
```

und hat das Kommando

```
ls /z/deck/util.plus
```

zur Folge.

null
Nichts im linken string verwendet das vorangegangene string von einem vorangegangenen Substitutionskommando, d.h.

```
!1:s//Memos/
```

setzt anstelle des vorangegangenen string "util" das string "Memos" ein und erzeugt das Kommando

```
ls /z/deck/Memos
```

Das anfaenglich verwendete string (in diesem Falle "util") muss im Kommandostring vertreten sein, sonst wird folgende Fehlermeldung

```
Modifier failed
```

generiert.

!$ Ein Historyverweis kann ohne die Spezifizierung eines Ereignisses gegeben werden; z.B. verweist "!$" auf das letzte Argument in einem Kommandostring. Gegeben ist das Kommando

```
ls -l /z/deck/util/cshell/csh.01
```

Das Kommando

```
cat !$
```

erzeugt

```
cat /z/deck/util/cshell/csh.01
```

In diesem Fall wird auf das vorangegangene Kommando verwiesen. Folglich gibt

```
!?string?^ !$
```

das erste und letzte Argument des Kommandos, das zu ?string? passt. Einfache Kommandosubstitutionen werden mit der Pfeiltaste vorgenommen. Das falsche eingegebene Kommando

```
cat /usr/lab/news/wega
```

kann mit dem Kommando

```
^lab^lib^
```

korrigiert werden.

{} Eine History-Substitution kann mit einer linken und rechten geschweiften Klammer "{" und "}" umgeben sein, um sie von den folgenden Zeichen zu isolieren. Folglich wird nach

```
ls /z/cheryl
```

durch

```
!{1}/temp

ls /z/cheryl/temp
```

auszufuehren, entgegengesetzt zum Kommando

```
!1/temp
```

das nach einem Kommando

```
1/temp
```

sucht.

% Erzeugt das Argument, das den unmittelbar einleitenden ?string? enthaelt. Das Kommando

```
cd /z/deck/util/shell
```

ist gegeben. Die Eingabe

```
ls !?shell?%
```

wird das Kommando

```
ls /z/deck/util/shell
```

erzeugen.

Die folgende Tabelle stellt eine Zusammenfassung der Metazeichen dar, die bei der History-Substitutionsfunktion verwendet werden.

Metazeichen in History-Substitutionen

Zeichen	Bedeutung
\	Escapes magic qualities
&	The string just substituted
//	(null) the string just searched for
!$	The last element of the last command
?x?	String search for "x"
^x^y^	Substitution routine
{x}	Search insulators
%	Element search

5. C-Shell-Kommandostruktur

Die C-Shell schafft eine hervorragende Arbeitsumgebung, da sie gegenueber vorangegangenen Shells eine Reihe von Verbesserungen bietet. Die C-Shell bietet insbesondere eine eingebaute Kommandosprache, die sich der Struktur der C-Programmiersprache bedient.

Dieser Abschnitt enthaelt allgemeine Kommandos, die entweder in der Kommandozeile oder vom Inneren des Hauptteils eines Shell-Skripts verwendet werden. Der naechste Abschnitt beinhaltet die C-Shell-Programmiersprache.

5.1. Einfuehrung in die C-Shell-Kommandos

Es gibt 38 eingebaute C-Shell-Kommandos. Diese Kommandos lassen sich in 4 Hauptkategorien einteilen.

Die ersten zehn sind allgemeine Kommandos. Die zweiten zehn befassen sich mit der Schaffung und Veraenderung der Arbeitsumgebung. Zusammen bilden diese 20 Kommandos einen Kommandosatz, der sowohl im Dialog als auch im Innern des Hauptteils eines Shellskripts verwendbar ist.

Der naechste Abschnitt stellt die Kommandos dar, die sich speziell mit der Programmierung befassen. Die 15 Kommandos in diesem Satz werden zur Steuerung des Ablaufs von Operationen in einem Shell-Skript verwendet.

Der letzte Satz von 3 Kommandos sind allgemeine Kommandos, die meist ausschliesslich vom Innern des Hauptteils eines Shell-Skripts verwendbar sind.

5.2. Allgemeine dialogorientierte Kommandos

Eingebaute Kommandos sind ein Teil der C-Shell selbst, es sind keine durch die C-Shell ausgefuehrten Einzelprogramme. Die folgenden eingebauten Kommandos sind sowohl im Dialog als auch im Hauptteil eines C-Shell-Skriptes verwendbar.

5.2.1. cd:

Syntax:

```
cd
cd name
```

Ohne ein Argument wandelt cd die gegenwaertige Arbeitsdirectory in die Home-Directory des Nutzers um. cd liest die HOME-Variable um die Home-Directory des Nutzers zu bestimmen.

Mit Pfadnamen als Argument wandelt cd die gegenwaertige Arbeitsdirectory in den angegebenen Directory-Namen um. Fehlernachrichten werden erzeugt, wenn der Zielname kein gueltiger Directory-Name ist, oder wenn dem Nutzer der Zugriff auf jene Directory nicht gestattet ist (siehe chmod(1)).

Die Ausgabe der gegenwaertigen Arbeitsdirectory erfolgt ueber das pwd(1)-Kommando.

Siehe auch: cd(1), pwd(1), chmod(1) und Abschnitt 7.1.5 - home.

5.2.2. echo:

Syntax:

```
echo [-n] string
```

Die Woerter der angegebenen Zeichenkette werden auf dem Terminal ausgegeben. Dies ist fuer Ausgaben (z.B. Hinweise ec.) eines Shellskript verwendbar. Echo kann ebenfalls zur Verifizierung der wahren Merkmale der Shell-Variablen und Metazeichen verwendet werden. Z.B. dehnt das Kommando

```
echo *
```

das Metazeichen Stern ("*") auf alle Dateinamen in der gegenwaertigen Arbeitsdirectory aus (siehe Teil 3 - Dateinamen-Substitution) und schreibt die Liste in folgendem Format:

```
csh.01 csh02 csh03 csh04 csh04 csh05 csh.9A csh.9T
```

Die -n Option unterdrueckt das Newline am Ende der Ausgabe. Vom Innern des Hauptteils eines Shell-Skripts erzeugt das Kommando

```
echo -n Hello
echo \ Roberta
```

die Ausgabe

```
Hello Roberta
```

Beachte, dass das "hard space" (die "\ "-Folge) benoetigt wird, um ein Leerzeichen zwischen beide Woerter zu setzen. Normalerweise ignoriert echo fuehrende und nachgestellte Freiraeume (Leerzeichen oder tab-Zeichen).

Siehe auch: echo(1) und echo2(1) und die echo-Variable in Teil 7.1 - vordefinierte C-Shell-Variablen.

5.2.3. glob:

Syntax:

```
glob string
```

Das glob-Kommando ist aehnlich dem echo-Kommando, mit der Ausnahme, dass Woerter nicht durch Leerzeichen getrennt werden und keine Newlines die angegebene Zeichenkette beenden. Dieses Kommando ist fuer Programme verwendbar, die Shell benutzen, um eine Liste von Woertern auszudehnen. Das Kommando:

```
glob *
```

erzeugt die Ergebnisse

```
csh.01 csh.02 csh.03 csh.04 csh.05 csh.9A csh.9T
```

5.2.4. history

Syntax:

```
history
```

Das History-Kommando zeigt die gespeicherten Kommandos an. Die Laenge der History-Liste wird durch die history-Variable bestimmt (siehe Abschnitt 7.1. vordefinierte C-Shell-Variablen). Die history-Variable kann mit dem Kommando:

```
set history=15
```

z.B. auf den Wert 15 gesetzt werden. Mit der zu 15 gesetzten history-Variablen kann das Kommando

```
history
```

eine History-Liste der letzten 15 Kommandos erzeugen. Die Ergebnisse werden in folgendem Format erzeugt:

```
    1       alias
    2       alias ll 'ls -l'
    3       foreach i (1 2 3 4)
            .
            .
            .
   15       history
```

Wenn Kommando Nummer 16 erreicht ist, wird die Liste mit Kommando Nummer 2 beginnen, etc. (siehe Teil 4 - Die History-Funktion)

5.2.5. nice:

Syntax:

```
nice
nice -number
nice command
nice -number command
```

Der nice-Wert setzt die Prioritaet eines Kommandos im System. Der nice-Wert jedes Prozesses kann mit dem Kommando:

```
ps -l
```

angezeigt werden.

```
... CPU PRI  NI  ADDR  SZ  WCHAN  TTY  TIME  COMD
...   0  30  20    f5  15   dd18  2    0:16  csh
...   0  30  20   5d3   9   debc  2    0:01  sh
...   0  30  20   4b1   9   ded8  2    0:02  csh
...   0  28  20   61a   8   e886  2    0:00  script
...   0  30  20    bb  14   df48  2    l:30  vi
...   9  30  20   77e   9   df64  2    0:02  csh
...   0  26  20   692   8   327a  2    0:00  script
... 101  56  20   706  11         2    0:02  ps
```

Anmerkung: Die ersten 5 Spalten wurden weggelassen, um die Ausgabe dem Platz anzupassen. (Siehe ps(1) fuer Details) Der nice-Wert (im Beispiel in Spalte 3 gezeigt) kann mit dem nice-Kommando erhoeht werden, um die Aufgabe mit einer niedrigeren Systemprioritaet laufen zu lassen. Ohne ein Argument erhoeht nice den nice-Wert fuer die gegenwaertige Shell um 7. Mit einem Argument in Form von

```
nice -N
```

wird der nice-Wert um N erhoeht. (N sollte eine beliebige Zahl bis 20 sein).

Mit einem Kommando als zweites Argument von der Form

```
nice -N command
```

wird der nice-Wert von command um N erhoeht. Es gibt keinen Weg fuer einen normalen Nutzer (ausser dem Superuser) eine negative nice-Zahl zu setzen, d.h. die Prioritaet eines Kommandos zu erhoehen. Der Superuser kann ein solches Kommando in der Form:

```
nice --N
```

eingeben. Das erste nice-Argument muss entweder ein Minuszeichen oder ein Kommando sein.

Siehe auch: nice(1)

5.2.6. rehash:

Syntax:

```
rehash
```

Die C-Shell enthaelt eine geordnete Liste (hash-Tabelle) aller dem Nutzer zur Verfuegung stehenden Kommandos. Diese Liste wird beim login und wenn eine neue C-Shell erzeugt wird gebildet. Die hash-Tabelle wird im Speicher aufbewahrt und enthaelt eine Liste aller Dateinamen (Kommandos) in den Directories des Suchpfads des Nutzers.

Wenn ein neues Kommando gebildet wird (wie im Falle eines neuen Shell-Skripts) erscheint es nicht in der hash-Tabelle, auch wenn es in einer Directory des Suchpfads ist. Das gilt, sofern das neue Kommando (Datei) nicht in der gegenwaertigen Arbeitsdirectory des Nutzers ist.

Ein neues Kommando wird mit dem Kommando rehash in die hash-Tabelle eingefuegt. Das Kommando muss in einer im Suchpfad des Nutzers genannten Directory liegen.

Die Nachricht:

```
command.name: command not found
```

wird erscheinen, wenn das Kommando nicht in der hash-Tabelle untergebracht oder nicht in der gegenwaertigen Arbeitsdirectory ist.

Siehe auch: Abschnitt 7.1.11 - path

5.2.7. repeat:

Syntax:

```
repeat N command
```

Das spezifizierte Kommando wird N-mal wiederholt.

Das Kommando:

```
repeat 5 ls
```

erzeugt Ergebnisse in folgendem Format:

```
csh.01 csh.02 csh.03 csh.04 csh.05 csh.9A csh.9T
csh.01 csh.02 csh.03 csh.04 csh.05 csh.9A csh.9T
csh.01 csh.02 csh.03 csh.o4 csh.05 csh.9A csh.9T
csh.01 csh.02 csh.03 csh.04 csh.05 csh.9A csh.9T
csh.01 csh.02 csh.03 csh.04 csh.05 csh.9A csh.9T
```

Fehlerdiagnostiken werden wiederholt.

5.2.8. time:

Syntax:

```
time
time command
```

Ohne Argument wird die Zeit, die von der gegenwaertigen Shell und deren Child-Prozessen verbraucht wurde ausgegeben. Das Kommando

```
time
```

erzeugt Ergebnisse folgender Art:

```
0.1u 0.3s 0:10 3%
```

Die erste Spalte sagt etwas ueber die Nutzersekunden, die zweite Spalte ueber die Systemsekunden aus, die dritte Spalte ist die tatsaechliche Zeit und die letzte Spalte gibt den Prozentsatz der vom Kommando verwendeten totalen Systemkapazitaet wieder.

Sind Argumente gegeben, wird die Laufzeit des spezifizierten Kommandos gemessen und erzeugt eine Zeituebersicht, wie oben beschrieben.

Das Kommando

```
time ls
```

erzeugt Ergebnisse folgender Art:

```
csh.01  csh.04  csh.07  csh.9B  junk
csh.02  csh.05  csh.08  csh.9C  make.out
csh.03  csh.06  csh.9A  csh.97  temp
0.1u 0.2s 0:01 23%
```

Das time-Kommando erzeugt Ergebnisse, auch wenn das gemessene Kommando einen Fehler erzeugt.

Siehe auch: time(1)

5.2.9. umask:

Syntax:

```
umask
umask N
```

Der umask-Wert bestimmt den impliziten Dateischutzmode fuer neue Dateien (siehe chmod(1)).

Ohne Argument zeigt das umask-Kommando den gegenwaertigen umask-Wert an. Mit einem Zahlenargument wird der umask-Wert auf N gesetzt. Das Kommando:

```
unmask 026
```

hat zur Folge, dass neue Dateien mit dem folgenden protection mode (wie mit dem ls -l Kommando angezeigt) erzeugt weden.

```
-rw-r----- 1 deck system 0 Dec 1 14:15 temp
```

Umask-codes fuer neue Dateien sind:

```
000 =   -rw-rw-rw-
111 =   -rw-rw-rw-
222 =   -r--r--r--
333 =   -r--r--r--
444 =   --w--w--w-
555 =   --w--w--w-
666 =   ----------
777 =   ----------
```

Beachte, dass das Ausfuehrungs-Bit ("x"-Bit) niemals gesetzt ist. Das ist ein Schutz vor versehentlichen Ausfuehrungen von Textdateien, die unbeabsichtigte (und moeglicherweise zerstoerende) Konsequenzen zur Folge haben koennen. Umask-Codes fuer Directories werden das Ausfuehrungs-Bit, falls erwuenscht, setzen. Die spezifischen Codes sind folgende:

```
000 =   drwxrwxrwx
111 =   drw-rw-rw-
222 =   dr-xr-xr-x
333 =   dr--r--r--
444 =   d-wx-wx-wx
555 =   d-w--w--w-
666 =   d--x--x--x
777 =   d---------
```

Wenn umask zurueckgesetzt wird, ist sein impliziter Wert 002.

Siehe auch: chmod(1)

5.2.10. wait:

Syntax:

```
wait
```

Das wait-Kommando veranlasst das Warten auf die Beendigung aller Hintergrund-(Child)-Prozesse.

Durch eine Unterbrechung (DEL-Taste) kann das wait-Kommando abgebrochen werden. Es gibt dann die Prozessidentifikationsnummern und die Kommandonamen aus. Betrachte den folgenden Ablauf:

```
sleep 200 &
27233
wait
     (interrupt)
27233 sleep
wait: Interrupted.
```

Im obigen Beispiel laeuft ein Kommando (sleep 200) im Hintergrund (&). Eine Prozessidentifikationsnummer wird (27233) ausgegeben und dann wird das wait-Kommando eingegeben.

Das wait-Kommando wird unterbrochen (DEL-Taste) die Prozessidentifikationsnummer wird zusammen mit dem Kommandonamen (27233 sleep) sowie der diagnostischen Nachricht wait: Interrupted, angezeigt.

Siehe auch: wait(1)

Die folgende Tabelle zeigt die Kommandos in Gruppe 1:

Zusammenfassung der eingebauten Kommandos

---

cd	Change working directory Aendert die Arbeitsdirectory
echo	Print a string on the terminal Schreibt eine Zeichenkette auf das Terminal
glob	Like echo -- but no spaces separate words Wie echo, doch ohne das Leerzeichen Woerter trennen
history	Print command history list Schreibt die Kommando-History-Liste
nice	Set the running priority of a command Setzt die laufende Prioritaet eines Kommandos
rehash	Re-sort the search path for commands Neu sortieren des Suchpfades fuer Kommandos
repeat	Repead a command Wiederholt ein Kommando
time	Time the execution of a command Misst die Ausfuehrung eines Kommandos
umask	Set the execution bits on new files Setzt die Ausfuehrungsbits neuer Dateien
wait	Wait for background jobs to finish Wartet auf die Beendigung von Hintergrundjobs

---

5.3. Umgebungskommandos

Dieser Kommandosatz wird verwendet, um die Arbeitsumgebung zu beeinflussen. Diese Kommandos werden verwendet, um Kurzschriftkommandos (Aliases) fuer lange Kommandos, C-Shell-Variablen und Kurzschrift-Woerter fuer Datei- und Directory-Namen zu bilden und zu entfernen.

5.3.1. alias/unalias:

Syntax:

```
alias name long_command
alias name
alias
unalias name
```

Das alias-Kommando schafft eine nutzerdefinierte Kurzschrift fuer lange Kommandos. Mit der Syntax alias name long_command wird ein alias-Name fuer das Kommando long_command gebildet. Das Kommando

```
alias h history
```

schafft das alias h, das als History-Kommando verwendet werden kann. Das Kommando:

```
h
```

erzeugt dieselbe Ausgabe wie das Kommando:

```
history
```

(history bleibt ein gueltiges Kommando). In diesem Zusammenhang erzeugt das alias-Kommando keine Ausgabe. Die Pruefung, dass h das alias fuer history ist, kann von den naechsten Kommandos ausgehen.

Mit einem Argument zeigt alias das alias fuer jenes Argument, falls es existiert, an. Mit dem geschaffenen h alias erzeugt das Kommando

```
alias h
```

die Ausgabe

```
history
```

Ohne ein Argument zeigt alias eine Liste der gegenwaertigen aliases an. Mit dem oben geschaffenen alias erzeugt das Kommando

```
alias
```

eine Auflistung in folgendem Format:

```
h history
```

unalias wird zum Entfernen eines alias verwendet (siehe unalias unten).

Eine Rekursion in einem alias, wie im Kommando:

```
alias ls 'pwd; ls'
```

ist nicht erlaubt. Das Kommando kann im alias verwendet werden, wenn es das erste Wort des alias selbst ist, z.B. arbeitet

```
alias ls 'ls; pwd'
```

ohne einen loop-Fehler.

Das Problem der Rekursion wird durch C-Shell durch die Fehlernachricht:

```
Alias loop
```

angezeigt. Das alias muss dann mit dem Kommando

```
unalias alias_name
```

entfernt werden.

Beispiel 1:

Um ein alias zu schaffen, das "ls" fuer das laengere Kommando "ls -l" verwendet, wird das folgende alias-Kommando benutzt:

```
alias ls 'ls -l'
```

Das Kommando

```
ls /z/carol
```

wird eine Ausgabe erzeugen, als waere das geschriebene Kommando

```
ls -l /z/carol
```

eingegeben worden.

Beispiel 2:

Aliases akzeptieren ebenfalls Eingaben. Z.B. kann ein Dateiname

an ein alias-command uebergeben werden.

Der Ausdruck "\!*" bedeutet, wie sein Gegenstueck im History-Mechanismus, "Argumentnummer 1 des letzten Wortes". (Siehe Teil 4 - Die History-Funktion). Dieser Ausdruck substituiert alle Argumente, die auf der Kommandozeile geschrieben sind (ausser Argument Nummer 0 - das Kommando selbst) in das alias-Kommando. Folglich bildet das Kommando

```
alias print 'pr \!* | lpr'
```

ein alias, print genannt, das pr(1) aufruft, ein oder mehrere Argumente (Dateinamen) als Eingabe akzeptiert und die Ausgabe jenes Kommandos an lpr(1) uebergibt. Das alias wird mit der Syntax

```
print file.1
```

verwendet. Die Ergebnisse sind die selben, als waere das folgende Kommando eingegeben worden:

```
pr file.1 | lpr.
```

Der Ausdruck \!* wird durch die Argumente 1 bis zum letzten (Argument Null in diesem Fall ist das Wort print) ersetzt.

5.3.2. exit:

Syntax

```
exit
exit (N)
```

Exit ist mit logout(1) vergleichbar. Es ist ein Mittel zur Beendigung der gegenwaertigen Arbeitsumgebung (der Shell). Exit ist eine permanente Beendigung der gegenwaertigen Arbeitsshell, entgegengesetzt zu einem zeitweiligen Austritt (fork), der den erzeugenden Shellprozess aktiv haelt, bis der Nutzer zu ihm zurueckkehrt.

Ist die gegenwaertige Arbeitsshell die login-shell des Nutzers, fuehrt exit die .logout-Datei (falls sie existiert) des Nutzers aus und entfernt den Nutzer aus dem System. Wenn die ignoreexit-Variable gesetzt ist, wird die Fehlernachricht

```
Can't exit, ignoreexit is set
```

zurueckgegeben. (Siehe Abschnitt 7.1. Vordefinierte C-Shell-Variablen).

Ist ignoreeof gesetzt, gibt die Eingabe von control-D die Fehlernachricht

Use "exit" to logout.

zurueck. In beiden Faellen gibt ein logout(1)-Kommando die Fehlernachricht:

Not login shell.

zurueck, wenn die gegenwaertige Shell nicht die login-shell ist.

Wenn sowohl die ignoreexit- als auch die ignoreeof-Variable gesetzt ist und die gegenwaertige Shell nicht die login-shell ist, so ist der einzige Weg, jene Shell zu verlassen, das Zuruecksetzen einer der Variablen. Das Kommando

unset ignoreeof

gestattet dem control-D (entspricht "end-of-file" eof), die gegenwaertige Shell zu beenden. Das Kommando

unset ignoreexit

gestattet dem exit-Kommando, die gegenwaertige Shell zu beenden.

Durch das exit-Kommando selbst wird die gegenwaertige Shell mit dem Wert der status-Variablen verlassen. Dieser Wert kann mit dem Kommando

echo $status

angezeigt werden.

Dieses Kommando zeigt den status-code der gegenwaertigen Shell an, "0" ist der normale exit status, "1" oder jede andere Zahl, die nicht 0 ist, stellt einen abnormalen exit-status dar, z.B. wenn ein Kommando misslingt.

Mit einem Zahlenargument kann exit die Shell verlassen und setzt die status-Variable auf eine spezifizierte Zahl N. Das ist fuer das Verfolgen von Prozessen in C-Shell-Skripten nuetzlich. Die runden Klammern, die die Zahl umgeben, sind notwendig.

Siehe auch: logout(1)

5.3.3. logout:

Syntax:

logout

Logout beendet eine Login Shell und fuehrt den Inhalt der ~/.logout-Datei (falls sie existiert) aus. Es ist besonders nuetzlich, wenn ignoreeof gesetzt ist, so das ein control-D Kommando

nicht funktioniert.

Logout kann nur von der login-Shell ausgefuehrt werden; ist die gegenwaertige Shell irgendeine andere, wird die Fehlernachricht

```
Not login shell
```

zurueckgegeben.

Siehe auch: Teil 9.2.1 - Die ~/.logout-Datei

5.3.4. set/unset:

Syntax

```
set
set variable=word
set variable=(wordlist)
set variable[index]=word

unset variable
```

Das set-Kommando ohne ein Argument zeigt den Wert aller Shell Variablen an, z.B.:

```
S               /z/deck/Util/Sh.1
U               /z/deck/Util/New.csh
argv            ()
exinit          set number wm=20 | version
history         50
home            /z/deck
ignoreeof
ignoreexit      1
mail            /usr/spool/mail/deck
path            (. /usr/bin /bin /z/deck/bin /etc)
prompt          deck # ! >
shell           /bin/csh
status          0
term            vt100
```

Variablen, die als Wert kein einzelnes Wort besitzen, werden als eine Wortliste, mit runden Klammern umgeben, geschrieben.

C-Shell-Variablen koennen mit dem Kommando

```
set variable=word
```

geschaffen werden, das variable den Wert word gibt. Variable kann eine vorderfinierte C-Shell-Variable (siehe Abschnitt 7.1. Vordefinierte C-Shell-Variablen) oder eine nutzerdefinierte Variable sein.

Variablen koennen als Wert auch eine Liste von Woertern zugewiesen

werden.

```
set variable = (wordlist).
```

Z.B. kann die Variable X zu allen Dateinamen gesetzt werden, die mit den Buchstaben "csh" in der gegenwaertigen Directory beginnen, mit dem Kommando

```
set X=csh*
```

Die Pruefung, dass X die Liste enthaelt, erfolgt durch das Kommando

```
echo $X
```

das die Ausgabe in folgendem Format erzeugt:

```
csh.01 csh.02 csh.03 csh.04 csh.05 csh.06 csh.9B
```

Jeder Komponententeil dieser Liste kann mit einem in eckige Klammern gesetzten subscript adressiert werden, wie beim Kommando

```
echo $x[3]
```

dass das 3. Element der Liste wiedergibt:

```
csh.03
```

Diese einzelnen Elemente einer Liste koennen in derselben Weise veraendert werden, in der sie adressiert werden - mit einem subscript-Wert. Die Kommandosyntax ist:

```
set variable[N]=word
```

das die N-te Komponente von variable zu word setzt; z.B. setzt das folgende Kommando den Wert der 3. Komponente der X-Variable zum Wort test

```
set X[3]=test.
```

Diese Komponente muss bereits existieren. Die Pruefung kann durch die folgenden Kommandos erfolgen:

```
echo $X
```

erstattet Bericht ueber alle Elemente in der Variablen X (die eine Aufstellung von Woertern darstellen):

```
csh.01 csh.02 csh.03 csh.04 csh.05 csh.06 csh.9B
```

Das Kommando

```
echo $X[3]
```

erstattet Bericht ueber den Wert des 3. Elements des Feldes test.

Variablen werden mit dem unset-Kommando aus der Variablenliste entfernt:

```
unset variable
```

Diese gesetzten Argumente koennen wiederholt werden, um mehrere Werte in ein einfaches set-Kommando zu sehen. Die Variablenexpansion geht bei allen Argumenten vor sich, bevor jegliches Setzen vonstatten geht.

Siehe auch Teil 7: Shell-Variablen und Abschnitt 3.2. Metazeichen ([, und])

5.3.5. sentenv/env:

Syntax:

```
sentenv NAME = value
env
```

Das sentenv-Kommando funktioniert wie das set-Kommando, es setzt environment variables, waehrend set shell variables setzt. Siehe Teil 10 zur Eroerterung der Umgebung und ihrer Variablen.

Das Kommando

```
sentenv NAME=value
```

setzt den Wert der Umgebungsvariablen NAME zu value. Vordefinierte Umgebungsvariablen sind:

```
LOGNAME     Login name
EXINIT      Ex editor initialization variables
HOME        Home directory
PATH        Search path for commands
SHELL       Shell being used
TERM        Type of terminal
TZ          Timezone
```

Env schreibt die Werte der gegenwaertig gesetzten Umgebungsvariablen. Siehe Teil 10 fuer eine Beschreibung der Umgebung und ihrer Umgebungsvariablen.

Siehe auch: env(1) Teil 10 - Die Umgebung

5.3.6. source:

Syntax:

```
source file.name
```

Die Shell liest Kommandos von file.name und fuegt sie in die

gegenwaertige Shell ein (entgegengesetzt zum Abspalten einer neuen Shell).

Source-Ausgaben koennen nicht umgelenkt werden.

Siehe auch: Teil 10 zur Eroerterung dessen, wie die C-Shell Kommandos ausfuehrt.

5.3.7. unalias/alias:

Syntax:

```
unalias pattern
```

Alle aliases, deren Namen zum spezifizierten Muster passen, werden gestrichen. Folglich werden alle aliases mit dem Kommando

```
unalias *
```

entfernt.

Siehe auch: Abschnitt 5.3.1. alias/unalias

5.3.8. unset/set:

Syntax:

```
unset pattern
```

Alle Variablen, deren Namen zum spezifizierten Muster passen, werden entfernt. Alle Variablen werden mit dem Kommando

```
unset *
```

entfernt; das kann aber unerwuenschte Nebenwirkungen hervorrufen.

Siehe auch: Abschnitt 5.3.4. - set/unset

5.3.9. Das At-Zeichen:

Syntax:

```
@ variable = (number operator number)
```

Das At-Zeichen setzt Variablen auf einen numerischen Wert. Die Variable erhaelt das Ergebnis der mathematischen Funktion, und nicht die Zeichenkette, als Wert.

Z.B. ergibt das Kommando:

```
        set x=( 6 + 6 )
```

das Ergebnis (ueber echo $x ausgebbar)

```
        6 + 6
```

und

```
        @ x=( 6 + 6 )
```

erzeugt das Ergebnis:

```
        12
```

Beachte, dass das At-Zeichen gegenueber der Syntax empfindlich ist. Die Leerzeichen, die die Zahlen voneinander trennen, sind wesentlich.

Zusammenfassung der Umgebungskommandos

---

alias	Substitute word for long command
env	Print the current environment variables
printenv	Print the current environment variables
exit	Terminate a shell (logout)
logout	Exit from the login shell
set	Establish a shell variable
setenv	Establish an environment variable
source	Execute a script in the current shell
unalias	Remove an alias
unset	Unset a shell variable
@	Like "set" but uses math functions

---

6. C-Shell-Programmiersprache

Die C-Shell kann wegen der umfangreichen Unterstuetzung von Ablaeufen als eine Programmiersprache angesehen werden. Die Syntax der C-Shell-Sprache ist aehnlich der C-Programmiersprache.

6.1. foreach/end-Gruppe

Syntax:

```
foreach name (list)
   command
end
```

Wenn das foreach-Kommando im Dialog eingegeben wird und name und list werden in der richtigen Syntax geschrieben, erscheint ein neues Prompt (ein Fragezeichen), um anzuzeigen, dass ein C-Shell-Schleife im Gange ist.

Auf Prompt (?) koennen eine oder mehrere Kommandoaussagen eingegeben werden. Die Schleife wird initiiert, wenn das Wort end alleine auf eine Zeile nach Prompt "?" geschrieben wird. An jener Stelle wird das Wort list (falls es Magiczeichen enthaelt) ausgedehnt und das Kommando ein Mal fuer jedes Element in der list ausgefuehrt. Die Ausfuehrung wird fortgesetzt, bis list erschoepft ist.

Die Programmierstruktur der foreach-Schleife (und aller Steuerungsstrukturen der C-Shell) verwendet die konventionelle C-Programmier-Syntax. Das Beispiel unten veranschaulicht eine einfache foreach-Schleife und die Daten, die sie erzeugt.

```
% foreach i ( 1 2 3 4 )
? echo $i
? end
1
2
3
4
```

In diesem Beispiel wird die Variable "i" erst zum Zeichen "1" gesetzt, dann wird das Kommando echo $i ausgefuehrt. Durch die end-Anweisung wird die Steuerung zurueck zur foreach-Anweisung gegeben, die ueberprueft, ob es noch mehr Punkte in der Liste gibt. Gibt es noch einen anderen Punkt, wird "i" zu diesem Punkt gesetzt und die Schleife wird wiederholt, bis es keine Punkte in der Liste zwischen den runden Klammern mehr gibt. Bei jedem nachfolgenden Durchlauf der Schleife, verweist "$i" auf den gegenwaertigen Wert der Variablen "i", wie er in der foreach-Aussage festgelegt ist. Bei der ersten Wiederholung der Schleife, wird i zum Zeichen "1" gesetzt. Bei der zweiten Wiederholung wird i zum Zeichen "2" gesetzt,

usw.

Jedes Zeichen oder String kann fuer name verwendet werden und jedes Zeichen oder String kann fuer list verwendet werden.

Magiczeichen werden ausgedehnt, es sei denn sie sind in Anfuehrungszeichen gesetzt. Z.B. wird das Kommando

```
foreach i (*)
```

"$i" auf jeden Dateinamen in der gegenwaertigen Arbeitsdirectory ausdehnen. Ein Beispiel fuer diesen Prozess koennte

```
% foreach i (csh.??)
? echo $i
? ls -l $i
? end
```

sein.

Dieses Kommando wird den Namen der Dateien wiedergeben, die mit "csh." beginnen und von zwei Einzelzeichen gefolgt werden und eine lange Auflistung jener Dateien der gegenwaertigen Directory ausgeben und entsprechende Ergebnisse, wie die folgenden, erzeugen:

```
csh.01
-rw-r--r--1 deck system 13620 Nov 4 15:05 csh.01
csh.02
-rw-r--r--1 deck system 14537 Nov 4 13:24 csh.02
csh.03
-rw-rw-r--1 deck system 24776 Nov 4 15:11 csh.03
csh.04
-rw-rw-r--1 deck system  8996 Nov 4 11:06 csh.04
csh.05
-rw-r--r--1 deck system  6631 Nov 3 17:04 csh.05
csh.9A
-rw-rw-r--1 deck system  3481 Nov 4 14:42 csh.9A
csh.9T
-rw-r--r--1 deck system  2995 Nov 3 17:14 csh.9T
```

Sowohl foreach als auch end muessen allein auf getrennten Zeilen erscheinen.

Das eingebaute Kommando continue kann verwendet werden, um die Schleife vorzeitig wieder zu beginnen und das eingebaute Kommando break, um sie vorzeitig zu beenden.

Siehe Beispiel 1 im Abschnitt 6.7.

6.2. while/end-Gruppe

Syntax:

```
while (expression)
  command
end
```

Solange der spezifizierte Ausdruck einen Wert ungleich Null ergibt, werden die Kommandos zwischen while und dem entsprechenden end ausgefuehrt. Der Abbruch (break) und die Fortsetzung (continue) koennen verwendet werden, um die Schleife vorzeitig zu beenden oder fortzusetzen.

Bei Dialogarbeit erfolgt eine Eingabeanforderung fuer die Schleifenkommandos analog zur foreach-Anweisung.

Siehe Beispiel 2 in Abschnitt 6.7.

6.3. if/else/endif-Gruppe

Syntax:

```
if (expression.1) then
        command.1
else if (expression.2) then
        command.2
else
        command.3
endif
```

If ist eine Steueranweisung, die im allgemeinen fuer das Treffen von Entscheidungen innerhalb der while und foreach loop-Strukturen verwendet wird.

Wenn der Ausdruck.1 wahr ist, wird das Kommando.1 ausgefuehrt. Kommando.1 muss ein einfaches Kommando, darf keine Pipeline, keine Kommandoliste oder keine in runde Klammern gesetzte Kommandoliste sein.

Wenn Ausdruck.1 falsch ist, wird die else if Bedingung getestet; wenn Ausdruck.2 wahr ist, werden die Kommandos in Kommando.2 ausgefuehrt.

Eine beliebige Anzahl von else-if-Paaren ist moeglich; es wird nur ein endif benoetigt. Der else-Teil ist gleichfalls optional. Die Woerter else und endif muessen am Anfang der Zeilen erscheinen; das if muss am Anfang einer Zeile oder nach einem else erscheinen. Siehe Beispiel 3 und 4 im Abschnitt 6.7.

6.4. Switch-Gruppe

Syntax:

```
switch (string)
        case label1:
                command
        breaksw
        case label2:
```

```
                        command
                breaksw
                default
                        command
        endsw
```

Switch ist im allgemeinen im Zusammenhang mit einer foreach oder while-Aussage gebraeuchlich. Jedes case label wird nacheinander mit dem spezifizierten string verglichen. Wenn das case label mit string uebereinstimmt, wird das zugeordnete Kommando ausgefuehrt.

Die Dateimetazeichen "*, ?, [und]" koennen in den case labels verwendet werden. Wenn keines der labels passt, bevor ein default label gefunden wird, beginnt die Ausfuehrung nach dem default label.

Jedes case label und das default label muessen am Anfang einer Zeile erscheinen. Das Kommando breaksw verursacht einen Sprung nach endsw. Andernfalls werden alle anderen nachfolgenden Kommandos abgearbeitet, wie in C. Wenn kein label passt und es kein default gibt, wird die Ausfuehrung nach endsw fortgesetzt.

Siehe Beispiel 5 im Abschnitt 6.7.

Kommandos zur Schleifensteuerung in einem Skript:

```
foreach Initiate a foreach loop
end     End of a foreach or while loop

while   Initiate a while loop
end     End of a foreach or while loop

The if group:

if      Initiate an if loop
else    Alternative decision in an if statement
endif   End of an if loop

The Switch group:

switch  Switch to the next interation of the variable
case    Label in a switch statement
breaksw Causes a break from a switch
default Default case in a switch statement
endsw   End of an switch loop

Independent loop control commands:

break   Drops out of the nearest loop
continue Continue exexution of nearest loop
goto    Jump to a new location
```

shift Go to the next argument in the argument variable

6.5. Unabhaengige Steuerkommandos

6.5.1. break

Syntax:

```
break
```

Veranlasst die Fortsetzung nach dem end der nahesten foreach- oder while-Anweisung. Die auf der gegenwaertigen Zeile verbleibenden Kommandos werden ausgefuehrt. Multi-level breaks sind folglich moeglich, wenn man sie alle auf eine Zeile schreibt.

Siehe Beispiel 6 im Abschnitt 6.7.

6.5.2. continue

Syntax:

```
continue
```

Veranlasst die Fortsetzung beim nahesten while oder foreach. Die auf der gegenwaertigen Zeile verbleibenden Kommandos werden ausgefuehrt.

6.5.3. goto

Syntax:

```
goto word
```

Fuer das angegebene word wird eine Dateinamen- und Kommandoexpansion durchgefuehrt, um eine Zeichenkette zu erhalten, die als Marke interpretiert wird. Shell sucht nun nach einer Zeile, die diese Marke plus einem Doppelpunkt am Anfang (moeglicherweise durch Leerzeichen oder Tabs eingeleitet) besitzt. Die Ausfuehrung wird nach der spezifizierten Zeile (mit der Marke) fortgesetzt.

Siehe Beispiel 8 in Abschnitt 6.7.

6.5.4. shift

Syntax:

```
shift
shift variable
```

Die Elemente von argv werden nach links verschoben und argv[1] entfaellt dabei. Es ist ein Fehler, wenn argv nicht gesetzt ist oder weniger als ein Wort als Wert besitzt. Die 2. Form fuehrt dieselbe Funktion fuer die spezifizierte Variable aus.

Siehe Beispiel 9 im Abschnitt 6.7.

6.6. Unabhaengige Shell-Skript-Kommandos

6.6.1. exec

Syntax:

```
exec command
```

Das spezifizierte Kommando wird anstelle der gegenwaertigen Shell ausgefuehrt; Z.B. fuehrt das Kommando

```
exec date
```

das date-Kommando aus und beendet dann die Shell. Wenn das Kommando von der login-shell ausgefuehrt wird, wird das Kommando den Nutzer aus dem System entfernen. Die '.logout'-Datei (falls sie existiert) wird nicht ausgefuehrt.

6.6.2. nohup

Syntax:

```
nohup
nohup command
```

Wenn ein Terminal mit dem System ueber ein modem durch Telefonleitungen verbunden ist, hat das Einhaengen des Telefonhoerers am Ende des Terminals ein logout zur Folge.

Nohup (no hang up) bewirkt, dass das Einhaengen des Hoerers fuer den Rest des Shell-Skripts ignoriert wird und der Skript fortgesetzt werden kann. Das nohup-Kommando mit einem Kommandoargument bewirkt, dass das spezifizierte Kommando laeuft und das Einhaengen ignoriert wird.

6.6.3. onintr

Syntax:

```
onintr -
onintr label
```

Das onintr (on interrupt)-Kommando steuert die Handlung des Shell-Skripts, wenn ein Unterbrechungssignal (normalerweise die

DELETE-Taste) von der Tastatur eingegeben wird.

Mit einem Minuszeichen-Argument werden alle Unterbrechungen ignoriert. Mit einem label-Argument bewirkt onintr, dass die Shell ein goto label ausfuehrt, wenn eine Unterbrechung auftritt.

Siehe Beispiel 4 im Abschnitt 6.7.

Die folgende Uebersicht fasst die 4. Kommandogruppe zusammen:

allgemeine Kommandos:

exec	Causes execution of a command with no return
nohup	No Hangup in a dial-up phone situation
onintr	Goto a new label on receiving an interrupt signal

6.7. Beispiele von Shell-Skripten

Anmerkung:

In diesen Shell-Skript-Beispielen sind die mit einem Doppelkreuz beginnenden Texte nur Kommentare, die auch weggelassen werden koennen.

```
# EXAMPLE 1 -- Foreach
#
foreach i (*)   # The variable "i" is set to all
                # the filenames in the current
                # working directory

    echo $i     # within the loop, each iteration
                # of the variable "i" is printed
                # on the screen

end             # end of the loop
```

Dieser C-Shell-Skript erzeugt eine Ausgabe in folgendem Format:

```
csh.01
csh.02
csh.03
csh.04
csh.05
csh.06
csh.07
csh.08
csh.09
temp
```

```
        # EXAMPLE 2 -- While
        #
        while (1)       # "1" is always true, therefore
                        # this is an endless loop

            echo "This is an endless loop"

                        # The string is printed forever
                        # or until it is interrupted

        end             # end of the while loop
```

Dieses C-Shell-Skript erzeugt die folgende Ausgabe:

```
        This is an endless loop
        This is an endless loop
        This is an endless loop
        This is an endless loop
        This is an endless loop
        This is an endless loop
        This is an endless loop
        This is an endless loop
        This is an endless loop
        .
        .
        .
```

bis eine Unterbrechung (die DELETE-Taste) eingegeben wird.

```
# EXAMPLE 3 -- If
#
foreach i (*) # The variable "i" is
              # set to all filenames
              # in the current working directory

    if ($i == temp) then

              # If this iteration of "$i" is
              # a file named "temp", then do
              # the following:

        echo "Here is the temp file"

              # Print the string

    else
              # If "$i" is not temp, then do
              # the following:

        echo $i

              # Print the filename

    endif     # Necessary end of the conditional
```

```
              # "if" statement
end           # End of the "foreach" loop
```

Der obige C-Shell-Skript erzeugt eine Ausgabe in folgendem Format:

```
        csh.01
        csh.02
        csh.03
        csh.04
        csh.05
        csh.06
        csh.07
        csh.08
        csh,09
        Here is the temp file
```

```
# EXAMPLE 4, An "if" conditional statement within a
# "while"-loop, -- "onintr" and "set" using the math
# statement "@" are also demonstrated
#
onintr hook     # Establishes #hook" as the  label
                # to "goto" on interrupt

set a=0         # Initializes the variable "a"
                # to "0" at the beginning of
                # the "while" loop

while (1)       # "1" is always true, this is
                # an endless loop

    if ($a < 5) then

                # If the variable "a" is less
                # than 5, then perform the next step

        echo "The number is less than 5"

                # Print the string
    else
                # If the number is not less than 5
                # then perform the next step:

        echo "The number is 5 or greater"

                # Print the string
    endif       # End of the "if" conditional statement

@ a++           # Set "a" to "a+1" (increment "a")

end             # End of the "while" loop

hook:           # the label identified in the
```

```
                # "onintr" statement. If an interrupt
                # is encountered, perform the following:

        echo "Interrupt encounter -- Good bye"

                # Print the string
```

Das obige C-Shell-Skript erzeugt eine Ausgabe in folgendem Format:

```
        The number is less than 5
        The number is less than 5
        The number is less than 5
        The number is less than 5
        The number is less than 5
        The number is 5 or greater
        The number is 5 or greater
        The number is 5 or greater
        The number is 5 or greater
        The number is 5 or greater
        The number is 5 or greater
        The number is 5 or greater
        The number is 5 or greater
        The number is 5 or greater
        .
        .
        .
        Interrupt encounter -- Good bye
```

```
# EXAMPLE 5, A "switch" statement nested in a "foreach"
# loop -- use of the metacharacter "?" is also demonstrated

foreach i (*)   # Sets the variable "i" to all
                # the filenames in the current
                # directory.

    switch ($i) # Check this iteration of "$i"
                # to see if it meets the following
                # conditions.

    case ????:  # If the filename in "$i" has four
                # characters, perform the following:

        echo " $i is a four character name"

                # Print the string

        breaksw # And exit out of this case test.

    case ?????: # If the filename in "$i" has five
                # characters, perform the following:

        echo " $i is a five character name"
                # Print the string

        breaksw # And exit out of this case test.

    case ??????: # If the filename in "$i" has six
                # characters, perform the following:

        echo "$i is a six character name"
                # Print the string

        breaksw # And exit out of this case test.

    default     # if the filename  in "$i" does not match
                # any of the above criteria, perform the
                # following:

        echo " $i is not a four, five or six character name "

                # Print the string

    endsw       # And exit out of the whole switch loop

end             # End of the "foreach" loop.
```

Der obige C-Shell-Skript erzeugt z.B. eine Ausgabe in folgendem Format: (Anmerkung: Die Ausgabe ist abhaengig von den Dateinamen in der gegenwaertigen Arbeitsdirectory)

```
    OUT01 is a five character name
    OUT02 is a five character name
    OUT03 is a five character name
```

```
a.out is a five character name
all.tables is not a four, five or six character name
csh.01 is a six character name
csh.02 is a six character name
csh.03 is a six character name
echo.test is not a four, five or six character name
tax is not a four, five or six character name
temp is a four character name
```

```
# EXAMPLE 6: A Break statement in a While loop
#
while (1)
                # This sets up an endless loop
    echo -n " enter x: "
                # The echo statement prompts
                # for input from the the terminal

    set x = `gets`
                # the "set" expression sets the
                # variable "x" to whatever is
                # entered at the terminal (the
                # input is captured with the
                # 'gets' expression)

    if ($x == 'a') then
                # If the input is the letter
                # "a" then proced to the break
                # statement.

    break       # goes the "end" statement
                # echos the last statement "it
                # broke" and drops out of the lop

    else        # If the input is not the letter
                # "a" then proceed back to the
                # nearest loop statement (while).

        echo "it did not break"
                # returns to the
                # beginning of the while loop

    endif       # ends the "if" branch

end
                # ends the "while" loop

    echo "it broke"
                # demonstrates the break
```

Der obige C-Shell-Skript erzeugt einen Austausch in folgendem Format:

```
 enter x: b
in did not break
 enter x: c
it did not break
 enter x: a
it broke
```

Anmerkung: Die "b", "c" und "a"-Zeichen sind Eingaben von der Tastatur als Antwort auf das "enter x:" prompt.

```
# EXAMPLE 7 -- Prompting for input,
# getting input, evaluating
# the input with an if statement, and demonstrating
# the continue statement. Onintr is also demonstrated.
#
while (1)
                # sets up an endless loop

    echo -n "enter x:"
                # as in the example above, this
                # prompts for input from the
                # terminal

    set x = `gets`
                # this set the variable "x" to
                # whatever is entered with the
                # 'gets' command

    if ($x == 'a') then
                # If the input is "a" continue

        echo "it continued"
                # This demonstrates the continuation
                # if the input is "a"

        continue
                # goes to beginning of the enclosing
                # while loop and starts over

    endif
                # ends the "if" statement if
                # the input is not "a"

        echo "it did not continue"
                # the if statement does not continue

        exit
                # if the input is not "a" the
                # "exit" command terminates the
                # "while" loop

end
                # end the "while" loop
```

Das obige C-Shell-Skript erzeugt einen Austausch in folgendem Format:

```
        enter x:a
        it continued
        enter x:a
        it continued
        enter x:a
        it continued
        enter x:b
        it did not continue
```

Anmerkung: Wie im vorangegangenen Beispiel sind die "a" und "b"- Zeichen Eingaben von der Tastatur.

```
# EXAMPLE 8 -- The Goto statement within a foreach
# loop. This shell script tests each filename for the
# name "temp". When the "temp" file is encountered, control
# goes to the label "branch".

foreach i ( * )
        # sets a "foreach" loop with the controlling
        # list variable "i" which is set to all the
        # files in the current direx
        # files in the current directory.

   if ( $i == temp ) then
        # tests each filename to see if it matches
        # the word "temp"

        goto hook
              # if it matches, control jumps
              # to the label "hook"

   else if ( $i != temp)
        # if the filename does not match the
        # word "temp" control drop to the next
        # statement

        echo "$i is not the temp file "
              # echos that the filename is
              # not "temp"
   endif
        # ends the "if" branch statement

end
        # ends the "foreach" loop

hook:
        # the destination of the "goto" label

   echo "Here is temp file -- end loop"
        # reports finding the "temp" file and
        # drops out of the loop
```

Figure 6-8 An Example of the Goto Statement

Das obige C-ShellSkript) erzeugt eine Ausgabe in folgendem Format. Beachte, dass die Ausgabe von den Dateinamen in der gegenwaertigen Arbeitsdirectory abhaengig ist.

```
    csh. 01 is not temp file
    csh. 02 is not temp file
    csh. 03 is not temp file
    csh. 04 is not temp file
    csh. 05 is not temp file
    csh. 06 is not temp file
    csh. 07 is not temp file
    csh. 08 is not temp file
    Here is the temp file -- end loop
```

```
# EXAMPLE 9 -- The Shift statement
#
set a = (*)
        # sets the variable "a" the list of all filenames
        # in the current working directory. If the files
        # are "csh.01 csh.02 csh.03 and csh.04" then
echo $a
        # the list will echo:
        # csh.01 csh.02 csh.03 csh.04
shift a
        # shifting drops the leftmost element to produce:
echo $a
        # csh.02 csh.03 csh.04
shift a
        # shifting drops the leftmost element to produce:
echo $a
        # csh.03 csh.04
```

Die Ausgabe wird ungefaehr so aussehen:

```
    csh.01 csh.02 csh.03 csh.04
    csh.02 csh.03 csh.04
    csh.03 csh.04
```

7. Shell-Variablen

7.1. Vordefinierte C-Shell-Variablen

Die Arbeitsumgebung kann auf unterschiedlichen Wegen beeinflusst werden; dieser Abschnitt erlaeutert die Verwendung der C-Shell-Variablen.

17 Variablen-Namen sind durch die C-Shell vordefiniert. Diese Variablen steuern viele der eingebauten C-Shell-Funktionen.

Abschnitt 7.3 stellt eine Eroerterung der Nutzer-definierten Variablen dar.

7.1.1. argv:

Syntax:

(Wird nicht vom Terminal aus gesetzt)

Argv ist die Kurzform fuer "argument variable". Jedes Kommando, das eingegeben wird, wird in Argumente aufgespalten (zergliedert) und jedes Argument im Kommando wird von 0 an numeriert und in die Argumentvariable fuer die Ausfuehrung plaziert. Im Kommando:

```
ls -l file.01
```

gibt es 3 Argumente:

```
Argument 0 ist das Kommando selbst; ls
Argument 1 ist -l
Argument 2 ist der Dateiname file.01
```

Beim login wird argv durch die C-Shell zu 0 gesetzt. Dieser Wert wird fuer jedes Kommando zu den Namen der Argumente, die fuer das Kommando gegeben werden, neu gesetzt.

Um die argv-Variable zu veranschaulichen, ist die folgende test genannte Datei ein C-Shell-Skript und enthaelt 6 Zeilen:

```
# test
echo $argv
echo $argv[1]
echo $argv[2]
echo $argv[3]
echo $argv[4]
```

Die erste Zeile der Datei enthaelt ein Doppelkreuz (um anzuzeigen, dass es ein C-Shell-Skript ist) und den Dateinamen test.

Der Rest der Datei enthaelt Kommandozeilen, die dafuer bestimmt

sind, die Merkmale der argv-Variable zu demonstrieren. Die zweite Zeile ist ein Kommando, um den gesamten Inhalt der argv-Variable wiederzugeben, die zweite Zeile ist ein Kommando, um Argument 1 der argv-Variable wiederzugeben, die dritte Zeile ist ein Kommando, um Argument 2 wiederzugeben, usw.

Wurde die Datei einmal geschaffen, so muss sie mit dem Kommando

```
chmod 777 test
```

ausfuehrbar gemacht werden. (Siehe chmod(1) im WEGA-Programmierhandbuch) und dann mit drei Argumenten, z.B. wie folgt, ausgefuehrt werden.

```
test a b c
```

erzeugt die folgenden Ergebnisse:

```
a b c
a
b
c
Subscript out of range.
```

Das Skriptkommando echo $argv druckt den gesamten Inhalt der argv-Variable aus - vom ersten Argument zum letzten Argument in argv (alle Argumente, ausser das nullte Argument). Der Ausdruck argv[*] kann ebenfalls verwendet werden. Die folgenden echo-Kommandos drucken die spezifischen Komponenten der Kommandoargumente ab - des 1., 2. und 3. Arguments. Beachte, dass der Aufruf des 4. Arguments $argv[4] den Fehler

```
Subscript out of range
```

erzeugt.

Mit der Syntax

```
$N
```

kann man den Zugriff auf jede Komponente von argv erhalten, wo N eine Zahl ist, die mit der Position des Arguments in der Argumentenliste korrespondiert. Folglich haette der Script auch wie folgt geschrieben werden koennen:

```
# test
echo $*
echo $0
echo $1
echo $2
echo $3
echo $4
```

Die nochmalige Ausfuehrung mit den 3 Argumenten

```
test a b c
```

erzeugt die folgenden Ergebnisse:

```
a b c
test
a
b
c
```

Es gibt einen zweifachen Unterschied. Erstens, ist ein Aufruf des Kommandos selbst (Argument Nummer 0) $0 moeglich und zweitens erzeugt ein Aufruf des Subscript-Wertes, der ausserhalb des Bereichs der Anzahl der Argumente liegt, keinen Fehler, sondern nur eine leere Zeile.

DEFAULT:

```
argv=()
```

Siehe auch: Abschnitt 5.2.2. - echo und 5.2.9. - unmask

7.1.2. child

Syntax:

```
(Nicht vom Terminal aus gesetzt)
```

Die child-Variable traegt die Zahl des letzten Hintergrundprozesses. Das ist nuetzlich fuer das Anhalten eines im Hintergrund laufenden Jobs. Das Kommando

```
kill -9 $child
```

wird den letzten Hintergrund-Job beenden.

DEFAULT:

```
unset by default
```

Siehe auch: Abschnitt 2.5. Ein im Hintergrund laufendes Kommando

7.1.3. echo

Syntax:

```
set echo
```

Die echo-Variable steuert, ob Kommandos, unmittelbar, nachdem sie eingegeben werden, wiedergegeben (geschrieben) werden oder nicht. Wenn sie gesetzt ist, erzeugt die echo-Variable

Ergebnisse in folgendem Format:

```
% ls
ls
csh.01 csh.02 csh.03 csh.04 csh.05
```

Die echo-Variable wird ebenfalls gesetzt, wenn die "-x" Kommandozeilenoption im csh(1)-Kommando angegeben wird. Wie im Kommando

```
csh -x test
```

Bei nicht eingebauten Kommandos geschehen alle Ausdehnungen, bevor sie wiedergegeben werden. Eingebaute Kommandos werden vor der Kommando- und Datennamensubstitution wiedergegeben.

DEFAULT:

```
unset by default
```

Siehe auch: echo(1), echo2(1) Abschnitt 5.2.2. - echo

7.1.4. history

Syntax:

```
set history = N
```

Die history-Variable steuert die Anzahl der Kommandos, die im Speicher in der history-list aufbewahrt werden. Eine zu grosse Zahl kann fuer C-Shell zu einem Speicherfehler fuehren. Das zuletzt ausgefuehrte Kommando wird immer in der History-Liste aufbewahrt.

DEFAULT:

```
unset by default
```

Siehe auch: Teil 4 - Die History-Funktion

7.1.5. home

Syntax:

```
set home=/path/home.directory
```

Die home-Variable verweist auf die home-Directory. Sie wird (anfaenglich) durch einen Eintrag in der /etc/passwd-Datei gesetzt. Sie kann auch im Dialog oder in einem Skript neu gesetzt werden.

Die home-Variable wird verwendet, um das Ziel des cd-Kommandos (wenn es ohne Argumente verwendet wird) festzulegen.

Sie wird ebenfalls verwendet, um den Wert des Metazeichens Tilde "~" festzusetzen.

Home kann zu jeder Directory gesetzt werden, es ist jedoch am gebraeuchlichsten, wenn es auf die Home-Directory verweist.

DEFAULT:

```
home = /path/home.directory
```

Siehe auch: Teil 10 - Die Umgebungsvariablen (HOME) Teil 3 - Die Metazeichen (~) und (/)

7.1.6. ignoreeof

Syntax:

```
set ignoreeof
```

Die ignoreeof-Variable bestimmt, wie die C-Shell die end-of-files-Signale (control-D) vom Terminal handhabt.

Wenn die ignoreeof-Variable gesetzt ist, verhindert sie, dass die Stamm-(login) Shell durch ein versehentliches Control-D beendet wird. Wenn ignoreof gesetzt ist und ein control-D eingegeben wird, wird eine Fehlernachricht

```
Use "logout" to lougot.
```

zurueckgegeben. Die Variable ist in Programmen nuetzlich, wo control-D's eingegeben werden muessen.

DEFAULT:

```
unset by default
```

Siehe auch: Teil 5 - Eingebaute Kommandos (logout, exit)

7.1.7. mail:

Syntax:

```
set mail=/path/directory
set mail=(N /path/directory)
```

Beachte, dass die Klammern eine Wortliste mit eingeschlossenen Leerzeichen umgeben muessen.

Die mail-Variable initiiert eine Prozedur, die die Directory /path/directory alle N-Sekunden nach neuer Post durchsucht.

Wird N weggelassen, ueberprueft Shell die Datei alle 5 Minuten.

Die Ueberpruefung erfolgt nach jedem Kommando, das N-Sekunden zurueck liegt (folglich erfolgt keine Ueberpruefung in einem langen Programm, wie vi(1), dafuer wird die Ueberpruefung nach Verlassen des vi ausgefuehrt).

Beim Auffinden einer neuen Post gibt C-Shell die Nachricht

```
you have new mail.
```

aus. Verschiedene Dateien koennen spezifiziert sein, und wenn es mehrere mail-Dateien gibt, so spezifiziert C-Shell den mail-Dateinamen mit

```
new mail in name.
```

DEFAULT:

```
mail = /usr/spool/mail/name
```

Siehe auch: mail(1)

7.1.8. noclobber

Syntax:

```
set noclobber
```

Wenn die noclobber-Variable gesetzt ist, werden ueber die Ausgabenumlenkung Beschraenkungen verhaengt, um zu sichern, dass Dateien nicht versehentlich zerstoert werden. Ein Versuch, die Ausgabe zu einer existierenden Datei (z.B. test) wie mit dem Kommando

```
who > test
```

umzulenken, hat die Fehlernachricht

```
test: File exists
```

zur Folge. Ausserdem werden Beschraenkungen ueber angehaengte (">>") Umlenkungen verhaengt, um zu sichern, dass die genannten Ausgabe-Dateien auf existierende Dateien verweisen. Ein Versuch, Informationen an eine nicht existierende Datei (z.B. new.file) anzuhaengen, wie mit dem Kommando

```
ls >> new.file
```

hat die Fehlernachricht

```
new.file: No such file or directory
```

zur Folge.

DEFAULT:

unset by default

Siehe auch: Teil 2 - Eingabe- und Ausgabeumlenkung.

7.1.9. noglob

Syntax:

```
set noglob
```

Ist die noglob-Variable gesetzt, verhindert sie die Dateinamenexpansion - die im Abschnitt 3.1. beschriebenen Metazeichen werden sich nicht auf passende Dateinamen ausdehnen. Das Kommando:

```
echo *
```

wird zurueckgegeben:

```
*
```

Das ist in C-Shell-Skripten nuetzlich, die sich nicht mit Dateinamen befassen, oder in Situationen, wo Metazeichen unausgedehnt passieren muessen.

DEFAULT:

unset by default

Siehe auch: Teil 3 - Dateinamenexpansion

7.1.10. nonomatch

Syntax:

```
set nonomatch
```

Wenn die Variable nonomatch nicht gesetzt ist und ein Kommando Metazeichen zur Dateinamensexpansion benutzt, so wird, wenn keine Uebereinstimmung festzustellen ist, eine Fehlernachricht ausgegeben. So wird z.B. beim Kommando:

```
ls a*
```

ein Fehler zurueckgegeben:

```
no match.
```

Ist jedoch die nonomatch-Variable gesetzt, so ist es fuer die Dateinamenexpansion kein Fehler, nicht zu irgendeiner existierenden Datei zu passen. Dafuer wird das Muster mit der Nachricht

```
a* not found
```

zurueckgegeben.

Es ist fuer primitive Muster dennoch ein Fehler, missgebildet zu sein; z.B. gibt das Kommando

```
echo [
```

dennoch den Fehler

```
Missing ]
```

zurueck.

DEFAULT:

unset by default.

Siehe auch: Teil 3 - Dateinamensubstitution.

7.1.11. path

Syntax:

```
set path=/directory...
```

Beim login untersuchte die Shell die Directories, die in der path-Variablen spezifiziert sind, um eine hash-table der Dateien jeder Directory zu schaffen. Diese hash-table wird zur Kommandoliste, die Shell bekannt ist.

Der Punkt (die gegenwaertige Arbeitsdirectory) wird stets fuer jedes Kommando untersucht. Folglich sollte er stets in der path-Variablen enthalten sein.

Wenn es keine path-Variable gibt, werden nur Kommandos, die einen vollstaendigen Pfadnamen spezifizieren ausgefuehrt, wie im Kommando

```
/bin/ls
```

Der Pfad ist ., /bin und /usr/bin. Fuer den Superuser ist der Suchpfad /etc, /bin und /usr/bin.

Die C-Shell wird zuerst in der gegenwaertigen Arbeitsdirectory (mit einem Punkt angezeigt ".") forschen, wenn die C-Shell einen Dateinamen findet, der mit dem Namen des Kommandos identisch ist; wird C-Shell versuchen, die Datei auszufuehren, als waere sie ein Programm. Arbeitet die Datei nicht richtig aus, so wird C-Shell den Fehler auf dem Standard-Error-Kanal (der gewoehnlich das Terminal ist) mitteilen.

Wird die Datei in der gegenwaertigen Directory nicht gefunden,

sucht C-Shell in der naechsten Directory (in diesem Falle /bin). Die /bin-Directory enthaelt die ueblichsten WEGA-Kommandos. Wird das Kommando/die Datei in /bin gefunden, so wird sie ausgefuehrt, falls nicht, forscht C-Shell in /usr/bin nach.

Andere Directories koennen im Forschungspfad eingeschlossen sein. Wenn der Pfad Leerzeichen enthaelt, muss er in Klammern gesetzt werden.

DEFAULT:

```
path = (. /bin /usr/bin)
```

Siehe auch: Teil 5 - Built-in-Kommandos (Rehash)

7.1.12. prompt

Syntax:

```
set prompt=string
```

Das prompt ist ein Signal von C-Shell, dass das Betriebssystem das letzte Kommando beendet hat und bereit ist, ein anderes entgegenzunehmen.

Das implizite prompt fuer den regulaeren Nutzer ist das Prozentzeichen "%", das implizite prompt fuer den Superuser ist ein Doppelkreuz "#".

Die prompt-Variable kann modifiziert sein, um nuetzlichere Informationen zu liefern. Das ueblichste Beispiel dafuer ist die Plazierung der Kommandonummer innerhalb des prompt. Das Kommando:

```
set prompt = "% \!"
```

wird das prompt

```
% 1
```

erzeugen. Beachte, dass Anfuehrungszeichen verwendet werden muessen, wenn das Prompt-String ein Leerzeichen enthaelt.

Die Zahl waechst um 1 fuer jedes Kommando an. Die Kommandonummer ist eine nuetzliche Information, wenn sie mit der History-Funktion verwendet wird (siehe unten).

DEFAULT:

```
prompt=%
```

Siehe auch: Teil 4 - Die History-Funktion

7.1.13. shell

Syntax:

```
set shell=/path/shell.name
```

Die shell-Variable spezifiziert die Shell, die bei login verwendet wird, entweder /bin/csh fuer C-Shell oder /bin/sh fuer Bourne-Shell. Diese Variable wird (anfaenglich) in der /etc/passwd-Datei gesetzt und kann entweder im Dialog oder im Hauptteil eines C-Shell-Skripts neu gesetzt werden.

DEFAULT:

```
shell=/bin/csh
```

Siehe auch: csh(1) und sh(1)

7.1.14. status

Syntax:

```
(Wird nicht vom Terminal aus gesetzt)
```

Die Status-Variable enthaelt den exit-status, der vom letzten Kommando zurueckgegeben wird.

Wird ein Kommando erfolgreich (ohne Fehler) ausgefuehrt, so wird die Status-Variable auf 0 gesetzt. Misslingt die richtige Ausfuehrung des Kommandos (z.B. wenn es einen Syntaxfehler oder eine andere Art von Fehlern gibt), wird die status-Variable auf einem Wert, der nicht 0 ist, gesetzt.

Diese Variable ist in C-Shell-Skripten nuetzlich, um Ausfuehrungsfehler der C-Shell-Skript-Kommandos zu melden.

Anmerkung: Das Kommando set wird fuer die status-Variable stets einen Wert von 0 zeigen, da das Kommando set erfolgreich ausfuehrt wurde. Der wahre Wert der status-Variable kann mit dem Kommando

```
echo $status
```

ausgegeben werden.

DEFAULT:

```
status=0
```

Siehe auch: Teil 5 - Built-in-Kommandos (exit)

7.1.15. term

Syntax:

```
set term=terminal.type
```

Die term-Variable enthaelt den Typ des verwendeten Terminals. Diese Information ist fuer Programme von Bedeutung, die spezifische Kommandos verwenden, um den Cursor zu manipulieren (z.B. der visuelle Editor vi(1)).

Die Shell passt den Zwei-Zeichen-Code von der term-Variable an die Beschreibungszeile des Terminals in der /etc/termcap-Datei an. Ein Beispiel fuer die erste Zeile des Beschreibungscodes des P8000-Terminals (kompatibel ADM31) wird unten gezeigt:

```
P8|P8000-A|P8000-Termianl komp. ADM31
```

Die term-Variable wird anfaenglich in der /etc/ttytype-Datei gesetzt; im Dialog oder im Hauptteil eines C-Shell-Skripts kann sie jedoch neu gesetzt werden.

Unter Nutzung des in der term-Variablen gefundenen Zwei-Zeichen-Codes passt C-Shell jenen Code dann an den Eintrag in der /etc/termcap-Datei an, um das "Terminal-to-WEGA" Software Interface zu initialisieren.

DEFAULT:

```
term=P8
```

Siehe auch: /etc/termcap-Datei und termcap(5)

7.1.16. time

Syntax

```
set time=N
```

Die time-Variable steuert die automatische Zeitmessung der Kommandos. Ist sie gesetzt, so hat jedes Kommando, das laenger als "N" CPU Sekunden dauert, eine Zeile zur Folge, die die Nutzerzeit, Systemzeit, reale Zeit und den Prozentsatz der Verwendung (das Verhaeltnis der Nutzerzeit + Systemzeit zur realen Zeit) zeigt.

Die Ausgabe hat folgendes Format: (Die Ausgabe des time-Kommandos ist die letzte Zeile des Beispiels):

```
% ps
   PID TTY TIME CMD
 23399 2   0:26 -csh
 25207 2   0:03 ps
0.2u 2.8s 0:05 60%
```

DEFAULT:

unset by default

Siehe auch: time(1) Teil 5 - Built-in-Kommandos (time)

7.1.17. verbose:

Syntax:

```
set verbose
```

Die verbose-Variable funktioniert wie die echo-Variable. Ist sie gesetzt, wird jedes Kommando wiedergegeben, wie im folgenden Beispiel:

```
% ls
ls
csh.01 csh.02 csh.03 csh.04 csh.05
```

Die verbose-Variable wird auch durch die "-v" Kommandozeilenoption beim csh-Aufruf gesetzt, wie in

```
csh -v test
```

Die verbose-Variable bewirkt, dass die Argumente jedes Kommandos nach der History-Substitution ausgegeben werden (anders als die echo-Variable, die bewirkt, dass die Argumente vor der History-Substitution ausgegeben werden).

DEFAULT:

unset by default

Siehe auch:
Teil 3 - Dateinamensubstitution
Teil 4 - Die History-Funktion

7.2. Vordefinierte Variablenwerte

Wenn keine anderen Werte fuer diese Variablen der C-Shell festgelegt sind, werden durch C-Shell (oder eine Funktion der C-Shell) beim login die folgenden impliziten Eigenschaften gesetzt. Das Kommando set zeigt die folgende Liste vordefinierter C-Shell-Variablen und deren Standardwerte.

```
argv    ()
home    /path/home.directory
path    (. /bin /usr/bin)
prompt  %
shell   /bin/csh
status  0
term    P8
```

Vordefinierte C-Shell-Variablen:

Name	Funktion
argv	Tracks command arguments
child	The number of last background command
echo	Echos each command
history	Sets the Length of the history memory
home	Sets the home directory
ignoreeof	sets response to control-D commands
mail	Sets mailbox and frequency of mail checks
noclobber	Sets file over-write protection
noglob	Sets filename expansion inhibitor
nonomatch	Sets "no match" error override
path	Sets search path for commands
prompt	Sets prompt
shell	Sets shell (/bin/csh or /bin/sh)
status	Tracks exit status of commands
term	Sets terminal type
time	Sets frequency of "time" report command
verbose	Sets verbose echoing of command lines

7.3. Nutzerdefinierte Variablen

Zusaetzlich zu den eingebauten Variablen, mit denen C-Shell ausgestattet ist, koennen Variablen durch jeden Nutzer festgelegt und manipuliert werden.

Eine Variable kann jede Zeichenkette sein, sie kann beliebige Werte erhalten, ,Zahlen, Buchstaben, Datei- oder Directorynamen, Strings aus Zahlen und Buchstaben usw.

Ein einfacher Handgriff ist es einen Directorynamen zu einer einfachen Buchstabenvariablen zu setzen, wie im folgenden Beispiel:

```
set M=/usr/doc/man/man1
```

Das Kommando

```
cd $M
```

ist dasselbe Kommando wie

```
cd /usr/doc/man/man1
```

Ein anderer Trick ist es eine Variable mit dem Datum zu setzen, wie mit dem folgenden Kommando:

```
set DATE=`date`
```

Die Variable DATE kann dann auf beliebige Weise manipuliert werden.

7.4. Nuzerdefinierte Variablensubstitution

Syntax:

```
set name=value
set name[N]=value
```

Jedes Wort - oder Zeichenstring kann als C-Shell-Variable gesetzt werden. Gegeben ist das Kommando, um eine Variable, genannt DATE, mit dem Datum zu setzen.

```
set DATE=`date`
```

Um den Inhalt der DATE-Variablen zu sehen, erzeugt das Kommando

```
echo $DATE
```

die Ausgabe:

```
Fri Dec 17 17:24:08 MEZ 1986
```

Anmerkung: Variablen muessen mit einem Dollarzeichen vor dem Namen aufgerufen werden.

Auf jedes Element im String kann mit einem Index, der am Variablennamen angehaengt wird, verwiesen werden, wie in:

```
echo $DATE[3]
```

um das 3. Wort im String zu erzeugen:

```
17
```

Um die Anzahl der Woerter in der DATE-Variablen zu pruefen, erzeugt das Kommando

```
echo $#DATE:
```

den Wert

```
6
```

Als Teil einer Kommandofolge im Hauptteil eines C-Shell-Skripts koennte es nuetzlich sein, zu bestimmen, ob eine Variable gesetzt wurde oder nicht. Das Kommando

```
echo $?DATE
```

liefert eine "1", wenn DATE gesetzt ist und eine "0", wenn nicht.

Syntax der Variablensubstitution:

Syntax	Bedeutung
$name	The variable name
${name}	Insulate name from surrounding characters
$name[N]	Argument N of name
${name[N]}	Argument N of name
$#name	Give number of words in the variable
${#name}	Give number of words in the variable
$?name	Substitute "1" if name is set "0" if not
$number	Same as $argv[number]
${number}	Same as ${argv[number]}

Metazeichen der Variablensubstitution:

Zeichen	Bedeutung
$N	Same as Sargv[N]
$0	Command file name (zeroth argument)
$?0	1 if current input filename is known, 0 if not
$*	Same as $argv[*]
$$	Process I.D. number of parent shell

7.5. Verwendung von Modifizierern in der Variablensubstitution

Die Modifizierer ":h", ":t" und ":r" bewirken fuer Variablen das Gleiche wie bei Kommandos.

Wenn z.B. der grosse Buchstabe U zur Directory /z/deck/Util/New.csh gesetzt ist, demonstrieren die folgenden Kommandos die Anwendung der Modifizierer.

```
echo $U
/z/deck/Util/New.csh
```

Der :h-Modifier entfernt den nachstehenden Dateinamen (/New.csh) und belaesst den Kopf

```
echo $U:h
/z/deck/Util
```

Der :r Modifier entfernt den Dateinamensuffix (den csh-Teil nach dem Punkt) und belaesst die Wurzel

```
echo $U:r
/z/deck/Util/New
```

Der :t Modifier entfernt den Kopf (/z/deck/Util) und belaesst den Restteil des Namens

```
echo $U:t
New.csh
```

Wenn isolierende geschweifte Klammern in der Kommandoform erscheinen, muessen die Modifizierer innerhalb der Klammern stehen.

Modifizierer fuer Variablensubstitution

Modifizierer	Wirkung
th	Head only
:t	Tail only
:r	Root only

Siehe auch: Abschnitt 4.5. - Die Modifizierung vorangegangener Kommandowoerter Teil 5 - Built-in Kommandos (modifiers)

8. Das csh-Kommando und C-Shell-Skripte

8.1. Das csh-Kommando

Syntax:

```
csh
csh [-option] filename
filename
```

Die C-Shell kann im Dialog als Kommando aufgerufen werden. Als Kommando und ohne ein Argument verwendet, wie im Kommando

```
csh
```

wird eine neue C-Shell (als child-Prozess bezeichnet) erzeugt. Es wird der Inhalt der ~/.cshrc-Datei (aber nicht der ~/.login Datei) neu ausgefuehrt und eine neue Umgebung mit einer neuen History-Liste geschaffen.

Durch sich selbst schafft das csh-Kommando eine neue Arbeitsumgebung mit den default-Werten (jenen, die in der Umgebung und der ~/.cshrc Datei festgelegt sind). Das ist in Situationen nuetzlich, wo der Nutzer wuenscht, jede neue Variable, aliases oder History-Hinweise von der unmittelbaren Arbeitsumgebung wegzuraeumen, um ein Shell-Skript oder andere Shell-Manipulationen ohne ausloggen zu ueberpruefen. Das csh-Kommando ist von groesserer Nuetzlichkeit, wenn es vom Innern eines laufenden Programms (d.h. vi(1), more(1) und write(1) aufgerufen wird.

Siehe Abschnitt 9.2. - Andere Dateien, die sich auf C-Shell beziehen.

8.2. Aufruf von csh zur Ausfuehrung eines Shell-Skripts

Syntax:

```
csh filename
```

Mit einem Dateinamenargument, in dem filename der Name der Datei ist, die ein oder mehrere Kommandos enthaelt, (die Datei ist als Skript oder Shell-Skript bekannt), versucht die C-Shell die Datei auszufuehren.

Z.B. ist eine Datei gegeben, die test genannt wird und folgende Zeilen hat:

```
ls
who
pwd
date
```

Alle Kommandos in der Datei werden nacheinander durch die Eingabe

von

```
        csh test
```

ausgefuehrt, um die Ergebnisse zu erzeugen:

```
    csh.01
    csh.02
    csh.03
    csh.04
    csh.05
    csh.06
    csh.9T
    refer.sheet

    karen   tty0    Nov     1 08:10
    cheryl  console Nov     1 14:02
    deck    tty2    Nov     1 10:37
    mike    tty6    Nov     1 14:43
    carol   tty8    Nov     1 08:35
    george  tty9    Nov     1 08:35

    /z/deck/Util/New.csh

    Mon Nov 1 15:19:39 MEZ 1986
```

Das csh-Kommando schafft (spaltet ab) eine neue C-Shell (erwaehnt auch als child-Prozess). Diese neue C-Shell liest und fuehrt den Inhalt der ~/.cshrc Datei aus und liest dann den Inhalt des Shell-Skripts Zeile fuer Zeile durch und fuehrt dann ihrerseits jedes Kommando aus (Es wurden zwischen den Ausgabeabschnitten Leerzeichen hinzugefuegt, um das Beispiel besser lesbar zu machen).

Wenn alle Zeilen gelesen wurden, stirbt die neue Shell (das Child) und die Steuerung kehrt zur Stamm-(login)-C-Shell zurueck. (Siehe Teil 10 zur Erlaeuterung dieses Prozesses.)

Das Ausfuehren eines Shell-Skripts mit dem csh-Kommando ist gleichbedeutend mit dem Ausfuehrbarmachen des Shell-Skripts mit dem chmod(1)-Kommando und der Eingabe des Namens der Datei, als waere sie ein Kommando. Das wird im folgenden Beispiel demonstriert:

```
        chmod 777 filename
        filename
```

8.3. Verwendung von C-Ausdruecken in Skripten

Shell-Skripten koennen komplexer werden als eine Sammlung von einzelnen Zeilenkommandos. Eine Reihe von eingebauten Kommandos kann in Ausdruecken verwendet werden, die die Operatoren aehnlich verwenden wie die der C-Programmiersprache.

Diese Ausdruecke koennen mit den @(set), exit, if und while-Kommandos verwendet werden. Nachfolgend eine Uebersicht der verfuegbaren Operatoren:

Vergleichsoperatoren in C-Shell-Skripten

Zeichen:	Bedeutung:
||	Logical "or"
&&	Logical "and"
|	Bitwise inclusive "or"
^	Bitwise exclusive "or"
&	Bitwise "and"
== !=	Equal to; Not Equal to
<= >= < >	Less than or equal to, Greater than or equal to, Less than, Greater than
<< >>	Shift operators
+ -	Add, Subtract
* / %	Multipli, Divide, Modulo
!	Negation
~	Complement
( )	Parenthesis -- bracket expressions

Die Prioritaet waechst von oben nach unten. Operatoren auf derselben Zeile sind von links nach rechts assoziativ.

Anmerkung: Die "equal to" und "not equal to"-Operatoren ("==" und "!=") vergleichen Argumente als Strings. Alle anderen Operatoren operieren mit Zahlen.

Der Ausdruck

```
if $argv[1] == temp
```

wird versuchen, den Inhalt von $argv[1] mit einer Datei, genannt temp, zu vergleichen, waehrend der Ausdruck

```
if $argv[1] >= temp
```

einen Syntaxfehler erzeugen wird.

Fehlende Argumente werden als "0" angesehen. Wenn die Variable a zu "1" gesetzt ist, wird der Ausdruck

```
if ( $a > ) echo hi
```

"hi" ausgegeben.

Das Ergebnis aller Ausdruecke sind strings, die Dezimalzahlen repraesentieren. Alle Operatoren muessen durch Leerzeichen vom umgebenden Text getrennt sein, ausser die folgenden Zeichen:

```
ampersand       "&"
pipe            "|"
less than       "<"
greater than    ">"
parenthesis     "(" und ")"
```

die neben den Rechengroessen plaziert werden koennen, wie im Beispiel:

```
#
set a=4
if ($a>3) who
```

das das who-Kommando ausfuehrt.

8.4. Beispiele von Shell-Skripten, die Operatoren verwenden

8.4.1. And und or-Operatoren

Die folgenden Operatoren sind in Bedingungsausdruecken nuetzlich, wo Werte von Ausdruecken und Kommandos berechnet werden muessen - "true" und/oder "false".

Die folgende "Gueltigkeitstabelle" veranschaulicht die Ergebnisse dieser Operatoren:

```
------------------------------------------------------------
        0 = false
        1 = true
        a = left side of the operator
        b = right side of the operator

                |   &
        a b     |   &&  ^

        0 0     0   0   0
        0 1     1   0   1
        1 0     1   0   1
        1 1     1   1   0
------------------------------------------------------------
```

Die folgende Syntax wird verwendet:

```
if (expression.a operator expression.b) command
```

und mit dem logischen "or"-Operator "||", wenn expression.a "true" ist - wird ihm ein Wert von "1" gegeben und expression.b ist "false" - wird ihm ein Wert von "0" gegeben, dann ist das naechste Ergebnis dieser 2 Ausdruecke "true" (wird ihm ein Wert von "1" gegeben - siehe 3. Zeile der obigen Tabelle) und ein Versuch wird unternommen, das Kommando auszufuehren. Betrachte die folgenden Shell-Skripte:

|| Logical "or"

```
#
set a=1
set b=9
if ($a == 1 || $b == 1) who
```

This script executes the who command.

&& Logical "and"

```
#
set a=1
set b=9
if ( $a == 1 && $b == 9 ) who
```

This script executes the who command.

| Bitwise inclusive "or"

```
#
set a=1
set b=9
if ( $a == 1 | $b == 1 ) who
```

This script executes the who command.

^ Bitwise exclusive "or"

```
#
set a=1
set b=9
if ( $a == 1 ^ $b == 1 ) who
```

This script executes the who command.

& Bitwise "and"

```
#
set a=1
set b=9
if ( $a == 1 & $b == 9 ) who
```

This script executes the who command.

8.4.2. Vergleichsoperatoren

== Equal to

```
#
set a=1
if ($a == 1 ) who
```

This script executes the who command.

!= Not Equal to

```
#
set a=1
if ( $a != 9 ) who
```

This script executes the who command.

<= Less than or equal to

```
#
set a=1
if ( $a <= 9 ) who
```

This script executes the who command.

>= Greater than or equal to

```
#
set a=1
if ( $a >= 1 ) who
```

This script executes the who command.

< Less than

```
#
set a=1
if ( $a < 2 ) who
```

This script executes the who command.

> Greater script executes the who command.

Greater than

```
#
set a=1
if ( $a > 0 ) who
```

This script executes the who command.

8.4.3. Verschiebeoperatoren

<< (Left Shift Operator) Das Verschieben ist ein binaere Operation; das Verschieben einer Zahl um 1 nach links, ist das gleiche, wie das Multiplizieren mit 2. Aus dem gleichen Grunde ist das Verschieben einer Zahl um 3 nach links (n << 3) dasselbe, wie das Multiplizieren dieser Zahl mit 8.

```
#
@ x=2
@ Y=( $x << 1 )
if ( $y == 4 ) who
```

```
    This script executes the who command.
```

>> (Right Shift Operator) Das Verschieben nach rechts ist Dividieren durch 2, das Verschieben um 3 nach rechts (n>>3) ist das gleiche, wie das Dividieren durch 8.

```
    #
    @ x=4
    @ y=( $x >> 1 )
    if ( $y == 2 ) who

    This script executes the who command.
```

8.4.4. Mathematische Operatoren

+ Addition

```
    #
    @ a=1+3
    if ( $a == 4 ) who

    This script executes the who command.
```

Beachte die Verwendung des at-Zeichens "@" im Zusammenhang mit einer mathematischen Operation. Es setzt fuer den Wert einer Variablen eine Dezimalzahl, anstelle eines strings.

- Subtraktion

```
    #
    @ a=9-1
    if ( $a == 8 ) who

    The script executes the who command.
```

* Multiplikation

```
    #
    @ a=2*4
    if ( $a == 8 ) who

    This script executes the who command.
```

/ Division

```
    #
    @ a=8/2
    if ( $a == 4 ) who

    This script executes the who command.
```

% Modulo

```
    #
```

```
@ a=9%4
if ( $a == 1 ) who
```

This script executes the who command.

8.4.5. Andere Operatoren

`!` Negation

```
#
set a=1
if ( $a !=9 ) who
```

This script executes the who command.

`( )` Parenthesis -- bracket expressions

```
#
set a=1
set b=9
if ( $a == 1 & $b == 9 ) who
```

This script executes the who command.

8.5. Operatoren zur Suche von Dateien

Die Operatoren zur Suche von Dateien werden verwendet, um die Eigenschaften einer gegebenen Datei zu pruefen. Bei Anwendung mit einer if-Aussage wird er mit folgende Syntax verwendet:

```
#
if ( -operator filename ) command
```

Im obigen Beispiel wird das folgende Kommando ausgefuehrt, wenn die Datei filename die Voraussetzungen erfuellt, die vom Operator gestellt werden.

Das Ausrufungszeichen "!" wird verwendet, um zu pruefen, ob die Voraussetzung nicht erfuellt ist, wie in der Syntax:

```
#
if ( ! -operator filename ) command
```

Die folgenden Operatoren zur Suche von Dateien sind verfuegbar:

Zeichen:	Bedeutung:
r	read access
w	write access
x	execute access
e	existence

o	ownership
z	zero size
f	plain file
d	directory

Existiert die Datei nicht oder ist kein Zugriff moeglich, geben alle Ermittlungen "0" zurueck (der Wert fuer einen falschen Ausdruck).

Sollten detaillierte Status-Informationen erwuenscht sein, sollte das Kommando ausserhalb eines Ausdrucks ausgefuehrt und die Variable status geprueft werden. (Siehe Abschnitt 7.1 - "Vordefinierte C-Shell-Variable" - Die Status-Variable).

Ein Beispiel fuer einen in einem Shell-Skript verwendeten Operator zur Suche von Dateien ist:

```
#
if (-e test) echo "The test file is here"
```

Wenn die "test" genannte Datei in der gegenwaertigen Arbeitsdirectory existiert, wird dieser Skript "The test file is here" wiedergegeben.

8.6. Optionen des csh-Kommandos

Syntax:

```
csh -option filename
```

-c Kommandos werden vom (einzelnen) folgenden Argument gelesen, das vorhanden sein muss. Alle verbleibenden Argumente werden in argv placiert.

-e Die C-Shell wird beendet, wenn irgendein Kommando anomal endet oder einen nonzero exit status ergibt.

-f Die C-Shell beginnt schneller, da sie weder Kommandos von der Datei ~/.cshrc in der Home-Directory des Aufrufers sucht noch ausfuehrt.

-i Die C-Shell ist interaktiv. Shells sind ohne diese Option interaktiv, wenn ihre Ein- und Ausgaben Terminals sind. Kommandos werden analysiert, aber nicht ausgefuehrt. Das unterstuetzt die syntaktische Ueberpruefung von C-Shell-Skripten.

-s Die Kommandoeingabe wird von der Standard-Eingabe genommen.

-t Eine einzelne Eingabezeile wird gelesen und ausgefuehrt. Ein \ kann am Ende dieser Zeile verwendet werden, um sie auf der naechsten Zeile fortzusetzen.

-v Bewirkt, dass die verbose-Variable gesetzt wird, was bewirkt, dass die Kommandieingabe nach der History-Substitution wiedergegeben wird.

-x Bewirkt, dass die echo-Variable gesetzt wird, so dass Kommandos unmittelbar vor der Ausfuehrung wiedergegeben werden.

-V Bewirkt, dass die verbose-Variable sogar vor Ausfuehrung von .cshrc gesetzt wird.

-X Bewirkt, dass die echo-Variable sogar vor Ausfuehrung von .cshrc gesetzt wird. Nach Bearbeitung der flag-Argumente, falls Argumente geblieben doch weder die -c, -i, -s oder -t-Option gegeben sind, wird das erste Argument (Argument 0 -der Name des C-Shell-Skripts) als Name einer Datei von Kommandos angesehen, die ausgefuehrt werden sollen. C-Shell eroeffnet diese Datei und bewahrt ihren Namen fuer moegliche Re-Substitutionen durch $0. Die C-Shell fuehrt eine Standard (Bourne) Shell aus, wenn das erste Zeichen eines Skripts kein Doppelkreuz ist. Alle verbleibenden Argumente werden in der Variablen argv placiert.

Option	Bedeutung
-c	single argument only
-e	exit on error
-f	faster (doesn't read ~/.cshrc)
-i	interactive -- prompts for input
-s	take commands from standard input
-t	read and execute single line of input
-v	verbose -- echo commands after history
-x	verbose -- echo commands before history
-v	verbose variable set before ~/.cshrc is read
-x	verbose set before ~/.cshrc and before history subs

8.7. Kommentarzeilen in Shell

Im allgemeinen bewirkt ein mit einem Doppelkreuz "#" beginnendes Wort fuer die C-Shell, dass jenes Wort und alle folgenden Zeichen bis zu einer neuen Zeile ignoriert werden.

Als erstes Zeichen in der ersten Spalte auf der ersten Zeile eines Shell-Skripts bedeutet das Doppelkreuz: "Benutze die C-Shell (/bin/csh), um diesen Skript auszufuehren." Andere Zeichen in der ersten Position bedeuten etwas anderes. Bitte beachte die folgende Tabelle fuer spezifische Bedeutungen:

erste Zeile im Skript	Bedeutung
#!sh	Use the "Bourne" shell -- /bin/sh
#!csh	Use the "C" shell -- /bin/csh
#!	Error, can't determine which shell
#!/xxx	Use the shell in the file /xxx
X	Any character other than "#" uses/bin/sh

9. C-Shell-Dateien

9.1. Start-Dateien

Die Buchstaben "rc" haengen an bestimmten Dateinamen. Diese Initialien stehen fuer "read command". Folglich ist die Datei ~/.cshrc die "csh read command" Datei. Jedes Mal, wenn die C-Shell (csh) aufgerufen wird, liest sie den Inhalt dieser Datei und fuehrt ihn aus.

"Rc"-Dateien werden durch die Shell (und andere Programme) zu verschiedenen Zeilen gelesen, um verschiedene Aufgaben zu erfuellen. Die "/etc/rc_csh" genannte Datei wird immer dann gelesen, wenn das Betriebssystem gestartet wird. Eine andere "rc"-Datei ist "~/.exrc", die immer dann gelesen wird, wenn der ex(1) Editor oder vi(1) Editor aufgerufen wird.

Ausser diesen "rc"-Dateien gibt es eine Reihe von anderen Dateien, die fuer die C-Shell von Bedeutung sind.

Die nachfolgende Liste zeigt die eingebauten Dateien, die fuer das System wichtig sind. Andere Dateien koennen, wenn erforderlich, von jedem Nutzer definiert und verwendet werden.

Special Files

~/.cshrc
~/.login
~/.logout
~/.exrc
~/.profile
/bin/sh
/bin/csh
/dev/null
/etc/cshprofile
/etc/passwd
/etc/group
/tmp/sh*

Es gibt keine eindeutigen Regeln fuer die Implementation der Initialisierungsdateien (~/.cshrc; ~/.login, ~/.logout und ~/.exrc). Die gelieferten Beispiele sind lediglich Vorschlaege; sie wurden dargestellt, um einen begrenzten Bereich von Moeglichkeiten vorzuschlagen; einige erfahrene Nutzer empfehlen viel kompliziertere Dateien. Es ist Geschmackssache.

Anmerkung: Die Konvention "~/.file" weist auf die Tatsache hin, dass die Datei in der Home-Directory des Nutzers sein muss. Entweder sucht C-Shell oder das Betriebssystem (WEGA) in jener spezifischen Directory nach der besonderen Datei.

9.1.1. ~/.cshrc

Die "~/.cshrc" Datei wird bei der Ausfuehrung jeder C-Shell gelesen. Beim login wird die "~/.cshrc"-Datei vor der "~/.login"-Datei gelesen. Die ~/.cshrc-Datei sollte Kommandos enthalten, die fuer jede Shell benoetigt werden.

```
# ~/.cshrc file
set ignoreeof
set history=15
set mail=(5 /usr/spool/mail/deck)
set prompt="`whoami` # \! > "
stty erase ^H kill ^X
umask 2
set U=~/Util/New.csh
alias h   history
alias bye logout
```

.cshrc file

Die ~/.cshrc Datei wird durch die C-Shell gelesen und muss folglich mit einem Doppelkreuz beginnen. Der ".cshrc"-Text wird als Kommentar ignoriert.

set ignoreeof

Die ignoreeof-Variable veranlasst die C-Shell, dass versehentliche control-D's und folglich versehentliche "logout's" zu ignorieren.

Siehe auch: Abschnitt 7.1 - Vordefinierte C-Shell-Variable

set history=15

Die history-Variable steuert die Groesse der Historyliste Siehe auch: Abschnitt 7.1 - Vordefinierte C-Shell-Variable Teil 4 - Die History-Funktion.

set mail =(5 /usr/spool/mail/deck)

Die mail-Variable setzt die Haeufigkeit der new-mail-Kontrollen (in diesem Falle 5 Sekunden zwischen den Kontrollen). Das 2. Argument setzt den Standort des Briefkastens. Die Zeitvariable kann weggelassen werden, ist sie jedoch enthalten, muss der Ausdruck in Klammern gesetzt werden.

Siehe auch: Abschnitt 7.1 - Vordefinierte C-Shell-Variable.

set prompt="`whoami` # \! >"

Die prompt-Variable setzt das Promptzeichen. In diesem Fall

besteht das Prompt aus: dem "whoami" Kommando, das, da es in zurueckgekippten Anfuehrungszeichen steht, durch den login-Namen des Nutzers ersetzt wird; einem Doppelkreuz; Schraegstrich-Ausrufungszeichen wird durch eine Zahl ersetzt, die mit jedem Kommando um 1 anwaechst; und einem "groesser-als-Zeichen", das hier verwendet wird, um das Ende des Prompts zu zeigen - in diesem Zusammenhang hat es keine spezielle Bedeutung (Metazeichen). Andere Teile des Prompts koennen spezielle Terminal-Screen-Attribute einschliessen (reverse video, flashing usw.).

Siehe auch: Abschnitt 7.1. Vordefinierte C-Shell-Variable

stty erase ^H kill ^X

stty(1) ist ein ausserhalb der C-Shell befindliches Kommando (es ist kein eingebautes Kommando), das verschiedene Terminalkennwerte setzt. Hier wird das erase-Zeichen zu control-H (der traditionellen Ruecktaste) gesetzt und das Kill-Zeichen zu control-X.

Siehe auch: stty(1)

umask 2

Das eingebaute Kommando umask setzt den Schutzmode der Datei fuer neu geschaffene Dateien. Ein umask Wert von 2 setzt einen Dateischutzmode, der dem Nutzer und der Gruppe eine Lese- und Schreiberlaubnis erteilt und jedem anderen eine Leseerlaubnis (rw-rw-r--).

Siehe auch: Abschnitt 5.2 - Allgemeine Kommandos, die nach dem Prompt eingegeben werden.

set U=~/Util/New.csh

Das ist eine nutzerdefinierte Variable. Das Kommando cd $U entspricht dem Kommando cd ~/Util/New.Csh. $U ist eine Kurzschrift.

Siehe auch: Abschnitt 7.3. - Nutzerdefinierte Variable

alias h history

alias bye logout

Wird ein alias vom Innern einer Subshell benoetigt, sollte das alias-Kommando in der ~/.cshrc-Datei plaziert sein, da jede neue Shell mit ihrer eigenen alias-Liste beginnt (aliases, die in der ~/.login-Datei gesetzt sind, werden nicht auf neue Shells uebertragen).

Siehe auch: Abschnitt 5.3 - Umgebungskommandos, die nach dem Prompt eingegeben werden.

9.1.2. ~/.login

Die ~/.login Datei wird durch die Shell beim login unmittelbar nach der ~/.cshrc-Datei gelesen. Sie wird einmal beim login gelesen und sollte folglich Kommandos enthalten, die entweder jenseits der Shells uebertragen (siehe Teil 10 zur Eroerterung der Umgebungsvariablen und des Uebertragens) oder Kommandos, die nur ein Mal am Anfang der Sitzung auf dem Terminal benoetigt werden. (Unter diese Kategorie koennte ein Erinnerungsprogramm oder das calendar(l)-Programm fallen.)

```
        # .login file
        setenv EXINIT "set number wm=20 | version"
        setenv HOME /z/deck
        setenv PATH ".:/usr/bin:/bin:~/bin:usr/games
        setenv SHELL /bin/csh
        set prompt="% "
        echo " "
        cat ~/.reminder
        echo " "
        calendar
```

Beispiel einer ~/.login-Datei

.login file

Die .login-Datei ist ebenfalls ein Shellskript, der von C-Shell gelesen wird und muss folglich mit einem Doppelkreuz beginnen. Der ".login"-Text wird als Kommentar ignoriert.

setenv EXINIT "set number wm=20 | version"

Die EXINIT-Variable nimmt die Stelle der ~/.exrc-Datei ein - sie wird vom ex-Kommando zum Setzen von ex-Optionen gelesen. In diesem Falle sind die number- und word-margin-Optionen gesetzt und das version-Kommando wird ausgefuehrt. Die EXINIT-Variable ist schneller als die ~/.exrc-Datei, da sie mit jeder neuen Shell uebertragen wird und nicht jedesmal, wenn der Editor aufgerufen wird, neu gelesen werden muss.

Siehe auch: Abschnitt 10.2. - Die erlaeuterten Umgebungsvariablen.

setenv HOME /z/deck

Die HOME-Variable wird zu dieser Home-Directory des Nutzers gesetzt.

Siehe auch: Abschnitt 10.2. - Die erlaeuterten Umgebungsvariablen.

setenv PATH .:/usr/bin:/bin:~/bin:/usr/games

Die PATH-Variable wird zu einer Reihe von Directories gesetzt, die nuetzliche Kommandos enthalten.

Siehe auch: Abschnitt 10.2. - Die erlaeuterten Umgebungsvariablen

setenv SHELL /bin/csh

Die SHELL-Variable setzt eine login-Shell, in diesem Fall C-Shell.

Siehe auch: Abschnitt 10.2. Umgebungsvariablen

set prompt = "% "

Das erste Prompt wird zum impliziten Wert gesetzt: dem Prozentzeichen.

Siehe auch: Teil 7 - Shell-Variable

echo " "

Dieses Kommando liefert einfach ein Leerzeichen zwischen jeglichem existierenden Material auf dem Bildschirm (wie der Nachricht des Tages) und dem folgenden Material.

cat ~/.reminder

Dieser Benutzer hat eine Erinnerungsdatei fuer bevorstehende Ereignisse geschaffen. Dieses Kommando liest die Datei bei jedem login.

echo " "

Dieses Kommando liefert eine weitere Leerzeile.

calendar

Dieses Kommando fuehrt das calendar-Programm aus.

Siehe auch: calendar(1)

9.2. Andere Dateien fuer C-Shell

9.2.1. ~/.logout

Die ~/.logout Datei wird von Shell beim logout gelesen. Sie sollte alle Informationen enthalten, die der Nutzer benoetigt, unmittelbar bevor er die Terminalsitzung beendet.

```
---------------------------------------------------------
        # .logout file
        who
        echo " "
```

```
        date
        echo " "
        cd; calendar
```

Beispiel einer ~/.logout-Datei

```
# .logout file
```

Die .logout-Datei ist ein Shell-Skript, der von C-Shell gelesen wird und folglich mit einem Doppelkreuz beginnt. Der .logout-Text wird als Kommentar ignoriert.

```
who
```

Beim logout informiert das who-Kommando den Nutzer ueber die anderen im System verbleibenden Nutzer.

```
echo " "
```

erzeugt Leerzeile

```
date
```

Das date-Kommando

```
echo " "
```

erzeugt Leerzeile

```
cd; calendar
```

Das cd; calendar-Kommando fuehrt das calendar-Programm in der Home-Directory des Nutzers (um Informationen ueber die Liste des naechsten Tages zu erhalten) aus.

9.2.2. ~/.exrc

Die ~/.exrc-Datei wird gelesen, wenn der ex- oder vi-Editor aufgerufen wird. Die Shell-Variable EXINIT uebt dieselbe Funktion aus.

Die ~/.exrc-Datei wird vom ex- oder vi-Editor gelesen und nicht von C-Shell; folglich sind keine Kommentarzeilen vorhanden.

Siehe auch: WEGA-Programmierhandbuch fuer vorhandene ex- und vi-Optionen.

```
        set number
        set wm=20
        set noredraw
        set slowopen
        set showmatch
        version
```

Beispiel einer ~/.exrc-Datei

9.2.3. /bin/sh

Die Datei enthaelt die Bourne-Shell fuer Shell-Skripte, die nicht mit einem Doppelkreuz "#" beginnen.

9.2.4. /bin/csh

Diese Datei enthaelt die C-Shell fuer Shell-Skripte, die mit einem Doppelkreuz beginnen "#". Siehe auch: csh(1)

9.2.5. /dev/null

Diese Systemdatei ist eine Quelle fuer leere Dateien. Jede Ausgabe, die zu dieser Datei gelenkt wird, ist verloren.

9.2.6. /etc/cshprofile

Die Datei etc/cshprofile ist aehnlich der ~/.cshrc-Datei, ausser dass sie auf der Systemstufe gelesen wird, bevor die ~/.cshrc-Datei gelesen wird. Sie enthaelt fuer jede C-Shell-Arbeitsumgebung Parameter. Sie wird von der login-Shell vor der ~/.cshrc-Shell gelesen.

Siehe auch: cshrc(5)

9.2.7. /etc/passwd

Dieses System ist die Quelle der Home-Directories und anderer grundlegender login-Informationen.

Siehe auch: passwd(5) WEGA-Systemhandbuch

9.2.8. /tmp/sh*

Temporary file for "<<" input.

In Programmen, die die Eingabe vom Hauptteil eines Shell-Skripts mit dem doppelten kleiner als - Zeichen nehmen, fertigt die Shell eine Kopie der Eingabe an und plaziert sie in einer neuen, /tmp/shNNNN genannten Datei, wo NNNN irgendeine von der Shell zur Abgrenzung der /tmp/sh-Datein zugewiesene Zahl ist.

Die Eingabe fuer den Shell-Skript wird dann von der vorlaeufigen Datei in /tmp gelesen.

Siehe auch: Abschnitt 2.9.2. Die Eingabe in einem Skript.

10. Die Umgebung

Die Umgebung ist eine Liste von Variablen, die allen Programmen zur Verfuegung stehen, die von der Shell, welche die Umgebungsvariablen geschaffen hat, ausgefuehrt werden.

Jedes Mal, wenn eine Shell geschaffen (abgespalten) wurde, liest sie in den in der Umgebung gesetzten Variablen. Folglich erbt jede Shell diese Umgebungsvariable und deren Werte.

Umgebungsvariablen koennen als "globale" Variablen angesehen werden, waehrend die C-Shell-Variablen als "lokale" Variablen angesehen werden koennen. Wie die C-Shell-Variablen, so gibt es vordefinierte Umgebungsvariablen und Nutzerdefinierte Umgebungsvariablen.

Da Umgebungsvariablen vererbt werden, muessen sie nur ein Mal beim login in der ~/.login-Datei gesetzt werden, die am Anfang jeder login-Sitzung gelesen wird. Diese Variablen werden auf alle nachfolgenden Shell's uebertragen; sie stehen allen nachfolgenden Programmen zur Verfuegung, ohne sie fuer jedes Programm neu setzen zu muessen.

Umgebungsvariablen werden mit folgender Syntax versehen:

```
setenv NAME value
```

Dieses Kommando kann im Dialog eingegeben oder in eine der "start-up" Dateien geschrieben werden (es wird die ~/.login-Datei empfohlen, da das Kommando nur ein Mal gelesen werden muss). Das Benennen der Umgebungsvariablen mit einem grossen Buchstaben, hat lediglich den Sinn, die zwei Arten von Variablen gesondert zu nennen. Umgebungsvariablen koennen mit jedem Zeichenstring benannt werden.

10.1. Umgebungsvariablen

Umgebungsvariablen sind nuetzlich, wo eine Variable jenseits einer Anzahl von unterschiedlichen Shells verwendet werden muss. Die untenstehende Tabelle zeigt die vordefinierten Umgebungsvariablen und deren Bedeutung.

Umgebungsvariablen

EXINIT	Ex editor initialization variables
HOME	Home directory
LOGNAME	Login name
PATH	Search path for commands
SHELL	Shell being used
TERM	Type of terminal
TERMCAP	File from which the TERM read
TZ	Timezone

10.2. Erlaeuterung der Umgebungsvariablen

Im WEGA-Betriebssystem werden die Umgebungsvariablen von jeder neuen C-Shell gelesen; ihnen werden entsprechende Werte mit entsprechenden Namen gegeben, die in Kleinbuchstaben uebertragen werden. Diese neuen "name/value"-Paare werden zu neuen C-Shell-Variablen, die der neuen Shell lokal zur Verfuegung stehen.

10.2.1. EXINIT

Syntax:

```
setenv EXINIT options
```

EXINIT steht fuer "ex-initialization", die EXINIT-Variable initialisiert die ex-Editor-Optionen.

Ein Beispiel eines Kommandos zum Setzen der EXINIT-Variablen ist:

```
setenv EXINIT "set number wm=20 showmatch | version"
```

Dadurch wird die Zeilenzahlfunktion, die "wrap-margin"-Funktion (auf 20 Zeilen vom rechten Rand) und die "showmatch"-Option des Editors gesetzt und es wird die Version des Editors, immer dann, wenn ex oder sein visuelles Gegenstueck vi aufgerufen werden, ausgegeben.

Beachte, dass zusammengesetzte Kommandos in Anfuehrungszeichen gesetzt werden und die set-Routine ueber pipe durch das versions-Kommando geleitet wird.

Die EXINIT-Variable uebt dieselbe Funktin aus, wie ihr Vorgaenger, die ~/.exrc-Datei, sie ist jedoch schneller, da die EXINIT-Variable automatisch ein Teil der ex-Umgebung ist, waehrend die ~/.exrc-Datei bei jedem Aufruf des Editors gelesen werden muss.

DEFAULT:

```
unset
```

10.2.2. HOME

Syntax:

```
setenv HOME /path/home.directory
```

Die HOME-Variable uebt dieselbe Funktion aus, wie die home-Variable in C-Shell. Sie legt den Standort fuer das cd-Kommando

und den Dateinamen fuer Tilde "~" fest, wenn es als Metazeichen benutzt wird.

Wenn die home-Variable nicht gesetzt ist (weder in der ~/.cshr-Datei, der ~/.login-Datei noch per Dialog) nimmt die home-Variable ihren Wert von der HOME-Variablen. D.h., dass der Wert von HOME auf jede neue C-Shell, wenn sie geschaffen wird (abgespalten wird), uebertragen wird.

Ungeachtet der aufgerufenen Shell, erbt jeder neue Prozess die Werte aller gesetzten Umgebungsvariablen. Beide Shell's lesen (und erben) die in der Umgebung gesetzten Werte.

DEFAULT:

```
HOME /path/users.home.directory
```

sofern nicht anderweitig gesetzt, nimmt die HOME-Variable ihren Wert vom Feld der home-directory der /etc/passwd-Datei.

10.2.3. LOGNAME

Syntax:

```
setenv LOGNAME name
```

Die LOGNAME-Variable beinhaltet den login-Namen des Nutzers.

10.2.4. PATH

Syntax:

```
setenv PATH /path/directory:/path/directory
```

Die PATH-Variable uebt dieselbe Funktion aus, wie die path-Variable fuer C-Shell.

BEISPIEL:

```
setenv PATH .:/usr/bin:/bin:~/bin:/etc:/usr/games
```

DEFAULT:

```
PATH .:/usr/bin:/bin
```

10.2.5. SHELL

Syntax:

```
setenv SHELL /path/shell.program
```

Die SHELL-Variable uebt dieselbe Funktion aus, wie die shell-Variable fuer C-Shell.

BEISPIEL:

```
setenv SHELL /bin/csh
```

DEFAULT:

```
SHELL/bin/csh
```

sofern nicht anderweitig gesetzt, nimmt die SHELL-Variable ihren Wert vom shell-Feld der /etc/passwd-Datei.

10.2.6. TERM

Syntax:

```
setenv TERM terminal.type
```

Die TERM-Variable uebt dieselbe Funktion aus, wie die term-Variable fuer C-Shell.

BEISPIEL:

```
setenv TERM P8
```

DEFAULT:

```
TERM P8
```

sofern nicht anderweitig gesetzt, nimmt die TERM-Variable ihren Wert von der /etc/ttytype-Datei.

10.2.7. TERMCAP

Syntax:

```
setenv TERMCAP /path/directory
```

Die TERMCAP-Variable beinhaltet den Namen der Datei, die fuer die Festlegung der TERM-Kommandos verwendet wird.

BEISPIEL:

```
setenv TERMCAP ~/bin/new.termcap
```

DEFAULT:

```
unset
```

Obwohl die TERMCAP-Variable implizit nicht gesetzt ist, wird der TERM-Wert von der Datei /etc/termcap genommen.

10.2.8. TZ

Syntax:

```
setenv TZ timezone
```

Die TZ-Variable enthaelt die Zeitzone des Geraetes in Stunden gemessen von Greenwich mean time.

BEISPIEL:

```
setenv TZ MEZ-1MES
```

DEFAULT:

Sie wird durch die Datei /etc/rc gesetzt.

Anhang C-Shell-Fehlernachrichten

Das Folgende ist eine kommentierte Teilliste von Fehlernachrichten, die durch die C-Shell als Antwort auf verschiedene Eingabefehler erzeugt werden. Die vollstaendige Liste von Fehlernachrichten folgt zum Schluss.

<< terminator not found

Im Zusammenhang mit einem C-Shell-Skript zeigt dieser Fehler an, dass die Marke, die verwendet wurde, um das Ende der Eingabe anzuzeigen, nicht Teil des Skripts ist. Das folgende Beispiel wuerde so einen Fehler verursachen:

```
# test
ex test << EOF
g/^$/d
w
q
```

Die Loesung besteht im Eintragen von EOF an das Ende des Skripts. Siehe Abschnitt 2.9. - Ein-/Ausgabesteuerung

Alias loop

Wird ein alias festgelegt, das sich selbst aufruft, wird ein alias loop gebildet. Die beiden folgenden Kommandos bilden ein alias loop:

```
alias ls list
alias list ls
```

Beide aliases koennen festgelegt werden, doch die Ausfuehrung beider wird eine Fehlernachricht zur Folge haben. Dieser Fehler loest sich durch unaliasing des Kommandos, das die Fehlernachricht geschaffen hat, wieder auf.

```
unalias alias.name
```

Siehe Abschnitt 5.3. Umgebungsvariablen, die im Dialog eingegeben werden bezueglich der Einzelheiten des alias-Kommandos.

Ambiguous

Dieser Fehler wird erzeugt, wenn ein Dateinamen-Metazeichen "*", "~", "?" in einer Weise verwendet wird, dass es auf eine Reihe von Dateien oder Directorys verweist, wo eine einzelne Datei oder Directory erwuenscht ist, wie beim Kommando

```
cd *
```

Die Fehlernachricht

```
*: Ambiguous.
```

folgt daraus. Die Loesung besteht darin, das Metazeichen durch einen spezifischen Datei- oder Directorynamen zu ersetzen.

Arguments too long

Dieser Fehler ist gewoehnlich mit der Metazeichenexpansion verbunden. Er kann aus dem folgenden Kommando resultieren:

```
echo /*/*/*
```

Die Loesung besteht im Liefern eines spezifischen Arguments.

Cannot determine type of shell to use

Dieser Fehler resultiert aus einem Symbol in der ersten Spalte der ersten Zeile eines Shell-Skripts, das nicht anzeigt, welche Shell zur Ausfuehrung des Skripts verwendet werden soll. Das folgende Skript wird diesen Fehler verursachen:

```
#!
who
```

Die Spezifizierung einer Shell durch ein legitimes Zeichen wird den Fehler aufloesen. Siehe Abschnitt 8.6. "Kommentarzeichen in Shell"

Can't from terminal

Einige Kommandos koennen von einem Terminal nicht ausgefuehrt werden. Z.B. wird das eingebaute onintr-Kommando die Fehlernachricht

```
onintr: Can't from terminal
```

erzeugen, wenn es vom Terminal eingegeben wird. Kommandos, die diesen Fehler erzeugen, sind fuer die Verwendung im Hauptteil eines Shell-Skripts gedacht. Siehe Teil 6 "Die Struktur der C-Shell-Programmiersprache.

Can't make pipe

Der Platz, der fuer temporaere Dateien bei Pipes benoetigt wird, wird im Root-Filesystem "/" zugeordnet. Wenn es voll ist,

gibt es keinen Platz mehr fuer Dateien, die vom Pipe-Mechanismus benoetigt werden. Die Loesung besteht darin, Platz im Root-Filesystem zu schaffen.

command not found

Wenn Shell das Kommando nicht lokalisieren kann, oder der Kommandoname falsch geschrieben wurde, hat das diesen Fehler zur Folge. Der Fehler folgt ebenfalls dann, wenn das Kommando nicht in der Hash-Tabelle von Kommandos festgehalten ist. Siehe Abschnitt 5.2.6. rehash.

Divide by 0

Der Fehler resultiert aus einer mathematischen Operation in einem Shell-Skript, die mit der Division durch 0 verbunden ist.

end not found

Sowohl die foreach- als auch die while-Schleifen erfordern eine abschliessende end-Anweisung.

endif not found

Die if, else-Strukturen erfordern eine endif-Anweisung. Siehe Abschnitt 6.3.

endsw not found

Die switch-Struktur erfordert eine endsw-Anweisung. Siehe Abschnitt 6.4.

Expression syntax

Verschiedene syntaktische Fehler koennen diese Fehlernachricht erzeugen. Die folgende if-Aussage

```
if (a > b) echo HI
```

erzeugt den Fehler

```
if: Expression syntax
```

da die Alphazeichen "a" und "b" mit dem mathematischen Operator groesser als ">" nicht verglichen werden koennen.

Improper mask

Das "mask" verweist auf den umask file protection mode code.

Improper then

verweist auf die then-Aussage in einem if-then kontext.

Interrupted

zeigt eine Programmunterbrechung an.

Invalid variable

Ein Fehler unterlaeuft beim Aufrufen oder Zuweisen von Variablen. Die Loesung besteht im richtigen Aufrufen und Zuweisen von Variablen.

label not found

Im Zusammenhang mit einem Shell-Skript mit einer goto label-Konstruktion muss das label im Skript erscheinen. Es ist ein Fehler, wenn label fehlt.

Missing)

Im Zusammenhang mit einer

```
foreach (list)
     oder
while (list)
```

Aussage muss list in Klammern stehen. Es ist ein Fehler, wenn eine Klammer fehlt.

Missing]

Im Zusammenhang mit einem

```
command [ range ]
```

Kommando muss der Bereich in eckigen Klammern stehen. Es ist ein Fehler, wenn eine Klammer fehlt.

Missing }

Im Zusammenhang mit einem

```
command { list }
```

Kommando muss list in geschweiften Klammern stehen. Es ist ein Fehler, wenn eine Klammer fehlt.

Mod by 0

In einer mathematischen Operation, die die modulo Funktion "fB%" zum Gegenstand hat, kann die rechte Seite der Gleichung nicht 0 sein.

No file for $0

Das Argument Null ist der Name der Datei, die ausgefuehrt wird. In der Datei test mit den folgenden Zeilen:

```
# test
echo $0
```

hat die Ausfuehrung mit dem Kommando

```
csh test
```

(oder das Aendern der Ausfuehrungs-Bits mit dem chmod-Kommando und dessen Ausfuehrung durch den Namen) die Antwort

```
test
```

zur Folge. Dasselbe Kommando im Dialog eingegeben

```
echo $0
```

hat die Fehlernachricht zur Folge.

No home

Jedes Kommando, das von der $HOME-Variablen abhaengig ist (z.B. das cd-Kommando), wird eine Fehlernachricht erzeugen, wenn die $HOME-Variable nicht gesetzt ist.

No match

Wenn die Dateinamen-Expansions-Metazeichen ("*", "[", "]", "{", "}", "?") verwendet werden, ist es ein Fehler, wenn kein Dateiname passt (sofern nicht die nonomatch-C-Shell-Variable gesetzt ist).

No more processes

Nur eine begrenzte Anzahl von Hintergrund-Prozessen kann durch einen einzelnen "Stamm"-(login) Prozess erzeugt werden. Ein Versuch, zusaetzliche Jobs im Hintergrund zu intiieren

hat diesen Fehler zur Folge.

No more words

Entspricht einem Versuch, Woerter nach dem Ende in einer Wortliste zu addressieren, z.B. in einer foreach, while oder case-Aussage.

non-ascii shell script.

Ein Versuch, eine Datei als Shell-Skript auszufuehren, die keine Ascii-Zeichen umfasst.

Not in while/foreach.

In Shell-Skripten kann dieser Fehler aus dem Versuch resultieren, ein Argument ausserhalb der while- oder foreach-Schleife zu addressieren.

Not login shell

Ein Versuch sich von einer Subshell auszuloggen erzeugt diesen Fehler. Die Loesung besteht darin, jede Subshell zu verlassen und dann das logout-Kommando einzugeben.

Out of memory

Die C-Shell kann einen Speicherueberlauf erzeugen.

Output redirection not allowed

Kommandos, die nicht fuer eine Ausgabe-Umlenkung gedacht sind (wie das source-Kommando) erzeugen einen Fehler in folgendem

```
source: Output redirection not allowed.
```

Pathname too long

Wenn der Pfadname zu lang ist, hat das diese Nachricht zur Folge. Die Loesung besteht im Austausch der Directory mit einer kleineren Directory und dem Zugriff auf die Dateien von dort.

Subscript error

Ein Versuch eine Variable mit einem illegalen Subscript-Wert zu indizieren.

Subscript out of range

Im Skript

```
# test
echo $argv[1]
echo $argv[2]
echo $argv[3]
echo $argv[4]
```

wird bei der Kommandozeile

```
test a b c
```

die Anweisung

```
echo $argv[4]
```

den Fehler

```
subscript out of range
```

erzeugen, da es nur 3 Argumente gibt.

Too dangerous to alias that

Ein Versuch dem Wort alias ein alias zuzuweisen, z.B. mit dem Kommando

```
alias alias a
```

hat diesen Fehler zur Folge. Das Problem kann mit dem Kommando

```
alias a alias
```

vermieden werden und zeitigt dieselbe Ergebniss.

Too few arguments

Einige Kommandos erfordern eine spezifische Anzahl von Argumenten.

Too many argument

Einige Kommandos erfordern eine spezifische Anzahl von Argumenten.

Too many)'s

Die Anweisung

```
foreach i (abc))
```

erzeugt den Fehler.

Undefined variable

Ein Versuch, eine nicht-definierte Variable zu verwenden, erzeugt diesen Fehler.

Unmatched `

Kommandos mit nicht geschlossenen backquotes, wie in

```
echo `date
```

erzeugen diesen Fehler.

Unmatched %c

Das ist ein catch-all Fehler, der sich auf alle Kommandos bezieht, die 2 Teile einer Anweisung erfordern. Der Fehler folgt, wenn der 2. Teil fehlt.

Variable Syntax

Ein Syntaxfehler.

Word too long

Manchmal kann C-Shell ein Wort nicht bewaeltigen, dass zu viele Zeichen enthaelt.

Words not ()'ed

Woerter in einer Liste, die nicht in den notwendigen Klammern stehen, wie im Kommando

```
foreach i  a b c d
```

erzeugen den Fehler:

```
foreach: Words not ()'ed
```

Fehlernachrichten:

```
-- Core dumped
%s: File exists
%s: non-ascii shell script
: Event not found
<< terminator not found
Alarm clock
Alias loop
Ambiguous
Ambiguous input redirect
Ambiguous output redirect
Arg list too long
Argument too large
Arguments too long
Bad ! arg selector
Bad ! form
Bad ! modifier:
Bad : mod in $
Bad address
Bad file number
Bad substitute
Bad system call
Badly formed number
Badly placed (
Badly placed ()'s
Block device required
Broken pipe
Bus error
Can't << within ()'s
Can't exit, ignoreexit is set
Can't from terminal
Can't make pipe
Cannot determine type of shell to use
Command not found
Cross-device link
Data transfer error
Device busy
Device write protected
Divide by 0
EMT trap
End of data
End of media
Error 0
Exec format error
Exit status %s
Expansion buf ovflo
Expression syntax
File exists
File table overflow
File too large
Floating exception
I/O error
IOT trap
Illegal instruction
Illegal seek
```

```
Improper mask
Improper then
Interrupted
Interrupted system call
Invalid argument
Invalid null command
Invalid variable
Is a directory
Killed
Line overflow
Missing )
Missing ]
Missing file name
Missing name for redirect
Missing }
Mod by 0
Modifier failed
Mount device busy
New mail
No args on labels
No children
No file for $0
No home
No match
No media
No more processes
No more processes, waiting for current ones to complete.
No more words
No output
No prev lhs
No prev search
No prev sub
No space left on device
No such device
No such device or address
No such file or directory
No such process
Not a directory
Not a typewriter
Not enough core
Not in while/foreach
Not login shell
Not owner
Out of memory
Output redirection not allowed
Pathname too long
Permission denied
Quit
Read-only file system
Result too large
Rhs too long
Segmentation violation
Sig %d
Subscript error
Subscript out of range
Subst buf ovflo
```

```
Syntax error
Terminated
Text file busy
Too dangerous to alias that
Too few arguments
Too many ('s
Too many )'s
Too many arguments
Too many links
Too many open files
Too many words from ``
Trace/BPT trap
Undefined variable
Unknown error
Unknown user: %s
Unmatched
Unmatched %c
Unmatched `
Use "exit" to leave csh.
Use "logout" to logout.
Variable syntax
Word too long
Words not ()'ed
You have %smail.
end not found
endif not found
endsw not found
label not found
not-ascii shell script
source: Output redirection not allowed
then/endif not found
```

C-ISAM

Indexsequentielle Zugriffsmethode

Vorwort

C-ISAM ist eine Bibliothek mit Funktionen, die in der Programmiersprache C geschrieben sind und mit Anwenderprogrammen verbunden werden koennen. Die indexsequentielle Zugriffsmethode ist guenstig anwendbar, wenn grosse Datenmengen gleichartig strukturierter Datensaetze zu verwalten sind.

Mit C-ISAM koennen indizierte Dateien eingerichtet und verwaltet werden. Indizierte Dateien bieten die Moeglichkeit des schnellen Zugriffs auf bestimmte ausgewaehlte Datensaetze, ohne die gesamte Datei sequentiell lesen zu muessen.

Inhaltsverzeichnis

1. Ueberblick

1.1. Einleitung

C-ISAM (C-Indexed Sequential Access Method) ist eine indexsequentielle Zugriffsmethode fuer das Betriebssystem WEGA. Sie besteht aus einer Bibliothek von Funktionen, die in der Programmiersprache C geschrieben sind und die die Erzeugung und Manipulation indexsequentieller Dateien erlauben. Die C-ISAM-Bibliotheken /urs/lib/libcisam.a (nichtsegmentiert) oder /usr/slibcisam.a (segmentiert) werden vom Lader durch Angabe der Option

"-lcisam"

mit dem Anwenderprogramm verbunden. Die C-ISAM-Bibliothek kann vom Benutzer auch aus anderen Programmiersprachen aufgerufen werden, die den Zugriff auf C-Bibliotheken ermoeglichen. Dem Programmierer stehen folgende Dienste zur Verfuegung:

- Erzeugung indexsequentieller Dateien
- Definition von Primaer- und Sekundaerschluesseln
- Hinzuzufuegen oder Loeschen von Indizes
- Hinzuzufuegen oder Loeschen von Datensaetzen
- Sequentieller oder Random-Zugriff auf Datensaetze
- Verriegelung einzelner Datensaetze, Gruppen von Datensaetzen oder der gesamten Datei
- Umbenennen oder Loeschen indizierter Dateisysteme
- Komprimieren indizierter Dateien, um den Plattenzugriff zu optimieren und Speicherplatz zu sparen.

Dieses Handbuch beschreibt die Anwendung der C-ISAM-Bibliotheksfunktionen (siehe Tabelle 1) zum Aufbau indizierter Dateisysteme. Die einzelnen Funktionen werden auch im WEGA-Programmierhandbuch (im Sinne eines Nachschlagewerkes) beschrieben. Den verwendeten Namen der C-ISAM-Funktionen ist deshalb immer der Suffix "(3)" angehaengt (z.B. isopen(3)). Er verweist auf das Kapitel 3 - Bibliotheksfunktionen - des WEGA-Programmierhandbuchs. Eine Zusammenfassung der Funktionen ist im Anhang A zu finden.

Tabelle 1 Funktionelle Zusammenfassung der C-ISAM Funktion

Funktion	Beschreibung
Operationen mit ungeoeffneten Dateien:	
isbuild	erzeugt und oeffnet eine Datei
isopen	oeffnet eine existierende Datei
isrename	benennt eine Datei um
iserase	loescht eine Datei
Funktionen mit geoeffneten Dateien:	
isclose	schliesst eine geoeffnete Datei
isaddinedx	fuegt einen sekundaeren Index hinzu
isdelindex	loescht einen sekundaeren Index
isstart	setzt den aktuellen Schluessel und Datensatz zurueck
islock	Lese-Verriegelung fuer die Datei
isunlock	entriegelt die Datei
isindexinfo	Holen von Index/Verzeichnis-Informationen ueber die Datei
isuniqueid	definiert einen einmaligen Schluessel fuer die Datei
isaudit	fuehrt Audit-Trail-Funktionen aus
Funktionen mit Datensaetzen:	
isread	liest den Datensatz sequentiell oder im Random-Mode
iswrite	fuegt einen Datensatz in die Datei ein
isrewrite (isrewcurr)	schreibt einen Datensatz neu
isdelete	loescht einen Datensatz
Sonstige Funktionen:	
isperror	druckt C-ISAM-Fehler
isld	laedt einen Wert aus einer Byte-Zeichenkette
isst	speichert den Wert in eine Byte-Zeichenkette

1.2. Aufbau von C-ISAM-Dateien

Eine C-ISAM-Datei setzt sich aus zwei WEGA-Dateien zusammen: der Datendatei (gekennzeichnet durch Suffix ".dat") und den Indexdateien (Suffix ".idx").

1.2.1. Die Datendatei (.dat)

Normale WEGA-Dateien sind nicht nach einer bestimmten Struktur aufgebaut; sie werden einfach als eine Folge von Bytes betrachtet. C-ISAM hingegen gestattet es, Dateien in einer selbstgewaehlten Struktur anzulegen, um den Zugriff auf Information zu erleichtern und zu beschleunigen. Diese Struktur laesst es zu, die Datendatei als eine Sammlung von Datensaetzen und einen Datensatz als eine Sammlung von Feldern zu betrachten, wobei ein oder mehrere Felder innerhalb des Datensatzes als Primaerschluessel definiert sind. Der Primaerschluessel dient der Identifizierung des Datensatzes und als ein Index fuer die Datei. Allen C-ISAM-Datendateinamen (maximal 10 Zeichen) wird bei deren Bildung automatisch der Suffix.dat angehaengt.

Nehmen wir z.B. eine Personaldatei, die fuer jeden Angestellten einen Datensatz enthaelt. So eine Datei koennte die Personalnummer des Angestellten haben, die als Primaerschluessel definiert ist. Einer oder mehrere Sekundaerschluessel koennen fuer die Datei definiert sein, um einen alternativen Index fuer die Datei zu schaffen. Mit der Personaldatei koennte fuer die Nachnamen der Angestellten ein Sekundaerschluessel definiert werden.

Datensaetze (Records)

Ein Datensatz ist eine logische Einheit von Informationen, die sich aus einem oder mehreren Feldern zusammensetzt (z.B. Informationen ueber einen Angestellten in einer Personaldatei einer Firma).

Felder (fields)

Ein Feld ist eine logische Einheit von Informationen in einem Datensatz. So koennte z.B. ein Personaldatensatz mehrere Felder enthalten, die eine Personalnummer des Angestellten, den Namen, die Nummer der Abteilung usw. einschliessen. C-ISAM erkennt Felder mit den folgenden Datentypen an (im Anhang C detailiert beschrieben):

Zeichenketten fester Laenge (0-255 Bytes)
Integerwerte (int und long)
Gleitkommazahlen (float und double).

Ein Feld kann mit einem beliebigen Versatz innerhalb des Datensatzes beginnen.

Der Primaerschluessel

Jede C-ISAM-Datei muss einen Primaerschluessel besitzen, durch den die Datensaetze mit einem Index versehen werden und somit zugreifbar sind. Ein Schluessel kann ein bis acht Felder umfassen. Standardmaessig muss der

Primaerschluessel die Datensaetze einer Datei eindeutig identifizieren. Andernfalls erlaubt der Schluessel Duplikate.

Wird zum Beispiel der Nachname eines Angestellten als Primaerschluessel fuer die Datei definiert, kann dieser Schluessel den Datensatz nicht eindeutig indizieren, da mehr als ein Angestellter den gleichen Nachnamen haben koennte. Der Primaerschluessel koennte daher so definiert werden, dass er 3 Felder umfasst: Vornamen, Nachnamen und Einstellungsjahr des Angestellten. Eine spezielle C-ISAM-Funktion, isuniqueid(3), liefert einen eindeutigen Primaerschluessel fuer die Datei, wenn auf normalem Wege kein eindeutiger Schluessel erkennbar ist.

Der Sekundaerschluessel

Zusaetzlich zum Primaerschluessel, der fuer die Indizierung verantwortlich ist, kann eine beliebige Anzahl von Sekundaerschluesseln fuer die Datei definiert werden.

1.2.2. Die Indexdatei (.idx)

Jede C-ISAM-Datendatei besitzt eine zugehoerige Indexdatei, die beim Aufbau der C-ISAM-Datei erzeugt wird. Der Name der Indexdatei entspricht dem Namen der C-ISAM-Datei, dem der Suffix ".idx" angehaengt wurde. Die Indexdatei besitzt ein Verzeichnis, in dem die Primaer- und Sekundaerschluessel beschrieben sind. Da es keine Begrenzung der Anzahl der Schluessel gibt, die fuer die Daten definiert werden koennen, kann die Indexdatei schnell anwachsen. Dadurch wird viel Plattenspeicherplatz verbraucht und der Systemdurchsatz wird verringert. Mit Hilfe von C-ISAM-Funktionen koennen die Schluesselwerte in der Indexdatei komprimiert werden. Die Indexkomprimierung wird in Teil 3 beschrieben.

2. Dateierzeugung und Indexdefinition

2.1. Einleitung

In diesem Abschnitt wird anhand verschiedener Musterprogramme die Bildung und Manipulation von C-ISAM-Dateien beschrieben. Dabei wird auch auf die Definition von Indizes sowie auf das Hinzufuegen von Indizes und Daten eingegangen.

2.2. Erzeugung von C-ISAM-Dateien

Die C-ISAM-Funktion isbuild(3) definiert und erzeugt eine C-ISAM-Datei. Im Ergebnis dieses Funktionsaufrufes entstehen 2 WEGA-Dateien: eine Datendatei mit dem Suffix ".dat", der dem Parameter filename angehaengt wird, und eine Index-Datei mit dem Suffix ".idx", Die Datendatei filename.dat enthaelt nur Daten; die Indexdatei filename.idx enthaelt ein Verzeichnis zur Beschreibung der Indizes der Datei sowie die Indizes selbst.

2.3. Indexdefinition

Jede C-ISAM-Datei muss einen Primaerschluessel besitzen. Sekundaerschluessel koennen fuer eine Datei ebenfalls definiert werden, entweder waehrend der Bildung der Datei oder zu einem spaeteren Zeitpunkt. Die Strukturen keydesc und keypart (siehe Bild 2-1) definieren die Indizes der Datei. Diese Strukturen werden von den Funktionen isbuild(3) und isaddindex(3) verwendet.

```
struct keydesc {
     int    k_flags     /* Flags */
     int    k_nparts    /* Anzahl der Schluesselteile */
     struct keyparts    /* Teile des Schluessels */
                 k_part[NPARTS]
};

struct keypart {
     int    kp_start;   /* Startbyte des Schluesselteils */
     int    kp_length;  /* Laenge in Bytes */
     int    kp_type;    /* Typ des Schluesselteils */
};
```

Bild 2-1. Aufbau der Strukturen keydesc und keypart fuer die Indexdefinition

2.3.1. Die Struktur keydesc

In der Struktur keydesc enthaelt der Integerwert k_flags die Komprimierungsinformationen und zeigt an, ob Duplikatschluesselwerte zugelassen sind. Dieser Integerwert ist die arithmetische Summe der Werte folgender Schluesseldeskriptoren:

ISNODUPS	keine Duplikate (Standard)
ISDUPS	Duplikate
DCOMPRESS	Duplikatkomprimierung
LCOMPRESS	Komprimierung der fuehrenden Bytes
TCOMPRESS	Komprimierung der nachfolgenden Bytes
COMPRESS	vollstaendige Komprimierung

Die Indexkomprimierung wird im Abschnitt 3 beschrieben. Der Integerwert k_nparts zeigt an, aus wieviel Teilen (Feldern) der Schluessel besteht. Jeder Teil muss durch eine Struktur keypart beschrieben sein. Die Anzahl der Elemente im Feld k_part sollte mit den Integerwerten in k_npart uebereinstimmen.

2.3.2. Die Struktur keypart

Die Struktur keypart gestattet das Zusammensetzen eines Schluessels aus mehreren Feldern, die die Teile des Schluessels darstellen. Ein Schluessel kann aus bis zu acht Teilen bestehen. Diese Teile eines Index muessen innerhalb des Datensatzes nicht zusammenhaengend auftreten, sie muessen auch nicht in einer besonderen Reihenfolge in dem Datensatz existieren. Der Integerwert kp_start bezeichnet das Startbyte im Schluesselteil als Versatz vom Beginn des Datensatzes an. Der Wert kp_length enthaelt die Laenge des Schluesselteils (in Bytes). Der Datentyp des Schluesselteils wird in kp_type angegeben. Die von C-ISAM unterstuetzten Typen sind im Anhang C beschrieben.

Die folgenden Beispiele, die auf einem fiktiven Personalsystem basieren, illustrieren die Dateibildung und Indexdefinition. Das Personalsystem bestehl aus 2 C-ISAM-Dateien, der Personaldatei und der Leistungsdatei. Die Personaldatei enthaelt fuer jeden Angestellten einen Datensatz folgender Struktur:

- Nummer des Angestellten
- Name
- Adresse.

Die Leistungsdatei enthaelt Informationen, die alle Ueberpruefungen der Arbeitsleistung fuer jeden Angestellten betrifft. Es gibt einen Datensatz fuer jede Leistungsueberpruefung, fuer die Aenderung der Berufsbezeichnung oder die vollzogene Gehaltsaenderung eines Angestellten. Folglich kann es fuer jeden Personal-Datensatz in der Personaldatei viele Datensaetze in der Leistungsdatei geben. Die Felddefinitionen fuer die Datensaetze sowohl in der Personaldatei

als auch in der Leistungsdatei werden nachfolgend gezeigt.

Definition der Personaldatei:

Bezeichnung des Feldes	Stellung im Datensatz
Personalnummer	0- 3 LONGTYPE
Nachname	4-23 CHARTYPE
Vorname	24-43 CHARTYPE
Adresse	44-63 CHARTYPE
Stadt	64-83 CHARTYPE

Definition der Leistungsdatei:

Bezeichnung des Feldes	Stellung im Datensatz
Personalnummer	0- 3 LONGTYPE
Datum der Beurteilung	4- 9 CHARTYPE
Leistungsbeurteilung	10-11 CHARTYPE
Gehalt nach Beurteilung	12-19 DOUBLETYPE
Titel nach der Beurteilung	20-50 CHARTYPE

2.4. Aufbau einer C-ISAM-Datei

Bild 2-2 zeigt ein Musterprogrammm, dass mittels der Funktion isbuild(3) sowohl eine Personaldatei als auch eine Leistungsdatei erzeugt. Fuer die Personaldatei ist der Primaerschluessel als Personalnummer definiert. Fuer die Leistungsdatei besteht der Primaerschluessel aus zwei Teilen, die aus der Personalnummer und dem Datum der Beurteilung besteht.

```
#include <isam.h>

struct    keydesc key;
int  fdemploy, fdperform;

/*
 * Bilden der C-ISAM-Dateien fuer die Datendateien
 * der Personaldatei und der Leistungsdatei
 */

main()
{
    mkemplkey();
    fdemploy = isbuild("employee",84,&key,
                       ISINOUT+ISEXCLLOCK);
    if (fdemploy < 0) {
        printf("isbuild error %d for employee file\n",
               iserrno);
        exit(1);
```

```
    }
    mkperfkey();
    fdperform = isbuild("perform",49,&key,
                        ISINOUT+ISEXCLLOCK);
    if (fdperform < 0) {
        printf("isbuild error %d for performance file\n",
               iserrno);
        exit(1);
    }
    isclose(fdperform;
}

mkemplkey()
{
    key.k_flags = 0;    /* no dups, no compression */
    key.k_nparts = 1;   /* one part index */

    key.k_part[0].kp_start = 0;        /* offset is zero */
    key.k_part[0].kp_type = LONGTYPE;  /* type is long   */
    key.k_part[0].kp_leng = 4;         /* 4 bytes        */
}

mkperfkey()
{
    key.k_flags = 0;     /* no dups, no compression */
    key.k_nparts = 2;    /* one part index          */

    key.k_part[0].kp_start = 0;        /* offset is zero */
    key.k_part[0].kp_type = LONGTYPE;  /* type is long   */

    key.k_part[1].kp_start = 4;        /* offset is 4    */
    key.k_part[1].kp_type = CHARTYPE;  /* type is char   */
    key.k_part[1].kp_leng = 6;         /* 4 bytes        */
}
```

Bild 2-2 Programm zum Aufbau von Dateien und Bilden von Indizes.

2.5. Das Hinzufuegen von Sekundaerindizes

Bei einigen Anwendungen reicht der Primaerschluessel nicht aus, um die Datei vollstaendig mit Indizes zu versehen. In diesen Faellen koennen ein oder mehrere Sekundaerindizes definiert werden. Es gibt keine Begrenzung fuer die Anzahl solcher Indizes. In der Praxis muessen der Platz und die Zugriffszeit in Betracht gezogen werden. Im Falle der Personalmusterdatei sind 2 Sekundaerindizes erwuenscht, ein Index fuer den Nachnamen in der Personaldatei und ein Index fuer das Feld "Gehalt" in der Leistungsdatei. Das folgende Programm bildet diese beiden Indizes (Bild 2-3). Es ist zu beachten, dass waehrend des Hinzufuegens der Indizes die Datei mit einer exklusiven Verriegelung geoeffnet werden muss. Exklusive Dateiverriegelungen sind im Parameter

```
        mode = ISINOUT + ISEXCLLOCK
```

der Funktion isopen(3) definiert. ISINOUT spezifiziert, dass die Datei sowohl fuer Eingabe als auch fuer Ausgabe geoeffnet werden soll. Die komponente ISEXCLLOCK bewirkt, dass die Datei ausschliesslich fuer den aktuellen Prozess verriegelt werden soll und keinem anderen Prozess ein Zugriff (ach nicht zum Lesen) auf diese Datei gestattet wird. Ausserdem ist zu beachten, dass Duplikate fuer beide Sekundaerindizes erlaubt werden sollen, und dass die Bezeichnung des Feldes die vollstaendige Komprimierung fuer dessen Werte haben soll, die in der Indexdatei gespeichert sind.

```
#include <isam.h>
#define SUCCESS 0

struct    keydesc key;
int  cstart, nparts;
int  fdemploy, fdperform;

/*
 * Das folgende Programm fuegt Sekundarindizes fuer
 * das fuer den Nachnamen vorgesehene Feld
 * in der Personaldatei und das Feld fuer das Gehalt
 * in der Leistungsdatei hinzu.
 */

main()
{
    int    cc;
    fdemploy = cc = isopen("employee", ISINOUT+ISEXCLLOCK);
    if (cc < SUCCESS) {
        printf("isopen error %d for employee file\n",
                iserrno);
        exit(1);
    }
    mklnamekey();
    cc = isaddindex(fdemploy, &key);
    if (cc != SUCCESS) {
        printf("isaddindex error %d for employee lname
                key\n", iserrno);
        exit(1);
    }
    isclose(fdemploy);

    fdperform = cc = isopen("perform", ISINOUT+ISEXCLLOCK);
    if (cc < SUCCESS) {
        printf("isopen error %d for perform file\n",
                iserrno);
        exit(1);
    }
    mksalkey();
```

```
    cc = isaddindex(fdperform, &key);
    if (cc != SUCCESS) {
        printf("isaddindex error %d for perform\n",
                iserrno);
        isclose(fdperform);
        exit(1);
    }
    isclose(fdperform);
}

mklnamekey()
{
    key.k_flags = ISDUPS + COMPRESS;
    key.k_nparts = 0;
    cstart = 4;
    nparts = 0;
    addpart(&key, 20, CHARTYPE);
}

mksalkey()
{
    key.k_flags = ISDUPS;
    key.k_nparts = 0;
    cstart = 12;
    nparts = 0;
    addpart(&key, sizeof(double), DOUBLETYPE);
}

addpart(keyp, len, type)
register struct keydesc *keyp;
int len;
int type;
{
    keyp->k_part[nparts].kp_start = cstart;
    keyp->k_part[nparts].kp_leng = len;
    keyp->k_part[nparts].kp_type = type;
    keyp->k_nparts = ++nparts;
    cstart += len;
}
```

Bild 2-3 Programm zur Hinzufuegung vom Sekundaerindizes.

2.6. Das Hinzufuegen von Daten

Wird eine Datei mit der Funktion isopen(3) geoeffnet, muss die Operation und der gewuenschte Verriegelungstyp (mode) spezifiziert sein. Fuer den Parameter mode sind folgende Angaben zulaessig:

ISINPUT	Anforderung zum Lesen
ISOUTPUT	Anforderung zum Schreiben
ISINOUT	Anforderung fuer Lesen und Schreiben.

2.6.1. Das Lesen und Lokalisieren des Datensatzes

Der Zugriff auf Datensaetze wird ueber die Funktionen isread(3) oder isstart(3) realisiert. Die Funktion isread(3) liest einen Datensatz in den Puffer ein, waehrend isstart(3) den Datensatz nur lokalisiert, jedoch nicht zurueckgibt. Beide Funktionen verwenden ein mode-Parameter, dessen Werte in Tabelle 2-1 definiert sind.

Tabelle 2-1 Mode-Parameter fuer die Funktionen isread(3) und isstart(3)

mode	Beschreibung
ISFIRST	Lokalisiert den ersten Datensatz
ISLAST	Lokalisiert den letzten Datensatz
ISNEXT	Lokalisiert den folgenden Datensatz
ISPREV	Lokalisiert den vorangegangenen Datensatz
ISCURR	Lokalisiert den aktuellen Datensatz
ISSEQUAL	Lokalisiert den Datensatz mit dem Schluesselwert, der mit dem spezifizierten Wert uebereinstimmt
ISGREAT	Lokalisiert den Datensatz mit dem Schluesselwert, der groesser ist als der spezifizierte Wert
ISGTEQ	Lokalisiert den Datensatz mit dem Schluesselwert, der groesser als oder gleich dem spezifizierten Schluesselwert ist

Wenn ISEQUAL, ISGREAT oder ISGTEQ spezifiziert ist, sucht die Funktion nach einem Datensatz, der mit dem vom Benutzer spezifizierten Wert uebereinstimmt. Bei der Funktion isread(3) muss es der aktuelle Schluessel sein. Im Falle isstart(3) kann jeder Schluessel im Schluessel-Diskriptor-Parameter spezifiziert sein. Der Benutzer ist fuer das Plazieren des Suchwertes im Datensatzpuffer an der Stelle, wo der Wert in dem Datensatz lokalisiert wird, verantwortlich.

Beispiel:
Der Primaerschluessel ist eine Zeichenkette der Laenge von 3 Byte, die mit einem Versatz von 2 Byte (vom Anfang des Datensatzes aus gerechnet) beginnt. Der erste Datensatz, auf den zugegriffen werden soll, hat als Primaerschluessel den Wert "ABC". In diesem Fall muss die Zeichenkette "ABC" mit dem Versatz 2 im Datensatzpuffer stehen.

Mit der Funktion isstart(3) kann eine partielle Suche nach Schluesseln realisiert werden. Um z.B. den ersten Datensatz, der mit "A" beginnt, zurueckzurufen, ist die Zeichenkette "A" mit dem Versatz von 2 Byte und der Laenge 1 in den Datensatzpuffer zu schreiben. Damit wuerde der Datensatz "AAA" vor "ABC" zurueckgegeben werden.

Wird die Funktion isread(3) verwendet und ist manuelles

Sperren bei der Dateieroeffnung spezifiziert, kann der Datensatz durch das Hinzufuegen des Wertes ISLOCK zum Parameter mode verriegelt werden (siehe Abschnitt 3).

2.6.2. Die Aktualisierung der Datei

Das Einfuegen eines Datensatzes in eine Datendatei wird mit der Funktion iswrite(3) erfuellt. Wird der Datensatz eingefuegt, werden die Indizes fuer jeden Schluessel (Primaer- und Sekundaerschluessel) aktualisiert. Eine Fehlernachricht wird ausgegeben, sollte der Versuch unternommen werden, einen Datensatz mit einem Duplikatschluesselwert einzusetzen, wenn die Datei keine Duplikatwerte zulaesst.

Wenn ein Datensatz neu geschrieben wird (Funktionen isrewrite(3) oder isrewcurr(3)), wird der existierende Datensatz durch einen neuen ersetzt. Der Wert des Primaerschluessels kann waehrend dieser Operation nicht geaendert werden. Es existieren 2 Formen der Kommandos zur Aktualisierung und Loeschung von Datensaetzen. Besitzt die Datei einen eindeutigen Primaerschluessel, ist die Funktion isrewrite(3) oder isdelete(3) zu benutzen, um einen Datensatz hinzuzufuegen oder zu loeschen. Sollte die Datei keinen eindeutigen Primaerschluessel besitzen, muss der Datensatz unter Nutzung der Funktion isread(3) oder isstart(3) lokalisiert und anschliessend mittels der Funktion isrewcurr(3) oder isdelcurr(3) aktualisiert werden.

Bild 2-4 zeigt ein Musterprogramm, das der Personaldatei Datensaetze hinzufuegt. Die Werte fuer die Felder dieser Datensaetze werden von der Standardeingabe abgefordert. Zu beachten ist dabei, dass die Personaldatei durch das Flag ISOUTPUT als mode-Parameter geoeffnet wird.

```
#include <isam.h>
#include <stdio.h>

#define      WHOKEKEY      0
#define      SUCCESS       0
#define      TRUE          1
#define      FALSE         0

char emprec[85];
char perfrec[51];
char line[82];
long empnum;

struct    keydesc key;
int  cstart, nparts;
int  fdemploy, fdperform;
int  finished = FALSE
```

```
/*
 * Dieses Programm fuegt der Personaldatei einen neuen
 * Personaldatensatz hinzu. Es fuegt ebenfalls den
 * ersten Leistungsdatensatz des Angestellten der
 * Leistungsdatei hinzu.
 */

main()
{
    int    cc;
    fdemploy = cc = isopen("employee",ISOUTPUT+ISMANULOCK);
    if (cc < SUCCESS) {
        printf("isopen error %d for employee file\n",
               iserrno);
        exit(1);
    }
    mklnamekey();
    fdperform = cc = isopen("perform",ISOUTPUT+ISMANULOCK);
    if (cc < SUCCESS) {
        printf("isopen error %d for perform file\n",
               iserrno);
        exit(1);
    }
    getemployee();
    while(!finished) {
        addemployee();
        getemployee();
    }
    isclose(fdemploy);
    isclose(fdperform);
}

getperform()
{
    double    new_salary;

    if (empnum == 0) {
        finished = TRUE;
        return(0);
    }
    stlong(empnum, perfrec);
    printf("Start Date: ");
    getline(line, 80);
    stchar(line, perfrec+4, 6);
    stchar("g", perfrec+10, 1);
    printf("Starting salary: ");
    getline(line, 80);
    sscanf(line, "%lf", &new_salary);
    stdbl(new_salary, perfrec+11);
    printf("Title: ");
    getline(line, 80);
    stchar(line, perfrec+19, 30);
    printf("\n\n\n");
}
```

```
addemployee()
{
    int    cc;

    cc = iswrite(fdemploy, emprec);
    if (cc != SUCCESS) {
        printf("iswrite error %d for employee\n", iserror);
        isclose(fdemploy);
        exit(1);
    }
}

addperform()
{
    int    cc;

    cc = iswrite(fdperform, perfrec);
    if (cc != SUCCESS) {
        printf("iswrite error %d for performance\n",
               iserror);
        isclose(fdperform);
        exit(1);
    }
}

putnc(c, n)
char *c;
int    n;
{
    while(n--)
        putchar(*(c++));
}

getemployee()
{
    printf("Employee number (enter 0 to exit): ");
    getline(line, 80);
    sscanf(line, "%ld", &empnum);
    if (empnum == 0) {
        finished = TRUE;
        return(0);
    }
    stlong(empnum, emprec);
    printf("Last name: ");
    getline(line, 80);
    stchar(line, emprec+4, 20);

    printf("First name: ");
    getline(line, 80);
    stchar(line, emprec+24, 20);

    printf("Address: ");
    getline(line, 80);
    stchar(line, emprec+44, 20);
```

```
    printf("City: ");
    getline(line, 80);
    stchar(line, emprec+64, 20);

    getperform();
    addperform();
    printf("\n\n\n");
}

getline(s, lim)
char    s[];
int    lim;
{
    int    c, i;

    for(i+0; i<lim && (c=getchar())!=EOF && c!='\n'; ++i)
        s[i] = c;
    if (c == '\n') {
        s[i] = c;
        ++i;
    }
    s[i] = '\0';
    return(i);
}

stchar(a, b, c)
char *a, *b;
int c;
{
    register int i;
    for(i+0; *a && (i<c); i++)
        *b++ = *a++;
    return(0);
}
```

Bild 2-4 Programm zum Hinzufuegen von Daten

2.7. Sequentieller Zugriff

Das Programm in Bild 2-5 demonstriert den sequentiellen Zugriff auf eine Datei. In diesem Falle wird die Personaldatei auf Befehl der Primaerschluessel "Personalnummer" gelesen. Da der Personalnummerindex als steigend definiert ist und keine Duplikatschluesselwerte erlaubt sind, wird die Datensatzsequenz vom niedrigsten zum hoechsten Wert der Personalnummer drucken lassen. Dieser Vorgang wird fortgesetzt, bis die Funktion isread(3) (unter Angabe von ISNEXT) den Wert -1 zurueckgibt, wobei im Feld iserrno der Wert EENDFILE abgelegt wird (Dateiende).

```
#include <isam.h>

#define      WHOKEKEY      0
```

```
#define      SUCCESS       0
#define      TRUE          1
#define      FALSE         0

char emprec[83];

struct    keydesc key;
int  cstart, nparts;
int  fdemploy, fdperform;
int  eof = FALSE

/*
 * Dieses Programm liest die Personaldatei sequentiell
 * unter Benutzung der Personalnummer und gibt jeden
 * Datensatz immer dann zur Standardausgabe, wenn er
 * gefunden wird.
 */

main()
{
    int    cc;
    fdemploy = cc = isopen("employee", ISINPUT+ISAUTOLOCK);
    if (cc < SUCCESS) {
        printf("isopen error %d for employee file\n",
               iserrno);
        exit(1);
    }
    mkemplkey();
    cc=isstart(fdemploy, &key, WHOLEKEY, emprec, ISFIRST);
    if (cc != SUCCESS) {
        printf("isstart error %d\n", iserrno);
        isclose(fdemploy);
        exit(1);
    }
    getfirst();
    while(!eof) {
        showemployee();
        getnext();
    }
    isclose(fdemploy);
}

showemployee()
{
    printf("Employee number: %ld", ldlong(emprec));
    printf("\nLast name: ");
    putnc(emprec+4, 20);
    printf("\nFirst name: ");
    putnc(emprec+24, 20);
    printf("\nAddress: ");
    putnc(emprec+44, 20);
    printf("\nCity: ");
    putnc(emprec+64, 20);
}
```

```
putnc(c, n)
char *c;
int    n;
{
    while(n--)
        putchar(*(c++));
}

getfirst()
{
    int    cc;
    if (cc = isread(fdemploy, emprec, ISFIRST)) {
        switch(iserrno) {
        case EENDFILE:
            eof = TRUE;
            break;
        default:
            printf("isread ISFIRST error %d\n", iserrno);
            eof = TRUE;
            return(1);
        }
    }
    return(0);
}

getnext()
{
    int    cc;
    if (cc = isread(fdemploy, emprec, ISNEXT)) {
        switch(iserrno) {
        case EENDFILE:
            eof = TRUE;
            break;
        default:
            printf("isread ISNEXT error %d\n", iserrno);
            eof = TRUE;
            return(1);
        }
    }
    return(0);
}

mkemplkey()
{
    key.k_flags = 0;
    key.k_nparts = 1;

    key.k_part[0].kp_start = 0;
    key.k_part[0].kp_type = LONGTYPE;
    key.k_part[0].kp_leng = 4;
}
```

Bild 2-5 Programm zum sequentiellen Zugriff auf eine Datei

2.8. Random-Zugriff

Das Programmbeispiel in Bild 2-6 verdeutlicht, wie der Random-Zugriff zu einer C-ISAM-Datei realisiert werden kann. Dieses Programm fordert ueber die Standardeingabe eine Personalnummer an, sucht den zu dieser Nummer korrespondierenden Datensatz in der Personaldatei und gibt die Ergebnisse auf die Standardausgabe. Das Makro ISEQUAL wird verwendet, um den Lesemode fuer die Funktion isread(3) in der C-Bibliotheksfunktion "reademp" zu spezifizieren. Wird kein Datensatz gefunden, der der vom Benutzer eingegebenen Personalnummer entspricht, wird der Fehlerkode ENOREC in iserrno zurueckgegeben, und isread(3) gibt den Wert -1 zurueck. Der C-Programmierer ist fuer die Handhabung des Rueckgabewertes verantwortlich. Entspricht der Rueckgabewert ENOREC, wird der Datensatzpuffer, der als Datensatzsparameter der Funktion isread(3) uebergeben wird, nicht geaendert (d.h, es wird kein Datensatz gelesen).

```
#include <isam.h>
#include <stdio.h>

#define      WHOKEKEY      0
#define      SUCCESS       0
#define      TRUE          1
#define      FALSE         0

char emprec[83];
long empnum;

struct    keydesc key;
int  cstart, nparts;
int  fdemploy, fdperform;
int  eof = FALSE

/*
 * Dieses Programm fordert interaktiv von der
 * Standardeingabe eine Personalnummer an, sucht
 * in der Personaldatei den korrespondierenden
 * Datensatz und gibt diesen auf der Standardausgabe
 * aus.
 */

main()
{
    int    cc;
    fdemploy = cc = isopen("employee",ISOUTPUT+ISAUTOLOCK);
    if (cc < SUCCESS) {
        printf("isopen error %d for employee file\n",
               iserrno);
        exit(1);
    }
    mkemplkey();
    getempnum();
```

```
    while(empnum != 0) {
        if(reademp() == SUCCESS)
            showemployee();
        getempnum();
    }
    isclose(fdemploy);
}

getempnum()
{
    char *line;

    printf("Enter employee number (zero to quit): ");
    getline(line, 80);
    sscanf(line, "%lf", &empnum);
    stlong(empnum, emprec);
}

showemployee()
{
    printf("Employee number: %ld", ldlong(emprec));
    printf("\nLast name: ");
    putnc(emprec+4, 20);
    printf("\nFirst name: ");
    putnc(emprec+24, 20);
    printf("\nAddress: ");
    putnc(emprec+44, 20);
    printf("\nCity: ");
    putnc(emprec+64, 20);
    printf("\n\n\n");
}

putnc(c, n)
char *c;
int    n;
{
    while(n--)
        putchar(*(c++));
}

reademp()
{
    int    cc;
    cc = isread(fdemploy, emprec, ISEQUAL);
    if (cc != SUCCESS) {
        switch(iserrno) {
        case EENDFILE:
            eof = TRUE;
            break;
        default:
            printf("isread ISEQUAL error %d\n", iserrno);
            eof = TRUE;
            return(1);
        }
```

```
    }
    return(0);
}

mkemplkey()
{
    key.k_flags = 0;
    key.k_nparts = 1;

    key.k_part[0].kp_start = 0;
    key.k_part[0].kp_type = LONGTYPE;
    key.k_part[0].kp_leng = 4;
}

getline(s, lim)
char    s[];
int    lim;
{
    int    c, i;

    for(i+0; i<lim && (c=getchar())!=EOF && c!='\n'; ++i)
        s[i] = c;
    if (c == '\n') {
        s[i] = c;
        ++i;
    }
    s[i] = '\0';
    return(i);
}
```

Bild 2-6 Programm fuer Random-Zugriff auf C-ISAM-Dateien

2.9. Verkettung

Das folgende Beispiel zeigt, wie ein Datensatz mit bereits verknuepfeten Datensaetzen verkettet wird. Im folgenden wird veranschaulicht, wie der Leistungsdatensatz durch den Primaerschluessel lokalisiert werden kann. Der Primaerindex ist aus der Personalnummer und dem Ueberpruefungsdatum zusammengesetzt.

Die folgende Programmfunktion soll der Leistungsdatei interaktiv einen neuen Datensatz hinzufuegen. Der Datensatz enthaelt das Datum der stattgefundenen Gehaltsueberpruefung, die gegenwaertige Leistungsbeurteilung des Angestellten, das auf der letzten Beurteilung basierende Gehalt des Angestellten und den neuen oder derzeitigen Titel des Angestellten. Alle Felder (ausser das neue Gehalt) werden vom Benutzer eingegeben. Das neue Gehalt wird durch Multiplikation des zuletzt gewaehrten Gehalts, das am Ende der "Kette" der verknuepften Datensaetzen der Leistungsgeschichte jenes Angestellten zu finden ist, mit einem von der Leistungsbeurteilung des Angestellten

abhaengigen Faktor berechnet. Um den juengsten Datensatz ueber die Leistungsgeschichte des gegebenen Angestellten zu finden, wird der Datensatzpointer zu dem Datensatz unmittelbar nach dem juengstmoeglichen Ueberpruefungsdatum jenes Angestellten positioniert. In dem Beispiel ist jedes moegliche Datum kleiner als 999999. Um den juengsten Datensatz der Leistungsgeschichte jenes Angestellten aufzufinden, wird die Funktion isread(3) mit der Option ISPREV als mode-Parameter ausgefuehrt. Dieses Verfahren geht betraechtlich schneller als das Aufsuchen des ersten Datensatzes der Leistungsgeschichte eines speziellen Angestellten und das dann folgende Aufsuchen nachfolgender Datensaetze (isread(3) mit ISNEXT) durch die ganze Kette.

```
#include <isam.h>
#include <stdio.h>

#define      WHOKEKEY      0
#define      SUCCESS       0
#define      TRUE          1
#define      FALSE         0

char perfrec[51];
char operfrec[51];
char line[81];
long empnum;
double new_salary, old_salary;

struct    keydesc key;
int  cstart, nparts;
int  fdemploy, fdperform;
int  finished = FALSE

/*
 * Dieses Programm liest interaktiv Daten von der Standard-
 * eingabe und fuegt der Leistungsdatei Leistungsdatensaet-
 * ze hinzu. Abhaengig von der Leistungsbeurteilung des
 * Angestellten werden die folgenden Gehaltserhoehungen
 * im Gehalt-Feld der Leistungsdatei plaziert.
 *
 *   Beurteilung           Prozentuale Steigerung
 *   ----------------------------------------
 *   schlecht (poor)             0.0 %
 *   normal (fair)               7.5 %
 *   gut (good)                 15.0 %
 */

main()
{
    int    cc;
    fdperform = cc = isopen("perform", ISINOUT+ISAUTOLOCK);
    if (cc < SUCCESS) {
        printf("isopen error %d for performance file\n",
               iserrno);
```

```
        exit(1);
    }
    mkperfkey();
    getperformance();
    while(!finished) {
        if(get_old_salary())
            finished = TRUE;
        else {
            addperformance();
            getperformance();
        }
    }
    isclose(fdperform);
}

addperformance()
{
    int    cc;

    cc = iswrite(fdperform, perfrec);
    if (cc != SUCCESS {
        printf("iswrite error %d\n", iserrno);
        isclose(fdperform);
        exit(1);
    }
}

gerperformance()
{
    printf("Employee number (enter 0 to exit): ");
    getline(line, 80);
    sscanf(line, "%ld", &empnum);
    if (empnum == 0) {
        finished = TRUE;
        retunr(0);
    }
    stlong(empnum, perfrec);

    printf("Review date: ");
    getline(line, 80);
    stchar(line, perfrec+4, 6);

    printf("Job rating (p=poor, f=fair, g=good): ");
    getline(line, 80);
    stchar(line, perfrec+10, 1);

    printf("Salary After Review: ");
    new_salary = 0.0;
    stdbl(new_salary, perfrec+11);

    printf("Title After Review: ");
    getline(line, 80);
    stchar(line, perfrec+19, 30);
```

```
    printf("\n\n\n");
}

get_old_salary()
{
    int mode, cc;

    stchar(perfrec, operfrec, 4);     /* get id number */
    stchar("999999", operfrec+4, 6);  /* largest data */

    cc = isstart(fdperform, key, WHOLEKEY,
                 operfrec, ISGTEQ);
    if (cc != SUCCESS) {
        switch(iserrno) {
        case ENOREC:
        case EENDFILE:
            mode = ISLAST;
            break;
        default:
            printf("isstart error %d", iserrno);
            printf("in get_old_salary\n");
            return(1);
        }
    } else {
        mode = ISPREV;
    }
    cc = isread(fdperform, operfrec, mode);
    if (cc != SUCCESS) {
        printf("isread error %d in get_old_salary",
            iserrno);
        return(1);
    }
    if (cmpnbytes(perfrec, operfrec, 4)) {
        printf("No performance record for employee
            number %ld.\n", iserrno);
        return(1);
    } else {
        printf("\nPerformance record found.\n\n");
        old_salary = new_salary = lddbl(operfrec+11);
        printf("Rating: ");
        switch(*(perfrec+10)) {
        case 'p':
            printf("poor\n");
            break;
        case 'f':
            printf("fair\n");
            new_salary *= 1.075;
            break;
        case 'g':
            printf("good\n");
            new_salary *= 1.15;
            break;
        }
        stdbl(new_salary, perfrec+11);
```

```
        printf("Old salary was %f\n", old_salary);
        printf("New salary is %f\n", new_salary);
    }
}

getline(s, lim)
char    s[];
int    lim;
{
    int    c, i;

    for(i+0; i<lim-1 && (c=getchar())!=EOF && c!='\n';++i)
        s[i] = c;
    if (c == '\n') {
        s[i] = c;
        ++i;
    }
    s[i] = '\0';
    return(i);
}

mkperfkey()
{
    key.k_flags = 0;
    key.k_nparts = 2;

    key.k_part[0].kp_start = 0;
    key.k_part[0].kp_type = LONGTYPE;

    key.k_part[1].kp_start = 4;
    key.k_part[1].kp_type = CHARTYPE;
    key.k_part[1].kp_leng = 6;
}

stchar(a, b, c)
char * a, *b;
int c;
{
    register int i;

    for (i=0; *a && (i<c); i++)
        *b++ = *a++;
    return(0);
}
```

```
cmpnbytes(a, b, c)
char *a, *b;
int c;
{
    register int i;

    for (i=0; i<c; i++)
        if (*a++ != *b++)
            return(1)
    return(0);
}
```

Bild 2-7 Interaktives Hinzufuegung von Datensaetzen

3. Indexkomprimierung und Indexkontrolle

3.1. Einleitung

Gegenstaende dieses Kapitels sind:

Indexkomprimierung	Die Faehigkeit, Schluesselwerte zu einer Indexdatei zu komprimieren, um Platz zu sparen und den Durchsatz zu erhoehen
Indexkontrolle	die Faehigkeit, die Indexdatei zu ueberpruefen und zu reparieren.

3.2. Indexkomprimierung

C-ISAM kann Schluesselwerte komprimieren, die in den Indexdateien enthalten sind. Dabei werden 3 Typen von Komprimierungen verwendet:

- Komprimierung der fuehrenden Zeichen (LCCOMPRESS),
- Komprimierung des letzten Zeichen (TCOMPRESS),
- Duplikatkomprimierung (DCOMPRESS).

Ausser der Einsparung von Plattenspeicherplatz kann die Schluesselkomprimierung die reale Antwortzeit bei Random-Zugriffen verbessern, indem die Anzahl der Schluesselwerte erhoeht wird, die in einer Seite der Indexdatei enthalten sind. Mit mehr Schluesselwerten pro Indexdateiseite sind weniger Plattenzugriffe notwendig, um einen gegebenen Datensatz zu finden. Da Plattenzugriffe den ueberwiegenden Prozentsatz der realen Zeit waehrend des Dateizugriffs ausmachen, kann die Schluesselkomprimierung zu Indexdateien die reale Antwortzeit verbessern. Diese Verbesserung wird besonders deutlich, je groesser das Feldformat ist und je mehr Duplikatwerte (fuehrende duplikate Zeichen und angehaengte Leerzeichen) einen grossen Prozentsatz der Zeichen der Schluessel ausmachen werden. Allein durch die Komprimierung fuehrender Zeichen koennen 5% der Groesse einer Indexseite eingespart werden. Wesentlich deutlichere Einsparungen koennen erzielt werden, wenn angehaengte Leerzeichen ebenfalls komprimiert werden. Bei einer Feldlaenge von 20 Zeichen betraegt die Einsparung der Seitengroesse durch Komprimierung von fuehrenden und angehaengten Zeichen 67,5 %.

Die Komprimierung besitzt den Nachteil, dass jedem Index in Abhaengigkeit von der Komprimierungsart eine bestimmte Anzahl von Bytes hinzugefuegt wird:

1 Byte fuer LCOMPRESS und TCOMPRESS

2 Bytes fuer DCOMPRESS

4 Bytes fuer die Verbindung aller drei Komprimierung-sarten.

In den Bildern 3-1 bis 3-3 sind die Komprimierungsarten LCOMPRESS, die Verbindung von LCOMPRESS und TCOMPRESS sowie die Verbindung der Komprimierungsarten LCOMPRESS, TCOMPRESS und DCOMPRESS dargestellt.

Schluesselwert	komprimiert mit LCOMPRESS	Byteeinsparung
Abt.....	#Abt.....	-1
Albert.....	#lbert.....	0
Altman.....	#tman.....	1
Altmann.....	#n.....	5
Hugo.....	#Hugo.....	-1
100 Bytes	96 Bytes	4 Bytes

Mit dieser Methode werden bei den vorgegebenen Schluesselwerten 4% des sonst benoetigten Speicherplatzes eingespart. Fuer jedes Feld wird jedoch ein Byte zusaetz-lich benoetigt.

Bild 3-1 Komprimierung fuehrender Zeichen

Schluesselwert	komprimiert mit TCOMPRESS	Byteeinsparung
Abt.....	#Abt	16
Albert.....	#Albert	13
Becker.....	#Becker	13
Hugo.....	#Hugo	15
80 Bytes	23 Bytes	57 Bytes

Mit dieser Methode werden bei den vorgegebenen Schluesselwerten 71% des sonst benoetigten Speicherplatzes eingespart. Fuer jedes Feld wird jedoch ein Byte zusaetz-lich benoetigt.

Bild 3-2 Komprimierung der letzten Zeichen

Die 3. Komprimierungsmethode ist die Duplikatkomprimierung (DCOMPRESS). Wenn Duplikat-Eintraege erlaubt sind, kann DCOMPRESS zu deren Eliminierung verwendet werden. Felder, die die Werte Stadt oder Staat enthalten, sind oft

Duplikat-intensiv. Darstellung 3-3 veranschaulicht die Duplikatkomprimierung, kombiniert mit der Komprimierung fuehrender und angehaengter Zeichen (COMPRESS).

COMPRESS = LCOMPRESS + TCOMPRESS + DCOMPRESS

Schluesselwert	komprimiert mit COMPRESS	Byteeinsparung
Abt.....	##Abt	15
Albert.....	##lbert	13
Albert.....	(kein Eintrag)	20
Becker.....	##Becker	12
Hugo.....	##Hugo	14
Hugo.....	(kein Eintrag)	20
120 Bytes	26 Bytes	94 Bytes

Mit dieser Methode werden bei den vorgegebenen Schluesselwerten 78% des sonst benoetigten Speicherplatzes eingespart. Fuer jedes benutzte Feld werden jedoch 2 Bytes zusaetzlich benoetigt.

Bild 3-3 Verbindung aller 3 Komprimierungsarten

3.3. Indexkontrolle

Das Programm bcheck ueberprueft und repariert Indexdateien. Dabei wird die Konsistenz der Dateien ueberprueft, die den Suffix .dat oder .idx besitzen. Die Optionen und die Syntax des Programms bcheck sind im folgenden aufgelistet. Wird in einer C-ISAM-Datei eine Inkonsistenz vermutet, sollte die entsprechende Datei mit dem Programm bcheck bearbeitet werden. Sofern die -n oder-y Option nicht verwendet wird, arbeitet bcheck interaktiv und wartet auf die Reaktion des Benutzers auf jeden gefundenen Fehler. Die -y Option sollte mit Vorsicht verwendet werden, da automatisch auf alle Fragen des Programms mit "ja" geantwortet wird. Bcheck sollte vor allem nicht unter Nutzung der Option -y abgearbeitet werden, wenn die Dateien erstmalig ueberprueft werden.

Anwendung: bcheck -ilny cisamfiles

-i kontrolliert die Indexdatei nur
-l listet Eingaenge in b-trees auf
-n antwortet "nein" auf alle Fragen
-y antwortet "ja" auf alle Fragen

Das folgende Beispiel zeigt einen fehlerfreien Lauf des Programms bcheck. Es ist zu beachten, dass fuer jeden Index eine Gruppe von Zahlen ausgegeben wird (bis zu 8

Zahlengruppen fuer jeden Index). Diese Zahlen zeigen die Position der Schluessel in jedem Datensatz an.

Programmaufruf:

```
% bcheck sale.pros

BCHECK  C-ISAM B-tree Checker version x.x

C-ISAM FILE: sale.pros.idx

** Check Dictionary
** Check Data File Records
** Check Indexes and Key Descriptions
**    Index 1 = unique key (0,4,2)
**    Index 2 = unique key (10,2,1)
**    Index 3 = unique key (62,35,0)
**    Index 4 = duplicates (37,25,0)
**    Index 5 = duplicates (264,20,0)
** Check  Data Record and Index Node Free Lists
479 index node(s) used, 0 free -- 2638 data record(s) used,
                                                    0 free
```

Das folgende Beispiel erkennt Fehler in der C-ISAM-Datei. Die Option -n wurde verwendet, um alle Fragen mit nein zu beantworten.

Programmaufruf:

```
% bcheck -n sale.ship.idx

BCHECK  C-ISAM B-tree Checker version x.x

C-ISAM FILE: sale.ship.idx

** Check Dictionary
** Check Data File Records
** Check Indexes and Key Descriptions
**    Index 1 = unique key (0,4,2)

ERROR: 12 bas data record(s)  Delete index ? no

**    Index 2 = unique key (4,2,1)

ERROR: 12 bas data record(s)  Delete index ? no

**    Index 3 = unique key (6,6,0)

ERROR: 12 bas data record(s)  Delete index ? no

** Check  Data Record and Index Node Free Lists

ERROR: 12 missing data record(s)  Fix data record
                                  free list ? no

5 index node(s) used, 0 free -- 0 data record(s) used,
                                      12 free
```

In diesem Falle muessen die Indizes geloescht und wieder aufgebaut werden. Um diese Indizes zu korrigieren, wuerde die -y Option verwendet werden, um alle Fragen, die von bcheck gestellt wurden, mit "ja" zu beantworten.

Programmaufruf:

```
% bcheck -y sale.ship.idx

BCHECK  C-ISAM B-tree Checker version x.x

C-ISAM FILE: sale.ship.idx

** Check Dictionary
** Check Data File Records
** Check Indexes and Key Descriptions
**    Index 1 = unique key (0,4,2)

ERROR: 12 bas data record(s)  Delete index ? yes

Remake index ? yes

**    Index 2 = unique key (4,2,1)

ERROR: 12 bas data record(s)  Delete index ? yes

Remake index ? yes

**    Index 4 = unique key (6,6,0)

ERROR: 12 bas data record(s)  Delete index ? yes

Remake index ? yes

** Check  Data Record and Index Node Free Lists

ERROR: 12 missing data record(s)  Fix data record
                                  free list ? yes

** Recreate Data Record Free List
** Recreate Index 3
** Recreate Index 2
** Recreate Index 1

5 index node(s) used, 0 free -- 0 data record(s) used,
                                        12 free
```

4. Die Datei- und Datensatzverriegelung

4.1. Einleitung

C-ISAM stellt zwei Ebenen der Verriegelung zur Verfuegung:

- Verriegelungen auf Dateiebene
- Verriegelungen auf Datensatzebene.

Innerhalb dieser beiden Ebenen bietet C-ISAM verschiedene Verriegelungsmethoden an.

4.2. Die Dateiverriegelung

Die Verriegelung von Dateien kann auf 2 Wegen erreicht werden:

- Exklusive Dateiverriegelung
- Manuelle Dateiverriegelung.

4.2.1. Exklusive Dateiverriegelung

Die exklusive Dateiverriegelung hindert andere Prozesse am Lesen oder Schreiben fuer die angegebene C-ISAM-Datei. Diese Verriegelung behaelt ihre Wirkung von dem Moment an, da die Datei mit Hilfe der Funktionen isopen(3) or isbuild(3) geoeffnet wird, bis zu dem Moment, da die Datei durch die Funktion isclose(3) geschlossen wird. Exklusive Dateiverriegelung wird durch das Hinzufuegen des Wertes ISEXCLLOCK zum Parameter mode des Funktionsaufrufes isopen(3) oder isbuild(3) spezifiziert. Exklusive Dateiverriegelung ist in den meisten Situationen nicht notwendig, muss jedoch verwendet werden, wenn ein Index durch die Funktion isaddindex(3) hinzugefuegt oder ein Index durch die Funktion isdelindex(3) geloescht werden soll. Das unten gezeigte Programmskelett veranschaulicht die Ausfuehrung einer exklusiven Dateiverriegelung.

```
myfd = isopen("myfile", ISEXECLLOCK+ISINOUT);

 ...

isclose(myfd);
```

4.2.2. Manuelle Dateiverriegelung

Die Methode der manuellen Dateiverriegelung stellt eine geteilte Verriegelung dar. Sie hindert andere Prozesse am Schreiben in eine gegebene C-ISAM Datei, erlaubt jedoch anderen Prozessen, die gesperrte C-ISAM Datei zu lesen. Die

geteilte Dateiverriegelung wird mit den Funktionsaufrufen islock(3) und isunlock(3) (mode ISINPUT) spezifiziert. Wenn eine C-ISAM Datei in dieser Weise gesperrt werden soll, muss der Parameter ISMANULOCK zum Parameter mode des Aufrufs isopen(3) oder isbuild(3) hinzugefuegt werden. Wird im Programm das Verriegeln gewuenscht, ist die Funktion islock(3) aufzurufen (eigentliche Verriegelung). Der Funktionsaufruf isunlock(3) hebt die Verriegelung wieder auf.

```
        myfd = isopen("myfile", ISMANULOCK+ISINOUT);

                /* "myfile" ist nicht verriegelt */

        islock(myfd);

                /* "myfile" ist verriegelt */

        isunlock(myfd);

                /* "myfile" ist nicht verriegelt */

        isclose(myfd);


        isclose(myfd);
```

4.3. Datensatzverriegelung

Es gibt 2 Gruppen von Datensatzverriegelungen:

- automatische Datensatzverriegelung
- manuelle Datensatzverriegelung.

Die automatische Datensatzverriegelung sperrt den Datensatz direkt vor dem Lesen durch den Funktionsaufruf isread(3). Die Verriegelung wird nach dem naechsten Aufruf einer C-ISAM-Funktion aufgehoben. Die automatische Datensatzverriegelung sperrt immer einen Datensatz ohne Beachtung der Zeitdauer der Verriegelung.

Anderseits kann die manuelle Datensatzverriegelung eine beliebige Anzahl von Datensaetzen verriegeln. Manuelle Datensatzverriegelung sperrt einen Datensatz, wenn dieser Datensatz mit Hilfe der Funktion isread(3) gelesen wird. Die Verriegelung des gesperrten und aller anderen derzeitig gesperrten Datensaetze wird ueber die Funktion isrelease(3) aufgehoben. Manuelle Datensatzverriegelung wird dann verwendet, wenn eine exaktere Kontrolle ueber die Verriegelung einer oder einer Serie von Datensaetzen erforderlich ist.

Beide Verriegelungsverfahren, das automatische und manuelle, sind "geteilte" Verriegelungen. Andere Prozesse koennen die Datensaetze lesen, die vom gegenwaertigen Prozess gesperrt wurden, doch sie koennen sie nicht verriegeln oder ueberschreiben (rewrite).

4.3.1. Automatische Datensatzverriegelung

Die automatische Datensatzverriegelung muss bei der Eoeffnung der C-ISAM-Datei spezifiziert werden. Diese Spezifikation erfolgt durch das Hinzufuegen des Wertes ISAUTOLOCK zum Parameter mode des Funktionsaufrufes isopen(3) oder isbuild(3). Von dem Zeitpunkt an, wo die Datei geoeffnet wird bis zu dem Moment, da sie geschlossen wird, wird jeder Datensatz automatisch verriegelt, bevor er gelesen wird. Jeder Datensatz bleibt bis zu dem Moment verriegelt, da der naechste C-ISAM-Funktionsaufruf fuer die aktuelle Datei ausgefuehrt wird. Folglich kann waehrend der Verwendung der automatischen Datensatzverriegelung nur ein Datensatz pro C-ISAM Datei zum jeweils gegebenen Zeitpunkt verriegelt werden. Das folgende Beispiel demonstriert eine Datensatzverriegelung.

```
myfd = isopen("myfile", ISINOUT+ISAUTOLOCK);

       ...

isread(myfd, myrecord, ISNEXT);  /* Datensatz wird hier */
                                 /* verriegelt, bevor   */
                                 /* er gelesen wird.    */
       ...

isrewcurr(myfd, myrecord);       /* Datensatz wird hier */
                                 /* freigegeben, wenn   */
                                 /* die Funktion been-  */
       ...                       /* det wurde.          */

isclose(myfd);
```

4.3.2. Manuelle Datensatzverriegelung

Manuelle Datensatzverriegelung muss vor jeder gewuenschten Aktion spezifiziert werden. Dazu ist dem Funktionsaufruf isopen(3) oder isbuild(3) beim Eroeffnen der C-ISAM-Datei der mode-Parameter ISMANULOCK hinzuzufuegen. Nachdem die Datei geoeffnet wurde, muss dem Parameter mode des Funktionsaufrufs isread(3) der Wert ISLOCK hinzugefuegt werden, wenn der gewuenschte Datensatz waehrend des Lesens verriegelt werden soll. Alle Datensaetze, die in dieser Weise gelesen werden, bleiben bis zu dem Zeitpunkt gesperrt, da die Verriegelung fuer alle Datensaetze durch den Aufruf der

C-ISAM-Funktion isrelease(3) aufgehoben wird. Die Anzahl der Datensaetze, die in dieser Weise gleichzeitig verriegelt werden koennen, ist abhaengig vom Betriebssystem. Das folgende Beispiel zeigt die Verriegelung und Freigabe einer Anzahl von Datensaetzen in einer C-ISAM-Datei.

```
myfd = isopen("myfile", ISINOUT+ISMANULOCK);

        ...

isread(myfd, first_record, ISEQUAL+ISLOCK);

        ...

isread(myfd, second_record, ISEQUAL+ISLOCK);

        ...

isread(myfd, third_record, ISEQUAL+ISLOCK);

        ...

isrelease(myfd);    /* Freigabe aller Datensaetze */

        ...

isclose(myfd);
```

Anhang A

Zusammenfassung der C-ISAM-Funktionsaufrufe

Anhang A fasst die C-ISAM-Funktionen zusammen, die detailiert in Teil 3 des WEGA-Programmierhandbuchs beschrieben sind. Alle C-ISAM-Aufrufe geben als Rueckgabewert entweder den Wert 0 (erfolgreiche Abarbeitung) oder -1 (Fehler) zurueck und setzen die globale Integervariable iserrno entweder 0 oder auf den entsprechenden Fehlerkode. Die Funktionen isbuild(3) und isopen(3) geben bei ordnungsgemaesser Abarbeitung einen gueltigen C-ISAM-Dateideskriptor oder den Wert -1 zurueck.

```
isaddindex(isfd, keydesc)
int isfd;
struct keydesc *keydesc;

isaudit(isfd, filename, mode)
int  isfd;
char *filename;
int  mode;

isbuild (filename, recordlength, keydesc, mode)
char *filename;
int recordlength;
struct keydesc *keydesc;
int mode;

isclose(isfd)
int isfd;

isdelete(isfd, record)
int isfd;
char record[];

isdelcurr(isfd)
int isfd;

isdelindex(isfd, keydesc)
int isfd;
struct keydesc *keydesc;

iserase(filename)
char *filename;

isindexinfo(isfd, buffer, number)
int isfd;
int number;
struct keydesc *buffer;
/*  oder
struct dictinfo *buffer;  */

double lddbl(p)
```

```
char  *p;

float  ldfloat(p)
char  *p;

int  ldint(p)
char  *p;

long  ldlong(p)
char  *p;

islock(isfd)
int isfd;

isopen(filename, mode)
char *filename;
int mode;

isread(isfd, buffer, mode)
int isfd;
char buffer[];
int mode;

isrelease(isfd)
int isfd;

isrename(oldname, newname)
char *oldname;
char *newname;

isrewcurr(isfd, record)
int isfd;
char record[];

isrewrite(isfd, record)
int isfd;
char record[];

stdbl(d, p)
double d;
char  *p;

stfloat(f, p)
float  f;
char  *p;

stint(i, p)
int i;
char  *p;

stlong(l, p)
long l;
char *p;
```

```
isstart(isfd, keydesc, length, record, mode)
int isfd;
struct keydesc *keydesc;
int length;
char record[];
int mode;

isuniqueid(isfd, uniqueid)
int isfd;
long *uniqueid;

isunlock(isfd)
int isfd;

iswrite(isfd, record)
int isfd;
char record[];
```

Anhang B

Fehlernachrichten und Statusbytes

B.1. Fehlernachrichten

Treten bei der Benutzung von C-ISAM-Funktionen Fehler auf, wird die globale Integervariable iserrno auf einen Wert im Bereich von 1 bis 113 gesetzt. Dabei bezeichnen Fehlerkodes zwischen 1 und 99 Fehler, die aus dem WEGA-Betriebssystem resultieren, waehrend Fehlerkodes zwischen 100 und 113 auf C-ISAM-Fehler hindeuten. Somit koennen Fehlerkodes, die in der globalen Variable errno erscheinen, ebenfalls in iserrno auftreten.

Tabelle B-1 C-ISAM-Fehlerkodes

Name	Nr.	Status 1	Status 2	Erlaeuterung
EDUPL	100	2	2	Es wurde ein Duplikatwert zum Index mittels iswrite, isrewrite, isrewcurr oder isaddindex hinzugefuegt.
ENOTOPEN	101	9	0	Die C-ISAM-Datei wurde noch nicht mit isopen eroeffnet.
EBADARG	102	9	0	Ein Argument eines C-ISAM-Aufrufs ist fuer diese Funktion nicht zulaessig.
EBADKEY	103	9	0	Eines oder mehrere Elemente der Schluesselbeschreibung enthalten unzulaessige Werte.
ETOOMANY	104	9	0	Die maximale Anzahl der Dateien, die gleichzeitig geoeffnet werden koennen, wurde bei diesem Aufruf ueberschritten.
EBADFILE	105	9	0	Das Format der C-ISAM-Datei wurde zerstoert.
ENOTEXCL	106	9	0	Die Datei, zu der ein Index hinzugefuegt oder geloescht werden soll, muss fuer exklusiven Zugriff eroeffnet sein.
ELOCKED	107	9	0	Der gewuenschte Datensatz bzw. die gewuenschte Datei wurde von einem anderen Nutzer verriegelt.
EKEXISTS	108	9	0	Der Index, der hinzugefuegt werden soll, existiert bereits.
EPRIMKEY	109	9	0	Es wurde der Versuch unternommen, den Primaerschluessel zu loeschen. Der Primaerschluessel kann nicht durch die Funktion isdelindex geloescht werden.
EENDFILE	110	1	0	Der Beginn oder das Ende der Datei wurde erreicht.
ENOREC	111	2	3	Es konnte kein Datensatz mit dem gewuenschten Wert an der spezifizierten Position gefunden werden.
ENOCURR	112	2	1	Dieser Aufruf muss auf dem aktuellen

	Datensatz operieren und es wurde noch keiner definiert.
EFLOCKED 113	Diese Datei ist durch einen anderen Benutzer exklusiv verriegelt.

B.2. Status Bytes

Nach jedem C-ISAM-Aufruf werden in den zwei Statusbytes (isstat1 und issatat2) Statusinformationen abgelegt. Das erste Byte (Tabelle B-2) enthaelt Statusinformationen allgemeiner Natur wie z.B. Erfolg oder Misserfolg des C-ISAM Aufrufs. Das zweite Byte (Tabelle B-3) enthaelt spezifische Informationen, deren Bedeutung in Verbindung mit den Statusinformationen in Byte 1 steht.

Tabelle B-2 C-ISAM-Statusbyte 1

Byte 1	Status
0	Ausfuehrung erfolgreich
1	Dateiende (EOF)
2	Ungueltiger Schluessel
3	Systemfehler
4	benutzerdefinierte Fehler

Tabelle B-3 C-ISAM-Statusbyte 2

Status-byte 1	Statusbyte 2
0-9	0 - Es steht keine weitere Information zur Verfuegung
0	2 - nach einem Lesefunktionsaufruf wird angezeigt, dass der Schluesselwert fuer den aktuellen Schluessel mit dem Wert des selben Schluessels im naechsten Datensatz ueberein-stimmt. - nach einem Funktionsaufruf zum Schreiben oder ueberschreiben wird angezeigt, dass der gerade geschriebene Datensatz ein Dupli-katschluesselwert fuer mindestens einen anderen vorhergehenden Schluesselwert ist.
2	1 - Der Primaerschluesselwert wurde zwischen der erfolgreichen Ausfuehrung einer Leseanweisung und der Ausfuehrung der naechsten REWRITE-Anweisung geaendert. 2 - Es wurde der Versuch unternommen, einen

Datensatz zu schreiben oder zu ueberschreiben, der einen Duplikatschluessel in einer Indexdatei bilden wurde.

3 - Es kann kein Datensatz mit dem spezifizierten Schluessel gefunden werden.

4 - Es wurde der Versuch unternommen, ausserhalb der extern definierten Grenzen einer Indexdatei zu schreiben.

9 Der Wert fuer das Statusbyte 2 wird vom Benutzer definiert.

Anhang C

Datentypen

In diesem Anhang werden die von C-ISAM-Funktionen benutzten Datentypen sowie deren Verwendung beschrieben.

C-ISAM unterstuetzt folgende Datentypen:

CHARTYPE	Zeichen (Bytes)
INTTYPE	2-Byte-Integer
LONGTYPE	4-Byte-Integer
FLOATTYPE	Gleitkommazahlen (einfache Genauigkeit)
DOUBLETYPE	Gleitkommazahlen (doppelte Genauigkeit)

C.1. CHARTYPE

Der Datentyp CHARTYPE bezeichnet in C-ISAM einen Bereich in einer Datendatei, der aus 8-Bit-Werten (Bytes) von 1 bis 255 besteht. Ein typisches Beispiel fuer den Datentyp stellt das Feld fuer den Namen einer Stadt oder die Adresse dar.

C.2. INTTYPE und LONGTYPE

Die Datentypen INTTYPE und LONGTYPE bestehen aus vorzeichenbehafteten Integerwerten der Laenge von 2 bzw. 4 Byte. Integerwerte werden in den Daten- und den Indexdateien stets in der Reihenfolge High-Teil, Low-Teil gespeichert (hoechstwertiges Byte zuerst, niederwertigstes Byte zuletzt). Dieses Speicherungsverfahren ist unabhaengig von der Form, in der Integerwerte im P8000 gespeichert sind. Es entspricht vom Prinzip her dem Datenformat, das vom C-Sprachsystem des P8000 verwendet wird, nur das C-ISAM-Long-Werte nicht auf Wortgrenzen (geraden Speicheradressen) abgelegt werden muessen. C-ISAM bietet dem Nutzer 4 Routinen zur Umwandlung zwischen dem C-ISAM-Integerdatenformat und dem maschinenorientierten Integerdatenformat:

ldint(p) Es wird ein maschinenorientierter Integerwert zurueckgegeben. Der Parameter p ist ein Pointer auf das erste Byte eines C-ISAM-Integerwertes (2 Byte).

stint(i,p) Der 2-Byte-Integerwert i wird als C-ISAM-Integerwert auf der durch den Zeichenpointer p angegebenen Speicheradresse abgelegt. Der Parameter p zeigt dabei auf das 1. Byte des C-ISAM-Integerformats.

ldlong(p) Es wird ein maschinenorientierter 4-Byte-Integerwert zurueckgegeben. Der Parameter p ist ein Pointer auf das erste Byte eines C-ISAM-Integerwertes (4 Byte).

stlong(i,p) Der 4-Byte-Integerwert i wird als C-ISAM-Integerwert auf der durch den Zeichenpointer p angegebenen Speicheradresse abgelegt. Der Parameter p zeigt dabei auf das 1. Byte des C-ISAM-Integerformats.

Diese Routinen werden nach dem Lesen eines C-ISAM-Datensatzes in den Puffer des Benutzers angewendet. Integerwerte, die in Nutzerprogrammen (z.B. in der Sprache C) verwendet werden sollen, muessen erst mit Hilfe der Funktionen ldint bzw ldlong in ein fuer den Prozessor verwendbares Format konvertiert werden.

```
int     int_machine;
long    long_machine;
char    *p_cisam_int, *p_cisam_long;

        ...

int_machine = ldint(p_cisam_int);
long_machine = ldlong(p_cisam_long);
```

Die Konvertierung von Integerwerten im Maschinenformat in das C-ISAM-Datenformat erfolgt mit Hilfe der Funktionen stint bzw stlong.

```
stint(int_machine, p_cisam_int);
stlong(long_machine, p_cisam_long);
```

C.3. FLOATTYPE und DOUBLETYPE

Die Datentypen FLOATTYPE und DOUBLETYPE sind die beiden verwendbaren Gleitkommaformate. Sie entsprechen den Datenformaten, die auch in der Programmiersprache C benutzt werden.

C-ISAM	Sprache C
FLOATTYPE	float
DOUBLETYPE	double

Der einzige Unterschied zwischen den C-ISAM-Gleitkommaformaten und den Gleitkommaformaten der Sprache C besteht darin, dass C-ISAM-Gleitkommzahlen im Speicher nicht auf Wortgrenzen beginnen muessen. Von C-ISAM werden die folgenden 4 zusaetzliche Routinen fuer die Konvertierung zwischen C-ISAM-Gleitkommaformaten und den C-

Gleitkommaformaten bereitgestellt:

ldfloat(p) Es wird eine maschinenorientierte Gleitkommazahl einfacher Genauigkeit zurueckgegeben. Der Parameter p ist ein Pointer auf das 1. Byte der C-ISAM-Gleitkommazahl.

stfloat(f,p) Die Gleitkommazahl f (einfache Genauigkeit) wird als C-ISAM-Gleitkommazahl auf der durch den Zeichenpointer p angegebenen Speicheradresse abgelegt. Der Parameter p zeigt dabei auf das 1. Byte der C-ISAM-Gleitkommazahl.

lddouble(p) Es wird eine maschinenorientierte Gleitkommazahl doppelter Genauigkeit zurueckgegeben. Der Parameter p ist ein Pointer auf das 1. Byte der C-ISAM-Gleitkommazahl.

stdouble(d,p) Die Gleitkommazahl d (doppelte Genauigkeit) wird als C-ISAM-Gleitkommazahl auf der durch den Zeichenpointer p angegebenen Speicheradresse abgelegt. Der Parameter p zeigt dabei auf das 1. Byte der C-ISAM-Gleitkommazahl.

Die Anwendung der Funktionen zur Gleitkommakonvertierung zwischen C-ISAM-Format und C-Format erfolgt analog zu den Integerkonvertierungen.

Anhang D

Deklarationsdateien

D.1. Die Deklarationsdatei isam.h

Die C-ISAM Deklarationsdatei isam.h enthaelt Definitionen, die fuer die Argumente mode verwendet werden sowie die Definitionen der Strukturen, auf die in den Funktionen Bezug genommen wird.

```
/* isam.h */

/*
 *       C-ISAM version x.x
 *  Indexed Sequential Access Method
 */

#define CHARTYPE    0
#define CHARSIZE    1

#define INTTYPE     1
#define INTSIZE     2

#define LONGTYPE    2
#define LONGSIZE    4

#define DOUBLETYPE  3
#define DOUBLESIZE  sizeof(double)

#define FLOATTYPE   4
#define FLOATSIZE   sizeof(float)

#define MAXTYPE     5
#define ISDESC      0200  /* add to make descending type */
#define TYPEMASK    017   /* type mask                   */

#define ldint(p)    (((p)[0]<<8)+((p)[1]&0377))
#define stint(i,p)  ((p)[0]=(i)>>8,(p)[1]=(i))
long ldlong();

#ifndef NOFLOAT
float ldfloat();
double lddbl();
#endif

#define ISFIRST      0    /* position to first record    */
#define ISLAST       1    /* position to last record     */
#define ISNEXT       2    /* position to next record     */
#define ISPREV       3    /* position to previous record */
#define ISCURR       4    /* position to current record  */
#define ISEQUAL      5    /* position to equal value     */
```

```
#define ISGREAT      6    /* position to greater value   */
#define ISGTEQ       7    /* position to >= value        */

/* isread lock modes */
#define ISLOCK     (1<<8)  /* lock record before reading */

/* isopen, isbuild lock modes */
#define ISAUTOLOCK (2<<8)    /* automatic record lock    */
#define ISMANULOCK (4<<8)    /* manual record lock       */
#define ISEXCLLOCK (8<<8)    /* exclusive isam file lock */

#define ISINPUT       0   /* open for input only         */
#define ISOUTPUT      1   /* open for output only        */
#define ISINOUT       2   /* open for input and output   */

/* audit trail mode parameters */
#define AUDSETNAME    0   /* set new audit trail name    */
#define AUDGETNAME    1   /* get audit trail name        */
#define AUDSTART      2   /* start audit trail           */
#define AUDSTOP       3   /* stop audit trail            */
#define AUDINFO       4   /* audit trail running ?       */

#define NPARTS        8   /* maximum number of key parts */

struct keypart
    {
    int kp_start;     /* starting byte of key part */
    int kp_leng;      /* length in bytes           */
    int kp_type;      /* type of key part          */
    };

struct keydesc
    {
    int  k_flags;     /* flags                          */
    int  k_nparts;    /* number of parts in key         */
    struct keypart
    k_part[NPARTS];   /* each key part                  */
             /* the following is for internal use only */
    int  k_len;       /* length of whole key            */
    long k_rootnode;  /* pointer to rootnode            */
    };
#define k_start   k_part[0].kp_start
#define k_leng    k_part[0].kp_leng
#define k_type    k_part[0].kp_type

#define ISNODUPS  000        /* no duplicates allowed  */
#define ISDUPS    001        /* duplicates allowed     */
#define DCOMPRESS 002        /* duplicate compression  */
#define LCOMPRESS 004        /* leading compression    */
#define TCOMPRESS 010        /* trailing compression   */
#define COMPRESS  016        /* all compression        */

struct dictinfo
    {
```

```
    int  di_nkeys;          /* number of keys defined    */
    int  di_recsize;        /* data record size          */
    int  di_idxsize;        /* index record size         */
    long di_nrecords;       /* number of records in file */
    };

#define EDUPL     100        /* duplicate record       */
#define ENOTOPEN  101        /* file not open          */
#define EBADARG   102        /* illegal argument       */
#define EBADKEY   103        /* illegal key desc       */
#define ETOOMANY  104        /* too many files open    */
#define EBADFILE  105        /* bad isam file format   */
#define ENOTEXCL  106        /* non-exclusive access   */
#define ELOCKED   107        /* record locked          */
#define EKEXISTS  108        /* key already exists     */
#define EPRIMKEY  109        /* is primary key         */
#define EENDFILE  110        /* end/begin of file      */
#define ENOREC    111        /* no record found        */
#define ENOCURR   112        /* no current record      */
#define EFLOCKED  113        /* file locked            */
#define EFNAME    114        /* file name too long     */
#define ENOLOK    115        /* can't create lock file */

/*
 * For system call errors
 *   iserrno = errno (system error code 1-99)
 *   iserrio = IO_call + IO_file
 *        IO_call  = what system call
 *        IO_file  = which file caused error
 */

#define IO_OPEN      0x10       /* open()    */
#define IO_CREA      0x20       /* creat()   */
#define IO_SEEK      0x30       /* lseek()   */
#define IO_READ      0x40       /* read()    */
#define IO_WRIT      0x50       /* write()   */
#define IO_LOCK      0x60       /* locking() */
#define IO_IOCTL  0x70          /* ioctl()   */

#define IO_IDX      0x01        /* index file     */
#define IO_DAT      0x02        /* data file      */
#define IO_AUD      0x03        /* audit file     */
#define IO_LOK      0x04        /* lock file      */
#define IO_SEM      0x05        /* semaphore file */

extern int iserrno;         /* isam error return code  */
extern int iserrio;         /* system call error code  */
extern char isstat1;        /* cobol status characters */
extern char isstat2;
extern char *isversnumber;  /* C-ISAM version number   */
extern char *iscopyright;
extern long isserial;

struct audhead
```

```
    {
    char au_type[1];     /* audit record type A,D,R,W */
    char au_time[4];     /* audit date-time           */
    char au_procid[2];   /* process id number         */
    char au_userid[2];   /* user id number            */
    };
#define AUDHEADSIZE  9   /* num of bytes in audit header */
```

Anhang E

Dateiformate

Aufbau des Kopfsatzes einer Indexdatei

Byte	Bedeutung
0	Kontrollbytes (2 byte, Wert: 0xFE53)
2	Anzahl reservierter Bytes am Beginn eines Indexdatensatzes (1 Byte, Wert: 2)
3	Anzahl reservierter Bytes am Ende eines Indexdatensatzes (1 Byte, Wert: 2)
4	Anzahl reservierter Bytes pro Schluesseleintrag einschliesslich Datensatznummer (1 Byte, Wert: 4)
5	Pointertyp und Laengenangabe (1 Byte, Wert: 4)
6	Datensatzlaenge der Indexdatei; relative Dateiflagbytes sind ausgeschlossen (2 Bytes, Wert: 511)
8	Schluesselanzahl (2 Bytes)
10	Flagwort; siehe Deklarationsdatei (2 Bytes, Wert: 0)
12	Dateiversionsnummer (1 Byte)
13	Datenstzlaenge; relative Dateiflagbytes sind nicht enthalten (2 Bytes)
15	Datensatznummer des 1. Datensatzes der Indexbeschreibung (4 Bytes)
19	Reserviert (6 Bytes)
25	Datensatznummer des ersten freien Datensatzes in der Datendatei (4 Bytes)
29	Datensatznummer des ersten freien Datensatzes in der Indexdatei (4 Bytes)
33	Datensatznummer des letzten Datensatzes in der Datendatei (4 Byte)
37	Datensatznummer des letzten Datensatzes in der Indexdatei (4 Byte)
41	Transaktionsnummer (4 Byte)

45 Identifikationsnummer (4 Byte)

49 Pointer auf Zugriffsprotokollinformationen (4 Byte)

Aufbau eines Indexverzeichnisses

Byte	Bedeutung
0	Byteanzahl dieses Eintrags (2 Bytes)
2	Datensatznummer des Datensatzes, der die Fortsetzung enthaelt (4 Bytes)
6	Laenge der Beschreibung (2 Bytes)
8	Adresse des Wurzeleintrags (4 Bytes)
12	Komprimierungsflags (1 Byte)
13	Laenge des Schluesselteils 1; hoechstwertiges Bit = dups (2 Bytes)
15	Position (2 Bytes)
17	Type; 0 = alphanumerisch (1 Byte)
	...
509	Flag (1 Byte, Wert: 0xFF)
510	Flag fuer Datensatzende; bestimmt den Datensatztyp; das hoechstwertige Bit wird fuer Sicherheitszwecke verwendet (1 Byte, Wert: 0x7E).

Fuer jeden Index sind die Eintrage notwendig, die hier fuer einen Schluessel in den Bytepositionen 6 bis 17 angegeben sind. Die in den Positionen 13 bis 17 erlaeuterten Eintrage wiederholen sich fuer jeden Indexteil.

Aufbau der Schluesselwerteintraege

Byte	Bedeutung
0	Anzahl der benutzten Bytes in diesem Eintrag (2 Bytes)
2	Anzahl fuehrender Bytes; nur bei LCOMPRESS oder COMPRESS (1 Byte)
3	Anzahl der Leerzeichen am Schluesselende; nur bei TCOMPRESS oder COMPRESS (1 Byte)
4	Schluesselwert (n Bytes)
4+n	wird fuer Duplikate benutzt (2 Bytes)
6+n	Pointer auf Datensatz; das hoechstwertige Bit wird als Flag fuer Duplikate benutzt (4 Bytes)
	...
509	Indexbaumnummer
510	Ebene innerhalb des Baumes (1 Byte, Wert 0, wenn zum Eintrag keine untergeordneten Eintraege mehr existieren).

Fuer jeden Schluesselwert sind die Eintrage notwendig, die hier fuer einen Schluessel in den Bytepositionen 2 bis 6+n angegeben sind.

Aufbau der Freispeicherliste

Byte	Bedeutung
0	Anzahl der benutzten Bytes in diesem Eintrag (2 Bytes)
2	Datensatznummer der Fortsetzung (4 Bytes)
6	Platz fuer bis zu 126 Datensatznummern (n Bytes)
	...

509 Wert 0xFF: Freispeicherliste der Datendatei
Wert 0xFE: Freispeicherliste der Indexdatei
(1 Byte)

510 Flag fuer Datensatzende; bestimmt den Datensatztyp; das hoechstwertige Bit wird fuer Sicherheitszwecke verwendet (1 Byte, Wert: 0x7F).

Aufbau des Eintrags ueber Zugriffsprotokollierung

Byte	Bedeutung
0	Anzahl der benutzten Bytes in diesem Eintrag (2 Bytes)
2	Flag (2 Bytes) Wert 0: Zugriffsprotokollierung ein Wert 1: Zugriffsprotokollierung aus
4	Pfadname der Protikolldatei (64 Bytes)
	...
509	Flag fuer Datensatzende; bestimmt den Datensatztyp; das hoechstwertige Bit wird fuer Sicherheitszwecke verwendet (1 Byte, Wert: 0x7F).

SCCS

Source Code Control System

Vorwort

Dieser Abschnitt beschreibt das source code control system unter WEGA. Es wird der Umgang mit den einzelnen Kommandos erlaeutert und viele Beispiele gegeben. Als ergaenzende Literatur wird auf das WEGA-Programmierhandbuch Teil 1 verwiesen.

Inhaltsverzeichnis Seite

1. Einleitung

Das Quelldateienverwaltungssystem SCCS ist ein System zur Kontrolle von Veraenderungen an Textdateien (kennzeichnet die Quelldatei und die Dokumentation der Softwaresysteme). Es bietet die Moeglichkeiten der Speicherung, Aktualisierung und Wiederherstellung jeder Version einer Textdatei, die Identifizierung einer wiederhergestellten Datei und bietet die Moeglichkeit aufzuzeichnen, wer jede Veraenderung vorgenommen hat, wann, wo und warum. SCCS ist ein Programmpaket, das unter WEGA laeuft. Es besteht aus folgenden Programmen:

admin cdc comb delta
get help prs rmdel
sact sccsdiff unget val
what.

Es ermoeglicht ein Archiv anzulegen und in diesem den Werdegang (d.h. alle Versionen) eines Dokumentes zu konservieren und damit verbundene Verwaltungstaetigkeiten komfortabel durchzufuehren. Dieses SCCS-Benutzerhandbuch behandelt folgende Themen :

-Wie beginnt man mit SCCS.

-Das Schema, das fuer die Identifizierung von Textversionen, die in der SCCS-Datei enthalten sind, verwendet wird.

-Grundlegende Informationen zur staendigen Benutzung der SCCS-Kommandos, einschliesslich einer Diskussion der besser verwendbaren Argumente.

-Der Schutz und die Kontrolle der SCCS-Dateien, sowie Unterschiede bei der Verwendung des SCCS durch Einzelnutzer und durch Nutzergruppen.

Man kann sich SCCS als einen Verwalter von Dateien vorstellen. Es gestattet die Wiederherstellung bestimmter Versionen der Dateien, es verwaltet an ihnen vorgenommene Veraenderungen, und zeichnet auf, wer die Veraenderung vorgenommen hat, wann, wo und warum. Das ist in Umgebungen wichtig, in denen Programme und die Dokumentation haeufig wiederkehrende Veraenderungen durchmachen (bedingt durch Erhaltungs- und/oder Vergroesserungs-arbeit). Denn es ist manchmal erwuenscht die Version eines Programms oder Dokuments wieder so herzustellen, wie es vor den durchgefuehrten Veraenderungen war. Das kann offensichtlich durch das Aufbewahren von Kopien (auf Papier oder durch andere Mittel) getan werden, doch das wird schnell unkontrollierbar und unwirtschaftlich, da die Anzahl der Programme und Dokumente waechst. SCCS bietet eine ansprechende Loesung, da es die Originaldatei unberuehrt laesst

und wann immer "Veraenderungen" an ihr vorgenommen werden, nur die Veraenderungen speichert. Jede Ausfuehrung einer Veraenderung wird "Delta" genannt.

Dieser Band beinhaltet folgende Abschnitte:

- SCCS fuer Anfaenger: Wie wird eine SCCS-Datei hergestellt, wie wird sie aktualisiert, und wie wird eine ihrer Versionen wiederhergestellt.

- Wie werden Deltas nummeriert: Wie werden Versionen der SCCS-Dateien nummeriert und bezeichnet.

- SCCS Kommando-Festlegungen: Festlegungen und Regeln, die im allgemeinen auf alle SCCS-Kommandos anwendbar sind.

- SCCS-Kommandos: Die Erlaeuterung aller SCCS-Kommandos, einschliesslich der Diskussion besser verwendbarer Argumente.

- SCCS-Dateien: Schutz, Format und Kontrolle von SCCS-Dateien, einschliesslich der Diskussion der Unterschiede zwischen der Nutzung des SCCS als Einzelperson oder als Mitglied einer Gruppe oder eines Projekts. Die Rolle eines "Project sccs administrators" wird vorgestellt.

2. SCCS fuer Anfaenger

Es wird vorausgesetzt, dass der Leser weiss, wie er unter WEGA Dateien bildet und einen der verfuegbaren Texteditoren (siehe vi(1), ex(1) oder ed(1) im WEGA-Programmierhandbuch) verwendet. Eine Reihe von Bedienungsbeisp. werden spaeter aufgefuehrt. Alle sollten ausprobiert werden: Der beste Weg, um SCCS kennenzulernen, ist dessen Verwendung.

Um das Material in diesem Band zu ergaenzen, sollten die detaillierten SCCS-Kommandobeschreibungen in Teil 1 des WEGA-Programmierhandbuches herangezogen werden.

2.1. Aufbau der Versionskontrolle

Von einem Originaldokument werden durch Abspeicherung der Aenderungen ("deltas") die verschiedenen Ausgaben des Originals erzeugt.

Release:

Die Ausgaben eines Dokumentes werden in sogenannte "Release" eingeteilt. Sowohl die Ausgabe ("Version") einer "Release" wie auch die "Release" selbst werden entsprechend ihres Entstehungszeitpunktes fortlaufend hochgezaehlt.

```
(Rel1-Version1 -Version2 ...
 Rel2-Version1 ...)
```

Gewoehnlich veraendert man die Release-Ziffer um eine groessere Veraenderung an der Datei anzuzeigen.

Version:

Jede "Version" einer Release kann mehrere "Zweige" und mehrere Ausgaben zu diesen Zweigen besitzen ("Zweigversionen"), d.h. z.B. zu Rel1 Version2 existiert Zweigversion1 von Zweig1 (1.2.1.1) und Zweigversion2 von Zweig1 (1.2.1.2). Sie werden entsprechend zu den Ausgaben der "Release" fortlaufend hochgezaehlt.

Nummerierung:

Die x-te Zweigversion des z-ten Zweiges der v-ten Version der r-ten Release wird mit r.v.z.x bezeichnet. Die r.v Ausgabe laesst sich auch als r.v.0.0-te Ausgabe verstehen.

Beispiel:

```
1.1
1.2
1.3 - 1.3.1.1 - 1.3.1.2
      1.3.2.1 - 1.3.2.2
1.4
2.1
```

SID:

Die Nummer, die man einer Ausgabe zuordnet, wird SCCS-IDentifikation ("SID") genannt und dient zur eindeutigen Festlegung der Ausgabe. Bei vielen SCCS-Kommandos wird bei Nichtangabe der SID die SID der aktuellen (juengsten) Ausgabe eingesetzt (diese muss nicht die letzte Ausgabe der letzten Release sein).

2.2. 'admin' - Das Erstellen einer SCCS-Datei

Betrachten wir z.B. eine Datei, die "lang" genannt wird und eine Liste von Programmiersprachen enthaelt.

```
c pascal fortran cobol algol
```

Wir wollen SCCS die Aufsicht ueber diese Datei erteilen. Das folgende admin- Kommando, das zur Verwaltung der SCCS-Dateien verwendet wird, schafft eine SCCS-Datei und initialisiert Delta 1.1 von der Datei "lang":

```
admin -ilang s.lang
```

Alle SCCS-Dateien muessen mit "s." beginnende Bezeichnungen besitzen, folglich "s.lang". Die Option -i zeigt gemeinsam mit dessen Argument "lang" an, dass admin eine neue SCCS-Datei schaffen soll und initialisiert sie mit dem Inhalt der Datei "lang". Diese Ausgangsversion ist ein Satz von Veraenderungen, die an der Null-SCCS-Datei ausgefuehrt wurden und heisst Delta 1.1 . Das admin-Kommando antwortet:

```
No id keywords (cm 7)
```

Das ist eine warnende Nachricht die auch durch andere SCCS-Kommandos ausgegeben werden kann. Ihre Bedeutung wird im Abschnitt 5.1. naeher beschrieben. In den folgenden Beispielen wird diese warnende Nachricht nicht gezeigt, obwohl sie durch verschiedene Kommandos tatsaechlich ausgegeben werden kann.

Die Datei "lang" kann entfernt werden ,da sie sich durch die Verwendung des get-Kommandos (weiter unten) leicht wiederherstellen laesst:

```
rm lang
```

2.3. 'get' - Die Wiederherstellung einer Datei

Das Kommando 'get s.lang' bewirkt die Bildung (Wiederherstellung) der letzten Version der Datei "s.lang", und gibt die folgenden Nachrichten aus:

```
1.1
5 lines
No id keywords (cm 7)
```

Das bedeutet, dass get die Version 1.1 der Datei wiederhergestellt hat, die aus 5 Textzeilen besteht. Der wiederhergestellte Text wird in einer Datei plaziert, deren Bezeichnung durch das Loeschen des "s."-Praefix von der Bezeichnung der SCCS-Datei gebildet wird. Folglich

wird die Datei "lang" geschaffen.

Das obige get-Kommando schafft nur die Datei "lang" read-only, und beinhaltet keine Informationen, die deren Schaffen betreffen. Um andererseits in der Lage zu sein, an der SCCS-Datei anschliessend Veraenderungen durch das Delta-Kommando (siehe unten) vorzunehmen, muss das get-Kommando ueber deine Absichten, dieses zu tun, informiert sein. Das geschieht wie folgt:

```
get -e s.lang
```

Die Option -e veranlasst get zur Schaffung einer Datei "lang" sowohl zum Lesen als auch zum Schreiben (so dass sie editiert werden kann) und plaziert bestimmte Informationen ueber die SCCS-Datei in eine andere neue Datei, die p.Datei genannt wird. Diese p.Datei wird dann spaeter durch das Delta- Kommando gelesen. Das get-Kommando gibt die selben Nachrichten wie zuvor aus, nur zusaetzlich wird das SID der Version (1.2) ebenfalls ausgegeben. Das Kommando 'get -e s.lang' gibt z.B. folgendes aus:

```
1.1
new delta 1.2
5 lines
```

Die Datei "lang" kann nun, z.B. durch

```
ed lang 29 $a snobol ratfor w 43 q
```

veraendert werden.

2.4. 'delta' - Das Aufzeichnen von Veraenderungen

Um innerhalb der SCCS-Datei die Veraenderungen aufzuzeichnen, die an "lang" durchgefuehrt wurden, geben Sie bitte folgendes Kommando ein:

```
delta s.lang
```

Delta fragt mit:

```
comments?
```

nach einem Benutzerkommentar, der der aktuellen Aenderung mitgegeben werden soll. z.B.

```
comments? zusetzen mehrerer Sprachen
```

Delta liest dann die p.Datei und ermittelt, welche Veraenderungen an der Datei "lang" vorgenommen wurden. Dazu wird automatisch ein get-Kommando erzeugt, um die Originalversion wiederherzustellen. Jetzt wird unter Verwendung

von diff(1) aus der Originalversion und der editierten Version eine Differenzdatei erstellt.

Anmerkung:

Alle Hinweise in der Form 'name(#)' verweisen auf eine naehere Beschreibung von 'name' die in der Sektion '#' des WEGA-Programmierhandbuchs zu finden ist. Die Bearbeitung wird beendet, wenn alle Veraenderungen an "lang" in "s.lang" gespeichert sind und delta meldet:

```
No id keywords (cm7)
1.2
2 inserted
0 deleted
5 unchanged
```

Die Ziffer "1.2" ist die SID fuer das gerade geschaffene Delta, und die naechsten drei Zeilen des Output weisen auf die Anzahl der Zeilen in der Datei "s.lang" hin. Es wurden 2 Zeilen eingefuegt, keine Zeile geloescht und 5 Zeilen wurden nicht geaendert.

2.5. Mehr ueber das 'get'-Kommando

Wie wir gesehen haben, stellt 'get s.lang' die letzte Version (jetzt 1.2) der Datei "s.lang" wieder her. Das geschieht durch das Beginnen mit der Originalversion der Datei und die erfolgreiche Verwendung der geordneten Deltas (Veraenderungen), bis alle verwendet wurden.

Fuer unser Beispiel sind die folgenden Kommandos alle gleichwertig:

```
get s.lang
get -r1 s.lang
get -r1.2 s.lang
```

Die der Option -r folgenden Ziffern sind SIDs (siehe Abschnitt 2.1. oben).

Beachte, dass das Weglassen der Versions-Ziffer der SID (wie im zweiten Beispiel oben) einer Spezifizierung der hoechsten Versions-Ziffer, die in der angegebenen Release existiert, gleichkommt. Somit bewirkt das 2. Kommando die Wiederherstellung der letzten Version in Release 1, also 1.2 . Das 3. Kommando dagegen bewirkt speziell die Wiederherstellung einer speziellen Version, in diesem Falle auch 1.2 .

Immer dann, wenn wirklich grundlegende Veraenderungen an einer Datei vorgenommen wurden, wird die Bedeutung dieser Veraenderung gewoehnlich durch eine neue Release-Ziffer (die erste Komponente von SID) des hergestellten Deltas

angegeben. Da die normale, automatische Nummerierung der Deltas durch die Erhoehung der Versions-Ziffer (zweite Komponente von SID) vor sich geht, muessen wir SCCS mitteilen, dass wir die Release-Ziffer veraendern wollen. Das geschieht durch das get-Kommando:

```
get -e -r2 s.lang
```

Da Release 2 nicht existiert, stellt get die letzte Version vor Release 2 wieder her. Es interpretiert dies auch als Forderung, die Release-Ziffer des Deltas, das wir zu 2 machen wollen, zu veraendern. Dadurch wird also die neue Ausgabe mit 2.1 und nicht mit 1.3 bezeichnet. Diese Information wird spaeter durch die p.Datei zum delta-Kommando uebertragen. Danach meldet sich get mit:

```
1.2
new delta 2.1
7 lines
```

Das zeigt an, dass die Version 1.2 wiederhergestellt wurde, und dass 2.1 die Version ist, die durch das Delta-Kommando geschaffen wird. Falls die Datei jetzt editiert wird, z.B. durch

```
ed lang
43
/cobol/d
w
37
q
```

und das Delta-Kommando fuehrte

```
delta s.lang

comments? Aus Sprachliste ist cobol geloescht
```

aus, werden wir anhand der Delta-Meldung sehen, dass die Version 2.1 tatsaechlich erstellt wurde:

```
No id keywords (cm7)
2.1
0 inserted                (keine Zeile eingefuegt   )
1 deleted                 (  1   Zeile geloescht   )
6 unchanged               (  6   Zeilen unveraendert)
```

Jetzt koennen in Release2 Deltas erstellt werden (Deltas 2.2, 2.3 usw.), oder es kann auch eine neue Release in einer entsprechenden Weise erzeugt werden. Dieser Prozess kann nach belieben fortgesetzt werden.

2.6. 'help' - Kommando

Wenn z.B. das Kommando

```
get  abc
```

ausgefuehrt ist, wird die folgende Nachricht ausgegeben:

```
ERROR [abc] : not an SCCS file (col)
```

Die Zeichenkette "col" ist ein Codename mit dessen hilfe eine zusaetzliche Erklaerung angefordert werden kann. Dazu geben Sie bitte folgendes help- Kommando ein:

```
help col
```

Dieses Kommando erzeugt dann die Ausschrift:

```
col:
"not an SCCS file"
A file that you think is an SCCS file
does not begin with the characters "s."  .
```

Folglich ist help ein nuetzliches Kommando wenn es irgendwelche Zweifel an der Bedeutung einer SCCS Nachricht gibt. Auf diese Weise koennen weiterfuehrende Erklaerungen fast aller SCCS-Nachrichten gefunden werden.

3. Wie Deltas nummeriert werden

Man kann sich die Deltas, die an einer SCCS-Datei ausgefuehrt werden, als die Knoten eines Baumes vorstellen, wo die Wurzel die Ausgangsversion der Datei ist. Das Wurzel-Delta (Knoten) wird normalerweise mit "1.1" bezeichnet und die nachfolgenden Deltas (Knoten) mit "1.2", "1.3" usw. Die Komponenten der Deltabezeichnungen werden "Release" bzw. "Level"-Ziffern genannt. Folglich geht die normale Benennung der nachfolgenden Deltas durch die Zunahme der Levelziffer vonstatten, was durch SCCS automatisch ausgefuehrt wird, wann immer Sie ein Delta schaffen . Die Entwicklung einer SCCS-Datei kann dargestellt werden wie in dieser Abbildung.

```
1.1 -- 1.2 -- 1.3 -- 1.4 -- 1.5 ---- 2.1 -- 2.2 -- 2.3 ...
 R E L E A S E 1                     R E L E A S E 2
```

So eine Struktur kann als "Stamm" des SCCS-Bauems bezeichnet werden. Sie stellt die normale aufeinanderfolgende Entwicklung einer SCCS-Datei dar, in der Veraenderungen, die ein Teil jedes gegebenen Deltas sind, von allen vorangegangenen Deltas abhaengen.

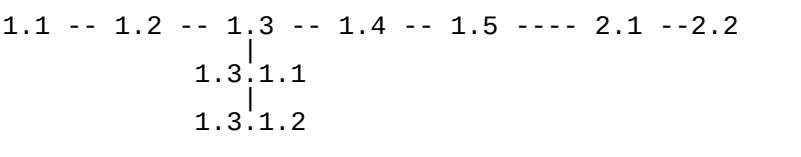
Es gibt jedoch auch Situationen, in denen es notwendig ist, Veraestelungen des Baumes zu bewirken, in welchem die als Teil des gegebenen Deltas verwendeten Veraenderungen nicht von den vorangegangenen Deltas abhaengig sind. Betrachten wir als Beispiel ein Programm, deren Version 1.3 sich in Produktion befindet, und fuer welches die Entwicklungsarbeit auf Release 2 bereits im Gange ist. Folglich kann Release 2 bereits einige Deltas haben, genau wie in der Abbildung dargestellt. Nehmen wir an, dass ein Bearbeieter ein Problem in Version 1.3 erkennt, und dass die Natur des Problems so gestaltet ist, dass es nicht darauf warten kann, in Release 2 repariert zu werden. Die zur Loesung des Problems notwendigen Veraenderungen werden als Delta an Version 1.3 (die Produktionsversion) ausgefuehrt. Dies schafft eine neue Version(1.3.1.1), die dann fuer den Nutzer freigegeben wird, die fuer Release 2 (d.h. Delta 1.4, 2.1, 2.2 usw.) ausgefuehrten Veraenderungen jedoch nicht beeinflusst.

Das neue Delta ist ein Knoten an einem "Zweig" des Baumes, dessen Bezeichnung aus '4' Komponenten besteht, naemlich den Release- und Level- Ziffern, wie bei den Stammdeltas, plus "Zweig"- und "folge"-Ziffern, wie folgt:

```
release.Version.Zweig.Zweigversion
```

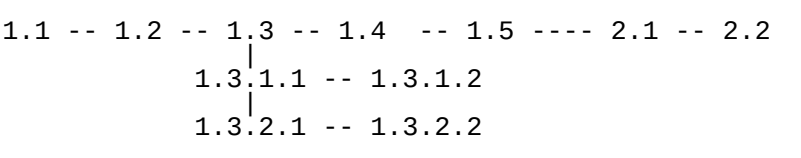
Die Zweig-Ziffer wird jedem Zweig zugewiesen, der ein Abkoemmling eines spezifischen Stamm-Deltas ist, mit dem ersten Zweig, der 1 ist, der naechste 2 usw. Die Folge-Ziffer wird in geordneter Weise jedem Delta auf einem entsprechenden

Zweig zugewiesen. Danach identifiziert das SID 1.3.1.2 das zweite Delta des ersten Zweigs, der vom Delta 1.3 abstammt. Das wird in der folgenden Abbildung dargestellt.



Dieses Konzept der Veraestelung kann bis zu jedem Delta im Baum ausgedehnt werden. Die Bezeichnung der resultierenden Deltas geht in der hier veranschaulichten Weise vonstatten.

Zwei Beobachtungen sind im Hinblick auf die Benennung der Deltas von Bedeutung. Erstens, beinhalten die Bezeichnungen der Stammdeltas genau 2 Komponenten, und die Bezeichnungen der Zweig-Deltas genau 4 Komponenten. Zweitens sind die beiden ersten Komponenten der Zweig-Delta-Bezeichnung jene des Ahnenstammdelta, und die Zweigkomponente wird in der Reihenfolge der Zweigbildung zugewiesen, unabhaengig von deren Standort bezueglich des Stammdeltas. Folglich kann ein Zweigdelta stets durch dessen Bezeichnung identifiziert werden. Obwohl das Ahnenstammdelta durch die Bezeichnung des Zweigdeltas identifiziert werden kann, ist es unmoeglich, den gesamten Weg vom Stammdelta zum Zweigdelta zu bestimmen. Wenn z.B. Delta 1.3 einen Zweig besitzt, der von ihm ausgeht, werden alle Deltas auf jenem Ast mit 1.3.1.h bezeichnet. Wenn dann ein Delta auf diesem Zweig einen anderen von ihm ausgehenden Ast hat, werden alle Deltas auf dem neuen Zweig mit 1.3.2.h (siehe Abbildung) bezeichnet. Die einzige Information, die von der Bezeichnung des Delta 1.3.2.2 stammen kann, ist, dass es das chronologisch zweite Delta auf dem chronologisch zweiten Zweig ist, dessen Stammahne das Delta 1.3 darstellt. Insbesondere ist es unmoeglich, von der Bezeichnung des Deltas 1.3.2.2 alle Deltas zwischen ihm uns seinem Stammahnen (1.3) zu bestimmen.



Es ist klar, dass das Konzept der Zweigdeltas das Erstellen beliebig zusammengesetzter Baumstrukturen zulaesst. Obwohl diese Faehigkeit fuer bestimmte Verwendungszwecke zur Verfuegung steht, wird nachdruecklich empfohlen, den SCCS-Baum so einfach wie moeglich zu halten, da der Umfang seiner Struktur infolge

der groesser werdenden Zusammensetzung des Baumes extrem schwierig wird.

4. SCCS-Kommandosyntax

Dieser Abschnitt erlaeutert die Regeln, die fuer die SCCS-Kommandos angewendet werden. Diese Regeln gelten fuer fast alle SCCS-Kommandos (ausser fuer help, what, sccsdiff und val). Fuer SCCS-Kommandos sind 2 Typen von Argumenten gueltig:

Optionen und
Datei-Argumente.

Optionen beginnen mit einem Minuszeichen (-), dem ein Kleinbuchstabe folgt und dem in einigen Faellen ein Parameter folgt. Diese Optionen steuern die Ausfuehrung des Kommandos, fuer das sie angegeben wurden.

Datei-Argumente (sie koennen Bezeichnungen von Dateien und/oder Verzeichnissen sein) spezifizieren die Datei(en), die vom gegebenen SCCS- Kommando bearbeitet werden sollen. Das Benennen eines Verzeichnisses kommt dem Benennen aller SCCS-Dateien innerhalb des Verzeichnisses gleich. Nicht- SCCS-Dateien und nicht lesbare Dateien (aufgrund der Dateizugriffsrechte in den genannten Verzeichnissen; siehe chmod(1)) werden stillschweigend ignoriert.

Im allgemeinen muessen Dateiargumente nicht mit einem Minuszeichen beginnen. Wenn die Bezeichnung "-" (ein einzelnes Minuszeichen) jedoch als ein Argument fuer ein Kommando angegeben ist, liest das Kommando die Optionen wie auch die SCCS-Archivnamen von <stdin>. Jede Zeile wird getrennt ausgefuehrt. Die Verbindung wird durch Eingabe von EOF (^D bzw. ^Z) auf <stdin> beendet. Diese Moeglichkeit wird oft in Verbindung mit dem find(1)- oder ls(1)- Kommando, verwendet. Noch einmal: Bezeichnungen von Nicht-SCCS- Dateien und nicht lesbaren Dateien werden stillschweigend ignoriert!

Alle Optionen, die in einem Kommando angegeben werden, sind auf alle Datei-Argumente des Kommandos anwendbar. Die Optionen werden vor jedem Datei-Argument bearbeitet, daher ist die Plazierung der Optionen beliebig (d.h., Optionen koennen mit Datei- Argumenten durchsetzt sein). Die Datei-Argumente werden von links nach rechts abgearbeitet.

Etwas anders ist die Syntax bei den Kommandos help, what, sccsdiff und val (siehe Abschnitte 5.5., 5.8., 5.9. und 5.11.). Bestimmte Handlungen verschiedener SCCS-Kommandos werden durch die in den SCCS-Dateien enthaltenen "flags" gesteuert. Einige dieser flags werden noch beschrieben. Eine vollstaendige Beschreibung all dieser flags finden Sie im WEGA-Programmierhandbuch Teil1/1 unter admin(1).

Die Dateizugriffsrechte sind fuer die Ausfuehrung verschiedener

SCCS-Kommandos von Bedeutung. Es wird angenommen, dass alle Nutzer eine Dateizugriffsberechtigung besitzen (siehe Abschnitt 6.1.).

Alle SCCS-Kommandos modifizieren eine SCCS-Datei indem mit einer Kopie dieser Datei gearbeitet wird. Diese Kopie wird x-Datei genannt. Sie sichert dass die SCCS-Datei nicht beschaedigt wird und andere SCCS-Prozesse nicht gestoert werden. Die Bezeichnung der x-Datei wird durch das Ersetzen des "s." der SCCS-Dateibezeichnung durch "x." gebildet. Wenn die Bearbeitung beendet ist, wird die alte SCCS-Datei entfernt und die x-Datei zur aktuellen SCCS-Datei umbenannt. Die x-Datei wird aus den Eintraegen die in der SCCS-Datei enthalten sind gebildet. Der x-Datei werden die gleichen Dateizugriffsrechte gegeben wie der SCCS-Datei (siehe chmod(1)). Um simultane Aktualisierungen an einer SCCS-Datei zu verhindern, schaffen die Kommandos, die die SCCS-Dateien modifizieren, eine lock-Datei (z-Datei genannt), deren Bezeichnung durch das Ersetzen von "s." der SCCS- Dateibezeichnung durch "z." gebildet wird. Die z-Datei beinhaltet die "Prozessnummer" des Kommandos, und die Existenz dieser z-Datei ist ein Hinweis fuer andere Kommandos, dass diese SCCS-Datei aktualisiert wird. Folglich koennen andere Kommandos, die SCCS-Dateien modifizieren, eine SCCS-Datei nicht bearbeiten, wenn die entsprechende z-Datei existiert. Die z-Datei wird mit dem Dateischutzmodus 444 (read-only; siehe chmod(1)) versehen und gehoert dem momentanen Nutzer. Diese Datei existiert nur fuer die Zeit der Ausfuehrung des Kommandos, das sie gebildet hat. Im allgemeinen koennen die Nutzer x-Dateien und z-Dateien ignorieren. Sie koennen im Falle des Zusammenbrechens (-stossens) von Systemen oder aehnlichen Situationen nuetzlich sein.

SCCS-Kommandos erzeugen auf <stdout> Fehlerhinweise in Form von:

```
ERROR [Name der bearbeiteten Datei]:Meldungstext (code)
```

Der eingeklammerte Code kann als Argument fuer das help-Kommando verwendet werden (siehe Abschnitt 5.5.), um eine weiterfuehrende Erlaeuterung der Fehlernachricht zu erhalten.

Das Entdecken eines fatalen Fehlers waehrend der Bearbeitung einer Datei veranlasst das SCCS-Kommando, die Bearbeitung jener Datei zu beenden und vorschriftsmaessig mit der naechsten Datei weiterzumachen, falls mehr als eine Datei angegeben wurde.

5. SCCS-Kommandos

Dieser Abschnitt beschreibt die Hauptmerkmale aller SCCS-Kommandos. Detaillierte Beschreibungen der Kommandos und all ihrer Argumente werden im WEGA-Programmierhandbuch gegeben und sollten fuer weitere Informationen herangezogen werden. Die unten stehende Erleuterung umfasst nur die gebraeuchlichsten Argumente der verschiedenen SCCS-Kommandos.

Da die Kommandos get und delta am haeufigsten Verwendung finden, werden sie zuerst ausgefuehrt. Die anderen Kommandos folgen entsprechend ihrer Wichtigkeit. Das folgende ist eine Zusammenfassung aller SCCS-Kommandos und deren Hauptfunktionen:

get	Stellt Ausgaben, d.h. vollstaendige Texte der SCCS-Dateien wieder her
delta	Traegt Veraenderungen (Deltas) am Text von SCCS-Dateien in das SCCS-Dateiarchiv ein, d.h. bildet neue Versionen
admin	Bildet SCCS-Dateien und fuehrt Veraenderungen an Parametern der SCCS-Dateien durch
prs	Gibt Teile einer SCCS-Datei im vom Nutzer spezifizierten Format heraus
help	Gibt Erklaerungen fuer Fehlermeldungen
rmdel	Entfernt ein Delta von der SCCS-Datei (gestattet das Entfernen von Deltas, die durch Fehler entstanden sind)
cdc	Veraendert den mit einem Delta verknuepften Kommentar
what	Durchsucht die Datei(en) nach allen Faellen des Auftretens einer bestimmten Struktur (Muster) und gibt aus, was ihr folgt; dies ist nuetzlich beim Suchen identifizierender Informationen, die durch das get-Kommando eingefuegt wurden
sccsdiff	Zeigt die Unterschiede zwischen jeweils 2 Versionen einer SCCS-Datei
comb	Setzt 2 oder mehr aufeinanderfolgende Deltas einer SCCS-Datei zu einem einzigen Delta zusammen; verringert oftmals die Groesse der SCCS-Datei
val	Bestaetigt eine SCCS-Datei
sact	Druckt die von der SCCS-Datei gegenwaertig

herausgegebenen Handlungen ab

unget Macht ein vorangegangenes get auf eine SCCS-Datei rueckgaengig

5.1. get

Das get-Kommando bildet eine Textdatei, die eine spezifische Version einer SCCS-Datei beinhaltet. Die spezifische Version wird wiederhergestellt, indem SCCS mit der Ausgangsversion beginnt und dann die Deltas (Veraenderungen) geordnet durchfuehrt, bis die gewuenschte Version erzielt wird. Die gebildete Datei wird g-Datei genannt. Ihr Name wird durch das Entfernen des "s." von der SCCS-Dateibezeichnung gebildet. Die g-Datei wird im aktuellen Verzeichnis gebildet und nur der Dateieigner hat Zugriffsrecht. Der Dateischutzmodus der der g-Datei zugewiesen wurde, haengt vom aufgerufenen get-Kommando ab (siehe 6.). Der gebraeuchlichste Aufruf von get ist:

```
get s.abc
```

das normalerweise die letzte Version auf dem Stamm des SCCS-Datei-Baums wiederherstellt, und erzeugt (z.B.) auf <stdout>

```
1.3
67 lines
No id keywords (cm7)
```

was anzeigt, dass

1. Version 1.3 der Datei "s.abc" wiederhergestellt wurde (1.3 ist das letzte Stammdelta)

2. diese Version 67 Textzeilen hat

3. keine Identifikations-Schluesselwoerter (id keywords) in der Datei substituiert wurden (siehe Abschnitt 5.1.1. zwecks Erleuterung der ID-Schluesselworte)

Der entstandenen g-Datei (Datei "abc") wird der Dateischutzmodus 444 (read- only) gegeben, da diese spezielle Art des get-Aufrufs nur zur Inspektion und Zusammenstellung der g-Datei und nicht zum Editieren (d.h., nicht fuer das Durchfuehren von Veraenderungen) benutzt wird.

Werden mehrere Dateinamen oder Verzeichnissnamen angegeben, wird fuer jede bearbeitete Datei der jeweilige Dateiname gefolgt von einer entsprechenden Meldung auf <stdout> ausgegeben. Z.B.:

```
get  s.abc  s.def
```

erzeugt:

```
s.abc
1.3
67 lines
No id keywords  (cm7)

s.def
1.7
85 lines
No id keywords  (cm7)
```

5.1.1. IDentifikations-Schluesselworte

Bei der Erzeugung einer g-Datei ist es nuetzlich und informativ, das Datum und die Zeit der Entstehung der wiederhergestellten Version, die Bezeichnung des Moduls usw. in die g-Datei einzutragen, so dass diese Information in einem Lade-Modul erscheint. SCCS bietet einen geeigneten Mechanismus, um dies automatisch vorzunehmen. Identifikations (ID) Schluesselworte, die ueberall in den entstandenen Dateien auftauchen, werden gemaess der Definition ihrer ID-Schluesselworte durch gegebene Werte ersetzt. Das Format eines ID Schluesselworts ist ein Grossbuchstabe mit einem Prozentzeichen (%). Z.B.:

```
%I%
```

wird als ID-Schluesselwort erkannt, und durch die SID der wiederhergestellten Version einer Datei ersetzt. In entsprechender Weise ist %H% als ID-Schluesselwort fuer das gegenwaertige Datum (in Form von "mm/dd/yy") und %M% als Bezeichnung der g-Datei definiert. Folglich erhaelt man bei der Ausfuehrung von get an einer SCCS-Datei, die die C-Notation

```
static  char  Version [] = "%M% %I %H"
```

enthaelt, z.B. folgendes:

```
static  char  Version [] = "Dateiname 2.3 09/19/88"
```

Wenn kein ID-Schluesselwort durch get substituiert werden kann, wird die folgende Nachricht ausgegeben:

```
No id keywords  (cm7)
```

Diese Nachricht wird normalerweise durch get als Warnung behandelt. Ist dagegen beim get-Aufruf das i-flag gesetzt, wird die Nachricht wie ein fataler Fehler behandelt (siehe Abschnitt 5.2. fuer weitere Informationen).

Eine vollstaendigen Auflistung der 20 moeglichen ID-Schluesselworte sind in get (1) zu finden.

5.1.2. Die Wiederherstellung verschiedener Versionen

Normalerweise ist die default-Version das letzte Delta der mit der hoechsten Zahl versehenen Release auf einem Stamm des SCCS-Dateibaums. Wenn jedoch die in Bearbeitung befindliche Datei ein d (default SID) flag hat, wird die SID des d-flags als ein default verwendet. Das default SID wird genau in der gleichen Weise interpretiert, wie der mit der -r Option versehene Wert von get.

Die -r Option wird verwendet, um ein wiederherzustellendes SID zu spezifizieren. In diesem Falle wird das d (default SID) flag (falls vorhanden) ignoriert. Z.B. stellt

```
get -r1.3 s.abc
```

die Version 1.3 der Datei "s.abc" wieder her und erzeugt (z.B.) auf <stdout>:

```
1.3
64 lines
```

Ein Zweigdelta kann in entsprechender Weise wiederhergestellt werden:

```
get -r1.5.2.3 s.abc,
```

welches auf <stdout>

```
1.5.2.3
234 Zeilen
```

erzeugt.

Wenn bei der -r Option ein Zwei- oder Vierkomponenten SID angegeben ist, und die Version in der SCCS-Datei nicht existiert, hat das eine Fehlernachricht zur Folge.

Das Weglassen der Versions-Ziffer, wie bei

```
get -r3 s.abc
```

bewirkt die Wiederherstellung des Stammdeltas mit der hoechsten Versions-Ziffer innerhalb der gegebenen Release, wenn die gegebene Release existiert. Folglich koennte das obige Kommando

```
3.7
213 lines
```

ausgeben.

Wenn die gegebene Release nicht existiert, stellt get das Stammdelta mit der hoechsten Release und Version her welche kleiner ist als die angegebene Release-Ziffer. Nehmen wir z.B. an, dass Release 9 in der Datei "s.abc" nicht existiert, und dass Release 7 die momentan hoechstbezifferte Release ist, koennte die Ausfuehrung von

```
get -r9 s.abc
```

erzeugen:

```
7.6
420 lines,
```

was anzeigt, dass das Stammdelta 7.6 die letzte Version der Datei "s.abc" unterhalb von Release 9 ist.

Entsprechend hat das Weglassen der Zweigversionsziffer, wie in

```
get -r.4.3.2 s.abc
```

die Wiederherstellung des Zweigdeltas mit der hoechsten Zweigversionsziffer auf dem gegebenen Zweig, falls er existiert, zur Folge. Wenn der gegebene Zweig nicht existiert, zieht das eine Fehlernachricht nach sich. Ein Beispiel fuer die fehlerfreie Meldung koennte so aussehen:

```
4.3.2.8
89 lines
```

Die -t Option wird verwendet, um die letzte ("top")Version in einer bestimmten Release (d.h., wenn sie mit keiner -r Option versehen wurde, oder wenn ihr Wert nur eine Releaseziffer ist) wiederherzustellen. Die letzte Version wird als jenes Delta definiert, das zuletzt vorgenommen wurde, unabhaengig von seiner Stellung auf dem SCCS-Dateibaum. Wenn also 3.5 das letzte Delta in Release 3 ist, koennte

```
get -r3 -t s.abc
```

erzeugen:

```
3.5
59 lines
```

Wenn jedoch das Zweigdelta 3.2.1.5 das letzte Delta (nach Delta 3.5 gebildet) waere, koennte das gleiche Kommando

```
3.2.1.5
46 lines
```

erzeugen.

5.1.3. Die Wiederherstellung einer SCCS-Datei um ein Delta zu erzeugen

Wird in einem get-Kommando eine -e Option angegeben, prueft get:

1. die "Benutzerliste" (das ist eine Liste mit login-Namen und/oder Gruppen- IDs von Benutzern, denen es gestattet ist, Deltas vorzunehmen (siehe Abschnitt 6.2)), um festzustellen, ob der login-Name oder Gruppen-ID des Benutzers auf der Liste steht. Beachte, dass eine Null (leere) Benutzerliste sich so verhaelt, als ob sie alle moeglichen login-Namen enthaelt.

2. dass die Release (R) der sich in Wiederherstellung befindlichen Version die Relation

niedrigste R. <= R <= hoechste R.

erfuellt, um festzustellen, ob die zugaenglich gemachte Release, eine geschuetzte Release ist (flrel und chrel sind als flags in der SCCS-Datei spezifiziert).

3. dass die Release (R) nicht gegen Editieren gesichert ist. Lock ist als flag in der SCCS-Datei spezifiziert.

4. ob an dieser SCCS-Datei mehrere Benutzer gleichzeitig editieren duerfen, (wenn ja, muss das j-flag gesetzt sein. (gleichzeitig laufende Editierungen werden in Abschnitt 5.1.5. beschrieben).

Das Fehlen einer der ersten drei Bedingungen bewirkt den Abbruch der Bearbeitung der entsprechenden SCCS-Datei. Wenn die obigen Ueberpruefungen erfolgreich verliefen, bewirkt die -e Option die Bildung einer g-Datei im gegenwaertigen Verzeichnis mit dem Dateischutzmodus des Dateibesitzers, Mode=644 (von jedem lesbar, nur vom Besitzer schreibbar). Wenn bereits eine schreibbare g-Datei existiert, beendet get mit einer Fehlermeldung. Das geschieht, um eine versehentliche Zerstoerung einer bereits existierenden g- Datei zu verhindern.

ID-Schluesselworte, die in der g-Datei auftauchen, werden durch get nicht substitutiert, wenn die -e Option angegeben ist, da die entstandene g-Datei nachfolgend fuer die Bildung eines weiteren Deltas genutzt werden soll, und das Ersetzen von ID-Schluesselworten verursachen wuerde, dass sie innerhalb der SCCS-Datei permanent veraendert werden. In Anbetracht dessen muss get nicht nach ID-Schluesselworten

in der g-Datei suchen, so dass die Nachricht

```
No id keywords (cm7)
```

niemals ausgegeben wird, wenn get mit der -e Option aufgerufen wurde.

Ausserdem bewirkt die -e Option die Bildung (oder Aktualisierung) einer p- Datei. Diese wird zur Uebertragung von Informationen an das delta-Kommando verwendet (siehe Abschnitt 5.1.4.). Das folgende ist ein Beispiel fuer die Verwendung der -e Option:

```
get -e s.abc
```

erzeugt (z.B.) auf <stdout>

```
1.3
new delta 1.4
67 lines
```

Wenn die -r und/oder -t Option zusammen mit der -e Option verwendet werden, wird die Version so wiederhergestellt, wie durch die -r und/oder -t Option spezifiziert. Die -i und -x Optionen koennen zur Bezeichnung einer Liste (Syntax dieser Liste, siehe get(1)) mittels get ein- bzw. auszuschliessender Deltas verwendet werden. Das Einschliessen eines Deltas bedeutet, dass die Veraenderungen, die das spezifische Delta darstellen, zum Einfuegen in die wiederhergestellte Version gezwungen werden. Das ist nuetzlich, wenn jemand die selben Veraenderungen an mehr als einer Version der SCCS-Datei vornehmen will. Das Ausschliessen eines Deltas heisst, dass es gezwungen wird, nicht ausgefuehrt zu werden. Das kann verwendet werden, um in der zu bildenden Version der SCCS-Datei die Wirkungen eines vorangegangenen Deltas rueckgaengig zu machen. Wann immer Deltas ein- oder ausgeschlossen werden, ueberprueft get auf moegliche Ueberlagerungen zwischen diesen und jenen Deltas, die normalerweise bei der Wiederherstellung der spezifischen Version der SCCS-Datei verwendet werden (zwei Deltas koennen sich ueberlagern, wenn z.B. jedes der beiden die selbe Zeile der wiederhergestellten g-Datei veraendert). Jede Ueberlagerung wird durch eine Warnung angezeigt. Sie stellt den Zeilenbereich der wiederhergestellten g-Datei dar, in dem das Problem aufgetreten ist. Vom Benutzer wird erwartet, dass er die g-Datei prueft, um festzustellen, ob tatsaechlich ein Problem aufgetreten ist. Ist das der Fall, muss das Problem z.B. durch editieren der Datei geloest werden.

Die -i und -x Optionen sollten mit besonderer Sorgfalt verwendet werden.

Die -k Option wurde geschaffen, um die Erneuerung einer g-Datei zu erleichtern, die nach der Ausfuehrung

von get mit der -e Option moeglicherweise unbeabsichtigt entfernt oder zerstoert wurde, oder in der das Ersetzen der ID-Schluesselworte unterschlagen wurde. Folglich ist eine g-Datei, die durch die -k Option entstand, mit der identisch, die durch get und einer -e Option hergestell wurde. Es findet jedoch keine Bearbeitung statt, die mit der p-Datei in Verbindung steht.

5.1.4. Gleichzeitiges Editieren mehrerer Versionen

Die Faehigkeit, verschiedene Versionen einer SCCS-Datei wiederherzustellen, laesst es zu, dass von mehreren Benutzern gleichzeitig eine Reihe von Deltas durchgefuehrt werden. Das bedeutet, dass eine Reihe von get-Kommandos mit der -e Option auf derselben Datei ausgefuehrt werden koennen. Dabei duerfen zwei get-Aufrufe nicht dieselbe Version herstellen (sofern gleichzeitiges Editieren nicht zugelassen ist, siehe Abschnitt 5.1.5.). Die p-Datei (die durch die -e Option des get-Kommandos gebildet wird) erhaelt ihren Namen indem in der SCCS-Dateibezeichnung das "s." durch "p." ersetzt wird. Sie wird in dem Verzeichnis gebildet, indem sich auch die SCCS-Datei befindet. Sie wird mit dem Dateischutzmodus 644 (fuer jeden lesbar, nur fuer den Besitzer schreibbar) versehen. Die p-Datei enthaelt fuer jedes noch in Arbeit befindliche Delta die folgenden Informationen:

- das SID der wiederhergestellten Version
- das SID, das dem neuen Delta nach dessen Erstellung gegeben wird
- den login-Namen des Nutzers, der get ausfuehrt.

Die erste Ausfuehrung von "get -e" bewirkt die Bildung der p-Datei fuer die entsprechende SCCS-Datei. Nachfolgende Ausfuehrungen aktualisieren die p- Datei nur durch das Einfuegen einer Zeile, die die obigen Informationen enthaelt. Bevor diese Zeile eingefuegt wird, kontrolliert get jedoch, dass keine Eintragung in der p-Datei gemacht wird, die das SID der Version spezifiziert, die bereits wiederhergestellt wurde, sofern nicht parallellaufende Editierungen zugelassen sind. Wenn beide Ausfuehrungen von get erfolgreich verliefen, weiss der Benutzer, dass andere Deltas in Bearbeitung sind. Wenn eine der beiden/beide get-Aufrufe nicht erfolgreich verliefen, hat das eine Fehlernachricht zur Folge. Es ist wichtig zu beachten, dass die verschiedenartigen Ausfuehrungen von get nur von verschiedenen Verzeichnissen (Directorys) aus aufgerufen werden. Andernfalls wird nur der erste Aufruf gelingen, da nachfolgende Aufrufe den Versuch des Ueberschreibens einer schreibbaren g-Datei unternehmen wuerden. Dieser Versuch wuerde jedoch eine SCCS-Fehlermeldung nachsich ziehen.

In der Praxis werden solche mehrfachen Kommandoaufrufe von verschiedenen Benutzern vorgenommen. Wenn jeder Benutzer ein anderes Arbeitsdirectory besitzt, tritt oben genanntes Problem nicht auf (siehe Abschnitt 6.1. wie es verschiedenen Nutzern gestattet wird, die SCCS-Kommandos auf dieselbe Datei anzuwenden).

Tabelle 5-1 stellt die zweckmaessigsten Anwendungsfaellefaelle dar. Es wird gezeigt:

- welche Version einer SCCS-Datei durch get wiederhergestellt wird

- welches SID der Version in Folge eines Deltas entsteht

- die Wirkung einer bei get angegebenen Option auf die Bildung des SID

Tabelle 5-1

	SID Angabe @	-b Option benutzt %	weitere Bedingungen	SID der letzten Version	SID durch Delta-erstellung
1.	keine #	nein	R Standard fuer mR	mR.mV	mR.(mV+1)
2.	keine #	ja	R Standard fuer mR	mR.mV	mR.mV.(mZ+1).1
3.	R	nein	R > mR	mR.mV	R.1 &
4.	R	nein	R = mR	mR.mV	mR.(mV+1)
5.	R	ja	R > mR	mR.mV	mR.mV.(mZ+1).1
6.	R	ja	R = mR	mR.mV	mR.mV.(mZ+1).1
7.	R	-	R < mR und R = mR R nicht existent	hR.mV ** mR.mV	hR.mV.(mZ+1).1 mR.mV.(mZ+1).1
8.	R	-	Nachkomme von Release > R und R existiert	R.mV	R.mV.(mZ+1).1
9.	R.V	nein	kein Nachkomme	R.V	R.(V+1)
10.	R.V	ja	kein Nachkomme	R.V	R.V.(mZ+1).1
11.	R.V	-	Nachkomme von Release>R	R.V	R.V.(mZ+1).1
12.	R.V.Z	nein	kein Zweig Nachkomme	R.V.Z.mS	R.V.Z.(mS+1)
13.	R.V.Z	ja	kein Zweig Nachkomme	R.V.Z.mS	R.V.(mZ+1).1
14.	R.V.Z.S	nein	kein Zweig Nachkomme	R.V.Z.S	R.V.Z.(S+1)
15.	R.V.Z.S	ja	kein Zweig Nachkomme	R.V.Z.S	R.V.(mZ+1).1
16.	R.V.Z.S	-	Zweig Nachkomme	R.V.Z.S	R.V.(mZ+1).1

Zeichenerklaerung:

@ : "R", "V", "Z" und "S" sind die "Release"-, "Version"-, "Zweig"- bzw. "Zweigversions"-komponenten der SID.

m : bedeutet "maximum". Folglich bedeutet "R.mV" z.B., die maximale Versions-Ziffer in Release R; "R.V.(mZ+1).1" bedeutet "die erste Folgeziffer auf dem neuen Zweig (d.h., maximale Zweigziffer plus 1) der Version V in der Release R". Beachte, wenn die angegebene SID von der Form "R.V", "R.V.Z" oder "R.V.Z.S" ist, muss jede der spezifizierten Komponenten existieren.

% : Die -b Option ist nur wirksam, wenn das b flag (siehe admin (1)) in der Datei vorhanden ist. In dieser Tabelle bedeutet eine Liste von "-" "irrelevant".

: Dieser Fall ist anwendbar, wenn das d-flag (default SID) in der Datei nicht vorhanden ist. Wenn das d-flag vorhanden ist, wird das vom d-flag erhaltene SID so interpretiert, als waere es auf der Kommandozeile spezifiziert worden. Folglich ist einer der anderen Faelle in dieser Tabelle anwendbar.

& : Dieser Fall wird verwendet, um die Bildung des ersten Deltas in einer neuen Release zu erzwingen.

**: "hR" ist die hoechste existierende Release, die niedriger ist, als die spezifizierte nichtexistierende Release.

5.1.5. Gleichzeitiges Editieren derselben Version

Unter normalen Bedingungen ist es nicht erlaubt, dass gleichzeitig mehrere get-Kommandos zur Editierung (-e Option ist spezifiziert) mit derselben SID ausgefuehrt werden. D.h., dass delta ausgefuehrt sein muss, bevor ein nachfolgendes get zum editieren auf derselben SID wie das vorangegangene get ausgefuehrt wird. Gleichzeitig mehrere Editierungen derselben wiederhergestellten SID sind jedoch zugelassen, wenn das j-flag (admin(1)) in der SCCS-Datei gesetzt ist. Im folgenden Beispiel werden zwei Nutzer (login-Namen: otto und emil) eine SCCS-Datei mit derselben SID erstellen. Bedingung ist, dass beim admin- Kommando das j-flag gesetzt wurde, also Mehrfacheditierung zugelassen ist (z.B. admin -fj s.abc). Beide Nutzer haben in ihrem login-Directory ein Sub-Directory mit dem Namen "sccs" angelegt. In diesem bearbeiten sie ihre SCCS- Dateien. Otto ist der Projektleiter und nennt die SCCS-Archivdatei mit dem Namen:

"/z/otto/sccs/s.abc" sein eigen. Emil ist sein Mitarbeiter. Otto hat schon begonnen eine weiterentwickelte Version zu bearbeiten. Folglich kann Otto eingeben:

```
   cd z/otto/sccs
   get  -e  s.abc
1.1
new delta 1.2
5 lines
```

Das get-Kommando mit der Option -e erzeugt eine Datei "/z/otto/sccs/abc", die editiert werden kann und eine Datei "/z/otto/sccs/p.abc", mit dem Inhalt:

"1.1 1.2 otto 88/09/21 10:31:21".

Fuenf Minuten spaeter moechte Emil Fehler in der letzten Version korrigieren und gibt aus seinem Directory folgendes ein:

```
   cd z/emil/sccs
   get  -e  /z/otto/sccs/s.abc
1.1
WARNING:
      being edited: `1.1 1.2 otto 88/09/21 10:31:21' (ge18)
new delta 1.1.1.1
5 lines
```

Die Warnung wurde ausgegeben, weil die Version 1.1 bereits von einem anderen Nutzer hergestellt wurde der schon die Version 1.2 bearbeitet. Dieser Nutzer ist Otto und er tat dies am 21.9.1988 um 10 Uhr 31. Dieses zweite get-Kommando erzeugt eine Datei "/z/emil/sccs/abc" und haengt an die bestehende Datei "/z/otto/sccs/p.abc" folgende Zeile an:

"1.1 1.1.1.1 emil 88/09/21 10:36:48".

Die Datei "/z/otto/sccs/p.abc" sieht demnach so aus:

```
     cat /z/otto/sccs/p.abc
  1.1 1.2 otto 88/09/21 10:31:21
  1.1 1.1.1.1 emil 88/09/21 10:36:48
```

In diesem Falle erzeugt ein Delta-Kommando, das mit dem ersten get uebereinstimmt (z.B. delta -r1.2 s.abc) delta 1.2 (vorausgesetzt, 1.1 ist das zuletzt erzeugte Stammdelta). Das Delta-Kommando, das mit dem zweiten get uebereinstimmt (delta -r1.1.1.1 /z/otto/s.abc) erzeugt Delta 1.1.1.1.

5.1.6. Optionen

Die Angabe der -p Option veranlasst get, den wiederhergestellten Text auf <stdout> auszugeben anstatt eine g-Datei anzulegen. Ausserdem werden alle Verarbeitungsmeldungen, die normalerweise auf <stdout> ausgegeben werden (wie SID der wiederhergestellten Version und die Anzahl der wiederhergestellten Zeilen), statt dessen auf <stderr> gelegt. Das kann verwendet werden, um z.B. g-Dateien mit beliebigen Bezeichnungen zu bilden. Beispiel:

```
get -p s.abc >'Dateiname' .
```

Die -s Option unterschlaegt die gesammte Ausgabe, die normalerweise zu <stdout> geht. Folgich werden das SID der wiederhergestellten Version, die Anzahl der Zeilen usw. nicht ausgegeben. Das beeinflusst jedoch nicht die Fehlermeldungen fuer <stderr>. Diese Option wird verwendet, um das Erscheinen von Meldungen auf dem Terminal des Benutzers zu verhindern und wird oft in Verbindung mit der -p Option gebraucht, um den Ausgabestrom von get umzuleiten, wie bei:

```
get -p -s s.abc | nroff
```

Die -g Option sorgt dafuer, die tatsaechliche Wiederherstellung des Textes einer Version der SCCS-Datei zu unterdruecken. Das kann in einer Reihe von Verfahren nuetzlich sein. Es kann mit -g z.B. die Existenz einer bestimmten SID in einer SCCS-Datei nachgewiesen werden. Beispiel:

```
get -g -r4.3 s.abc
```

Dies gibt die angegebene SID, falls sie in der SCCS-Datei vorhanden ist, aus oder erzeugt eine Fehlernachricht, wenn sie nicht existiert. Eine andere Verwendungsmoeglichkeit der -g Option besteht in der Wiederherstellung einer p-Datei, die moeglicherweise unbeabsichtigt zerstoert wurde.

```
get -e -g s.abc
```

Die -l Option bewirkt die Erzeugung einer l-Datei, die durch das Ersetzen des "s" einer SCCS-Dateibezeichnung durch "l" bezeichnet wird. Diese Datei wird im gegenwaertigen Verzeichnis mit dem Dateischutzmodus 444 (read-only) gebildet und ist dem login-Nutzer eigen. Sie enthaelt eine Tabelle (deren Format in get(1) beschrieben wird) die darstellt, welche Deltas zur Rekonstruktion einer bestimmten Version der SCCS-Datei verwendet wurden.

Z.B. erzeugt

```
get -r2.3 -l s.abc
```

eine l-Datei, die darstellt, welche deltas verwendet werden, um die Version 2.3 der SCCS-Datei wiederherzustellen.

Wird ein "p" nach der -l Option angegeben, wie bei

```
get -lp -r.2.3 s.abc
```

wird die Ausgabe der l-Datei auf <stdout> gelenkt. Beachte, dass die -g Option zusammen mit der -l Option zur Unterdrueckung der tatsaechlichen Wiederherstellung des Textes verwendet werden kann.

Die -m Option ist fuer die zeilenweise Identifizierung der durchgefuehrten Veraenderungen an einer SCCS-Datei nuetzlich. Die Angabe dieser Option bewirkt, dass jede Zeile der entstandenen g-Datei durch das SID des Deltas eingeleitet wird, das das Einfuegen jener Zeile verursachte. Das SID ist vom Text der Zeile durch ein tab-Zeichen getrennt.

Die -n Option bewirkt, dass jede Zeile der erzeugten g-Datei durch den Wert des ID-Schluesselwortes %M% (siehe Abschnitt 5.1.1.) und einem tab-Zeichen eingeleitet wird. Die -n Option wird meistens in einer pipeline mit grep(1) verwendet. Um z.B. alle Zeilen zu finden, die zu einer gegebenen Struktur in der letzten Version jeder SCCS-Datei in einem Verzeichnis passen, kann folgendes Kommando ausgefuehrt werden:

```
get -p -n -s directory | grep pattern
```

Wenn sowohl die -m als auch die -n Option spezifiziert ist, wird jede Zeile der entstandenen g-Datei durch den Wert des ID-Schluesselwortes %M% und einem tab eingeleitet (das ist die Wirkung der -n Option) und wird gefolgt von der Zeile in dem Format, das durch die -m Option hergestellt wurde. Da die Verwendung der -m Option und/oder der -n Option die Modifizierung der g- Datei-Inhalte bewirkt, darf so eine g-Datei nicht fuer die Bildung eines Deltas verwendet werden. Deshalb kann weder die -m Option noch die -n Option zusammen mit der -e Option angegeben werden. Siehe get(1) bezueglich der vollstaendigen Beschreibung zusaetzlicher get Optionen.

5.2. delta

Das Delta-Kommando wird verwendet, um die an einer g-Datei vorgenommenen Veraenderungen mit der entsprechenden SCCS-Archivdatei zu verbinden, d.h., ein Delta zu bilden und demzufolge eine neue Version der Datei.

Das Aufrufen des Delta-Kommandos verlangt die Existenz einer p-Datei (siehe Abschnitte 5.1.3. und 5.1.4.). Delta prueft die p-Datei, auf das Vorhandensein von Nutzer-login-Namen. Sollte keiner gefunden werden, hat das eine Fehlernachricht zur Folge. Wird Delta mit der -e Option aufgerufen, fuehrt es dieselbe Kontrolle der Zugriffsrechte durch wie get. Sollten die Kontrollen erfolgreich verlaufen, stellt delta fest, was in der g-Datei veraendert wurde, indem es die g-Datei (durch diff(1)) mit seiner eigenen vorlaeufigen Kopie der g-Datei, wie sie vor dem Editieren war, vergleicht. Diese vorlaeufige Kopie der g-Datei wird d-Datei genannt. Ihr Name entsteht durch Ersetzen des "s" der SCCS-Dateibezeichnung durch "d". Dann erfolgt vom Delta-Kommando automatisch die Ausfuehrung eines internen get mit der in der p-Datei angegebenen SID. Die vom Delta-Kommando angeforderte p-Datei muss jene sein, die den login-Namen des Benutzers enthaelt, der das Delta ausfuehrt. Da der Benutzer, der die g-Datei wiederherstellt, jener sein muss, der das Delta bilden wird. Wenn jedoch der login-Name des Benutzers in mehr als einem Eintrag der p-Datei erscheint (d.h., derselbe Benutzer fuehrte get mit der -e Option mehr als einmal auf derselben SCCS-Datei aus), muss delta mit der -r Option verwendet werden, um die SID, die den p-Dateieintrag eindeutig identifiziert, anzugeben. Die angegebene SID kann entweder die durch get wiederhergestellte SID oder die durch Delta zu bildende SID sein. Dieser Eintrag in der p-Datei wird verwendet, um die SID zu ermitteln, die durch das Delta-Kommando gebildet werden wuerde.

In der Praxis ist

```
delta  s.abc
```

der gebraeuchliste Aufruf. Das Delta-Kommando meldet sich mit einer Frage an den Nutzer:

```
comments?
```

Daraufhin kann der Nutzer eine kurze Beschreibung der durchgefuehrten Aenderungen angeben. Dieser Text muss mit einem <cr> enden und kann bis zu 512 Zeichen lang sein. Soll der Text bei der Ausgabe mehrzeilig erscheinen, so ist das <nl>-Symbol: "/" bei der Eingabe in den Text an der Stelle einzufuegen, wo die Zeilenschaltung erfolgen soll.

Wenn die SCCS-Datei ein v-flag hat (siehe Abschnitt 5.3.2.) meldet sich das Delta-Kommando zuerst mit dem Promt-Zeichen

```
MRS?
```

auf dem <stdout>. Dieses Prompt wird nur ausgegeben, wenn

<stdout> ein Terminal ist. Von <stdin> wird dann die "MR-Nummer" (Modification Requestnumbers) gelesen, getrennt durch Leerzeichen und/oder Tab-Zeichen und auf die gleiche Weise begrenzt, wie die Antwort auf das Prompt "comments?". In einer straff kontrollierten Umgebung wird erwartet, dass deltas nur als Ergebnis von Stoermeldungen, Veraenderungswuenschen, Stoerkarten usw. (hier zusammen Modification Requests oder MRs genannt) gebildet werden und, dass es erwuenscht und notwendig ist, solche MR-number(s) in jedem Delta aufzuzeichnen.

Die -y und/oder -m Option wird verwendet, um den Kommentar (comments bzw. MR numbers) in der Kommandozeile mit anzugeben. Z.B.:

```
delta -y"beschreibender Kommentar" -m"mrnum1 mrnum2" s.abc
```

In diesem Fall werden die entsprechenden Prompts nicht ausgegeben, und vom <stdin> wird nicht gelesen. Die -m Option ist nur gestattet, wenn die SCCS- Datei ein v-flag besitzt. Diese Optionen sind nuetzlich, wenn Delta vom Innern einer "shell-Prozedur" (siehe sh(1)) ausgefuehrt wird.

Der Kommentar (comments und/oder MR numbers), ob von delta abgefragt oder durch die Optionen geliefert, wird als Teil der Liste des zu bildenden Deltas aufgezeichnet und fuer alle SCCS-Deltas, die durch denselben Aufruf von delta bearbeitet werden, verwendet. Das schliesst ein, dass alle genannten Dateien ein v-flag haben muessen, wenn delta durch mehr als ein Dateiargument aufgerufen wird und die erste genannte Datei dieses flag besitzt. Entsprechend kann keine der genannten Dateien dieses flag besitzen, wenn die erste genannte Datei es nicht hat. Jede Datei, die mit diesen Regeln nicht konform geht, wird nicht bearbeitet.

Wenn die Bearbeitung beendet ist, gibt delta (auf <stdout>) das SID des gebildeten Deltas (von der p-Datei-Liste erhalten) und die Anzahl der durch delta eingefuegten, geloeschten und unveraendert gelassenen Zeilen aus. Demzufolge koennte

```
1.4
14 inserted
7 deleted
345 unchanged
```

ein typisches Ausgabebild sein.

Es ist moeglich, dass die Anzahl der als eingefuegt, geloescht oder unveraendert gemeldeten Zeilen mittels delta nicht mit den Vorstellungen des Benutzers ueber vorgenommene Veraenderungen an der g-Datei uebereinstimmt. Der Grund dafuer ist, dass es gewoehnlich eine Reihe von Moeglichkeiten gibt, einen Satz solcher Veraenderungen zu

beschreiben, besonders dann, wenn Zeilen in der g-Datei umherbewegt werden und delta wahrscheinlich eine Beschreibung findet, die sich von der Vorstellung des Benutzers unterscheidet. Jedoch sollte die Gesamtsumme der Zeilen des neuen Deltas (die Anzahl der eingefuegten plus die Anzahl der unveraenderten) mit der Anzahl der Zeilen in der editierten g-Datei uebereinstimmen.

Sollte das delta-Kommando im Herstellungsprozess eines Deltas keine ID- Schluesselworte in der editierten g-Datei finden, wird die Nachricht

No id keywords (cm7)

nach den Prompts fuer den Kommentar, aber vor jeder anderen Ausgabe ausgegeben. Das zeigt an, dass alle ID-Schluesselworte, die in der SCCS- Datei existiert haben koennten, durch ihre Werte ersetzt oder waehrend des editierens geloescht worden sind. Die Ursache koennte das Bilden eines Deltas von einer g-Datei sein, die durch ein get ohne die -e Option (Rueckruf, sodass get in jedem Falle die ID-Schluesselworte ersetzt) geschaffen wurde. Es kann auch durch das unbeabsichtigte Loeschen oder Veraendern der ID-Schluesselworte waehrend des Editierens der g-Datei erfolgt sein. Eine andere Moeglichkeit ist, dass die Datei keine ID- Schluesselworte enthalten hat. In jedem Falle ist es dem Benutzer ueberlassen, die Ursache dieser Warnung festzustellen, und notwendige Handlungen einzuleiten. Ist jedoch das i-flag in der SCCS-Datei gesetzt, wird diese Meldung nicht als Warnung behandelt, sondern als fataler Fehler. Im letzteren Fall wird das Delta nicht gebildet. Nachdem die Bearbeitung einer SCCS-Datei beendet ist, wird die entsprechende Eintragung aus der p- Datei entfernt.

Alle Aktualisierungen an einer p-Datei werden an einer vorlaeufigen Kopie, der g-Datei, vorgenommen. Diese Kopie wird x.Datei genannt und ist im obigen Abschnitt 4 beschrieben. Gibt es nur einen Eintrag in der Liste der p-Datei, wird die p-Datei selbst entfernt.

Ausserdem entfernt delta die editierte g-Datei, ausser bei Angabe der -n Option. Folglich erhaelt

delta -n s.abc

die g-Datei bis zum Ende der Bearbeitung.

Die -s ("silent") Option unterdrueckt die gesamte Ausgabe, die normalerweise zu <stdout>, ausser bei den Prompts "comments?" und "MRS". Folglich veranlasst die Verwendung der -s Option zusammen mit der -y Option (und moeglicherweise mit der -m Option) delta, weder von <stdin> zu lesen noch an <stdout> zu schreiben. Die Unterschiede

zwischen der g-Datei und der d-Datei (siehe oben), die das Delta (Veraenderungen) darstellen, koennen auf <stdout> durch Verwendung der -p Option ausgegeben werden. Das Format dieser Ausgabe stimmt mit dem durch diff(1) hergestellten ueberein.

5.3. admin

Das admin Kommando wird verwendet, um die SCCS-Dateien zu verwalten, d.h., neue SCCS-Dateien zu bilden und die Parameter der existierenden zu veraendern. Wenn eine SCCS-Datei gebildet wird, werden deren Parameter durch die Verwendung von Optionen initialisiert, oder ihnen werden default Werte zugewiesen, wenn keine Optionen angegeben sind. Dieselben Optionen werden zur Veraenderung der Parameter von bereits existierenden Dateien verwendet.

Zwei Optionen werden in Verbindung mit dem Aufsuchen und Korrigieren von SCCS-Dateien bereitgestellt und im Abschnitt 6.3. eroertert.

Neugebildeten SCCS-Dateien wird der Dateischutzmode 444 (read-only) gegeben und sie gehoeren dem Dateinutzer (muss nicht immer der Dateibesitzer sein).

Nur ein Nutzer, dem das Schreiben im dem Verzeichnis, das die SCCS-Datei enthaelt, erlaubt ist, kann das admin Kommando auf der Datei verwenden.

5.3.1. Die Bildung von SCCS-Dateien

Eine SCCS-Datei kann durch die Ausfuehrung des Kommandos

```
admin -ifirst s.abc
```

gebildet werden. Der Parameter der -i Option ('first'), gibt die Bezeichnung einer Datei an, von der der Text des Ausgangsdeltas der SCCS-Datei "s.abc" genommen werden soll. Das Weglassen des Parameters der -i Option zeigt an, dass admin von <stdin> den Text des Anfangsdeltas lesen soll. Folglich stimmt das Kommando

```
admin -i s.abc <first
```

mit dem vorangegengenen Beispiel ueberein. Wenn der Text des Anfangsdeltas keine ID Schluesselworte enthaelt, wird die Nachricht

```
No id keywords (cm7)
```

durch admin als Warnung ausgegeben. Wenn jedoch derselbe

Aufruf des Kommandos auch das i-flag setzt (nicht zu verwechseln mit -i Option), wird die Nachricht als ein Fehler behandelt und die SCCS-Datei wird nicht gebildet. SCCS Dateien koennen nur einzeln unter Nutzung der -i Option gebildet werden.

Wenn eine SCCS-Datei gebildet wird, ist die ihrem ersten Delta zugewiesene Release-Ziffer normalerweise "1" und ihre Level-Ziffer stets "1". Demgemaess ist "1.1" normalerweise das erste Delta einer SCCS-Datei. Die -r Option wird verwendet, um die Release-Ziffer, die dem ersten Delta zugewiesen werden soll, zu spezifizieren. Folglich zeigt

```
admin -ifirst -r3 s.abc
```

an, dass das erste Delta jetzt mit "3.1" anstatt mit "1.1" bezeichnet werden sollte. Da diese Option nur bei der Bildung des ersten Deltas bedeutungsvoll ist, ist dessen Verwendung nur mit der -i Option erlaubt.

5.3.2. Das Einfuegen des Kommentars fuer das Anfangsdelta

Wenn eine SCCS-Datei gebildet wird, hatt der Benutzer die Moeglichkeit, einen Kommentar zur Erlaeuterung des Grundes fuer die Bildung der Datei anzugeben. Das wird ermoeglicht durch die Angabe des Kommentars hinter der -y Option und/oder der Angabe der MR-Nummer hinter der -m Option in genau derselben Weise wie fuer delta. Wird die -y Option weggelassen, wird eine Kommentarzeile in Form von

```
date and time created YY/MM/DD HH:MM:SS by loginname
```

automatisch erzeugt. Wenn der Wunsch besteht, MR-Nummern (-m Option) anzugeben, muss das v-flag ebenfalls gesetzt werden (mittels der -f Option, die unten beschrieben wird). Das v-flag stellt nur fest, ob MR-Nummern angegeben werden muessen oder nicht, wenn irgendein SCCS-Kommando verwendet wird, das einen delta-Kommentar (siehe sccsfile(5)) in der SCCS- Datei modifiziert. Daraus folgt:

```
admin -ifirst -mmrnuml -fv s.abc
```

Beachte, dass die -y und -m Optionen nur dann wirkungsvoll sind, wenn eine neue SCCS-Datei gebildet wird.

5.3.3. Die Initialisierung und Modifizierung von SCCS-Dateiparametern

Der Teil der SCCS-Datei, der dem erlaeuternden Text (siehe Abschnitt 6.2.) vorbehalten ist, kann durch die Verwendung der -t Option initialisiert und veraendert werden. Die Absicht des erlaeuternden Textes besteht in der Zusammenfassung der Inhalte und des Zwecks der SCCS-Datei, obwohl ihre Inhalte beliebig sein koennen und sie beliebig lang sein kann.

Wird eine SCCS-Datei unter Angabe einer -t Option gebildet, muss dem t der Name der Datei folgen, von der der erlaeuternde Text gelesen werden soll. Das Kommando

```
admin -ifirst -tdesc s.abc
```

z.B. spezifiziert, dass der erlaeuternde Text von der Datei "desc" genommen werden soll. Bei der Bearbeitung einer existierenden SCCS-Datei spezifiziert die -t Option, dass der gegenwaertig in der Datei befindliche erlaeuternde Text (falls es einen gibt), durch den Text in der genannten Datei ersetzt werden soll. Folglich spezifiziert

```
admin -tdesc s.abc
```

dass der erlaeuternde Text der SCCS-Datei durch die Inhalte von "desc" ersetzt werden soll, das Weglassen der Dateibezeichnung nach dem -t Option, wie in

```
admin -t s.abc
```

bewirkt das Entfernen des erlaeuternden Textes von der SCCS-Datei.

Die flags (siehe Abschnitt 6.2.) einer SCCS-Datei koennen mit Hilfe der -f bzw. -d Option initialisiert, veraendert oder geloescht werden.

Die flags einer SCCS-Datei werden verwendet, um bestimmte Handlungen verschiedener Kommandos zu steuern. Siehe admin(1) bezueglich der Beschreibung aller flags. Z.B. bewirkt das i-flag, dass die Warnnachricht, dass keine ID Schluesselworte in der SCCS-Datei enthalten sind, als ein Fehler behandelt werden soll, und das d (default SID) flag bewirkt dass die default-Version der SCCS-Datei mittels get Kommando wiederhergestellt werden soll. Die -f Option wird verwendet, um ein flag und moeglicherweise dessen Wert zu setzen. Z.B.

```
admin -ifirst -fi -fmmodname s.abc
```

setzt das i-flag und das m-(module name) flag. Der hinter dem m-flag angegebene Name wird vom get-Kommando verwendet um das ID-Schluesselwort %M% zu ersetzen. Ist das m-flag nicht angegeben wird die Bezeichnung der g-Datei als Ersatz fuer das %M%-ID-Schluesselwort verwendet. Beachte, dass mehrere -f Optionen auf einen einzigen Aufruf von admin angegeben werden koennen, und dass -f Optionen fuer

die Bildung einer neuen- oder einer bereits existierenden SCCS-Datei angegeben werden koennen.

Die -d Option wird verwendet, um ein flag von der SCCS-Datei zu loeschen und kann nur angegeben werden, wenn eine existierende Datei bearbeitet wird.

Als Beispiel entfernt das Kommando

```
admin -dm s.abc
```

das m-flag von der SCCS-Datei. Es koennen mehrere -d Optionen bei einem einzigen Aufruf von admin angegeben, und mit den -f Optionen vermischt werden.

SCCS-Dateien enthalten eine Aufstellung ("user list") der login-Namen und/oder Gruppen-IDs von Benutzern, denen es gestattet ist, Deltas herzustellen (siehe Abschnitt 5.1.3. und 6.2.). Diese Liste ist standardmaessig leer, was besagt, dass jeder Deltas herstellen kann. Um in diese Liste login-Namen und/oder Gruppen-IDs einzutragen, wird die -a Option verwendet. Z.B. traegt

```
admin -axyz -awql -a1234 s.abc
```

die login-Namen "xyz" und "wql" und das Gruppen-ID "1234" in die Liste ein. Die -a Option kann entweder bei der Bildung einer neuen Datei oder der Bearbeitung einer Existierenden verwendet werden und kann mehrere Male erscheinen. Die -e Option wird in analoger Weise verwendet, wenn jemand login-Namen oder Gruppen-IDs von der Liste entfernen ("loeschen") moechte.

5.4. prs

Sowohl alle vom Benutzer eingetragenen Texte wie auch alle Verwaltungsinformationen im SCCS-Archiv sind Datenschluesselwoerter zugeordnet (nicht zu verwechseln mit ID-Schluesselwoertern). Mit Hilfe des prs-Kommandos koennen durch Angabe von Datenschluesselwoertern alle im SCCS-Archiv gespeicherten Informationen ganz oder teilweise auf <stdout> ausgegeben werden (siehe Abschnitt 6.2.). Mit Hilfe der -d Option kann eine Zeichenkette angegeben werden. Diese Zeichenkette kann beliebigen Text durchsetzt mit Daten-Schluesselworten enthalten. Daten-Schluesselworte werden durch entsprechende Werte gemaess ihrer Definitionen ersetzt. Z.B. wird

```
:I:
```

als das Daten-Schluesselwort definiert, was durch das SID eines angegebenen Deltas ersetzt wird. Demgemaess wird

:F: als das Daten-Schluesselwort fuer die gegenwaertig bearbeitete SCCS-Dateibezeichnung definiert, und :C: wird definiert als die Kommentarzeile, die mit einem spezifizierten Delta verbunden ist. Alle Teile einer SCCS-Datei haben ein mit ihnen verbundenes Daten-Schluesselwort. Bezueglich der vollstaendigen Auflistung der Daten-Schluesselworte, siehe prs(1).

In der Zeichenkette der -d Option konnen die Daten-Schluesselworte belibig oft auftauchen. Folglich kann

```
prs -d":I: DAS IST EIN DELTA AUS :F: :I" s.abc
```

z.B. auf dem <stdout>

```
2.1 DAS IST EIN DELTA AUS s.abc 2.1
```

erzeugen. Informationen ueber ein einzelnes Delta koennen durch die Angabe jenes Deltas mit Hilfe der -r Option erhalten werden. Z.B. kann

```
prs -d":F:: :I: Kommentarzeile ist: :C:" -r1.4 s.abc
```

die folgende Ausgabe erzeugen

```
s.abc: 1.4 Kommentarzeile ist: THIS IS A COMMENT
```

Wenn die -r Option nicht angegeben ist, so ist der Wert des SID implizit der des zuletzt erzeugten Deltas.

Ausserdem koennen die Informationen ueber die Reihenfolge der Deltas durch die Angabe der -l oder -e Option erlangt werden. Die -e Option fuehrt das prs-Kommando fuer die SID's aus, die aelter sind als die durch die -r Option angegebene SID. Mit der -l Option wird das prs-Kommando fuer alle Ausgaben, die spaeter erzeugt wurden, als die Ausgabe, die durch die -r Option spezifiziert wird, durchgefuehrt. Folglich kann das Kommando

```
prs -d:I: -r1.4 -e s.abc
```

ausgeben:

```
1.4
1.3
1.2.1.1
1.2
1.1
```

und das Kommando

```
prs -d:I: -r1.4 -l s.abc
```

kann

```
3.3
3.2
3.1
2.2.1.1
2.2
2.1
1.4
```

erzeugen. Die Substitution von Daten-Schluesselworten fuer alle Deltas der SCCS-Datei kann durch die Angabe sowohl der -e als auch der -l Option erreicht werden.

5.5. help

Das help Kommando druckt Erklaerungen von SCCS-Kommandos und Nachrichten ab, die diese Kommandos ausgeben koennen. Argumente von help, von denen keines oder mehr angegeben werden koennen, sind nur die Bezeichnungen der SCCS-Kommandos oder die Codeziffern, die in Klammern nach den SCCS-Nachrichten erscheinen. Wenn kein Argument angegeben wird, fragt help selbst nach einem. Help benoetigt keinerlei Optionen oder Dateinamen. Eine erlaeuternde Information, fuer das Argument (falls vorhanden) wird auf <stdout> ausgegeben. Wenn keine Information gefunden wird, erscheint eine Fehlernachricht. Beachte, dass jedes Argument unabhaengig bearbeitet wird, und ein aus einem Argument resultierender Fehler wird die Bearbeitung anderer Argumente nicht beenden.

```
help  ge5  rmdel
```

z.B. erzeugt

```
ge5:
"nonexistent sid"
The specified sid does not exist in the
given file.
Check for typos.
rmdel:
   rmdel  -rSID Datei....
```

5.6. rmdel

Das rmdel-Kommando wurde geschaffen, um das Entfernen eines deltas von einer SCCS-Datei zu gestatten. Die Verwendung sollte jenen Faellen vorbehalten sein, in denen unkorrekte und globale Veraenderungen in einem Teil des zu entfernenden Deltas gemacht wurden.

Das zu entfernende Delta muss ein "Blatt"-Delta (letztes Delta eines Zweiges) sein. D.h., es muss das zuletzt gebildete Delta auf einem Zweig oder einem Stamm des

SCCS-Dateibaums sein. In dem Beispiel eines SCCS-Baumes koennen nur die Deltas 1.3.1.2, 1.3.2.2 und 2.2 entfernt werden; wurden sie einmal entfernt, dann koennen die Deltas 1.3.2.1 und 2.1 entfernt werden, usw.

```
1.1
1.2
1.3 - 1.3.1.1 - 1.3.2.1
      1.3.1.2   1.3.2.2
2.1
2.2
```

Um dem momentanen Nutzer das Entfernen eines Deltas zu gestatten, muss er eine Schreibberechtigung in dem Verzeichnis besitzen, das die SCCS-Datei enthaelt. Ausserdem muss der echte Nutzer entweder jener sein, der das zu entfernende Delta gebildet hat, oder muss der Eigentuemer der SCCS-Datei und deren Verzeichneisses sein.

Die obligatorische -r Option wird verwendet, um das vollstaendige SID des zu entfernenden Deltas zu spezifizieren (d.h., es muss fuer ein Stammdelta 2 Komponenten und fuer ein Zweigdelta 4 Komponenten haben). Folglich spezifiziert

```
rmdel -r.2.3 s.abc
```

das Entfernen des (Stamm)Deltas "2.3" der SCCS-Datei. Vor dem Entfernen des Deltas prueft rmdel, ob die Release-Ziffer (R) des gegebenen SID die Relation

```
niedrigste < = R < = hoechste
```

erfuellt.

Rmdel prueft ebenfalls, ob das spezifizierte SID nicht das der Version ist, fuer die ein get zum editieren ausgefuehrt und das mit ihr verbundene Delta noch nicht hergestellt wurde. Ausserdem muss der login-Name oder das Gruppen-ID in der "Benutzerliste" der Datei erscheinen, oder die "Benutzerliste" muss leer sein. Das spezifizierte Release kann auch nicht gegen ein Editieren gesichert werden (d.h., wenn das l-flag gesetzt ist (siehe admin(1)) darf das spezifizierte Release nicht in der Liste enthalten sein). Wenn diese Bedingungen nicht erfuellt werden, wird die Bearbeitung beendet und das Delta nicht entfernt. Nachdem das spezifizierte Delta entfernt wurde, wird sein Typanzeiger in der "Deltatabelle" der SCCS-Datei (siehe Abschnitt 6.2.) von "D" (fuer "delta") zu "R" (fuer "removed") geaendert.

5.7. cdc

Das cdc-Kommando wird verwendet, um den Kommentar der bei der Bildung eines Deltas angegeben wurde, zu veraendern. Sein Aufruf ist analog zu dem des rmdel-Kommandos, ausser, dass das zu bearbeitende Delta kein Blattdelta sein muss. Z.B. spezifiziert

```
cdc -r3.4 s.abc
```

dass der Kommentar des Deltas "3.4" der SCCS-Datei veraendert werden soll.

Der neue Kommentar wird bei cdc in der gleichen Weise angegeben, wie der beim delta-Kommando. Der alte Kommentar, der mit dem spezifizierten Delta verbunden ist, wird beibehalten, doch mit einer Kommentarzeile, die dessen Veraenderung (d.h. Beseitigung) anzeigt, eingeleitet. Der neue Kommentar wird vor dieser Kommentarzeile eingefuegt. Die "eingefuegte" Kommentarzeile enthaelt den login-Namen des Nutzers, der cdc ausfuehrt, und die Zeit dessen Ausfuehrung.

Cdc gestattet ebenfalls das Loeschen ausgewaehlter MR-Nummern, die mit dem spezifizierten Delta verbunden sind. Dies wird durch das Einleiten der ausgewaehlten MR-Nummern mit dem Zeichen "!" spezifiziert. Folglich fuegt

```
cdc -r1.4 s.abc
MRs? mrnum3 !mrnum1
comments? loeschen falscher MR-Nummer und einfuegen
          richtiger MR-Nummer
```

"mrnum3" ein und loescht "mrnum1" fuer Delta 1.4.

5.8. what

Das what Kommando wird verwendet, um identifizierende Informationen innerhalb jeder SCCS-Datei zu finden. Verzeichnisbezeichnungen und die Angabe von "-" (ein einzelnes Minuszeichen) werden nicht, wie durch andere SCCS-Kommandos, besonders behandelt, und es werden keine Optionen vom Kommando akzeptiert.

What sucht in der(den) gegebenen Datei(en) nach jeglichem Auftreten des Musters: '@(#)' , das den Ersatz fuer das %Z%-ID-Schluesselwort (siehe get(1)) darstellt und gibt auf <stdout> aus, was diesem Muster bis zum ersten Anfuehrungszeichen ("), groesser als (>), backslash (\), newline oder NUL Zeichen folgt. Wenn also z.B. die SCCS-Datei "s.prog.c" (die ein C-Programm darstellt) die Zeile (die %M% und %I% Schluesselworte sind im Abschnitt 5.1.1. definiert)

```
   .
char  id[] = %Z%%M%:%I%";
   .
   .
```

enthaelt, und dann das Kommando

```
get  -r3.4  s.prog.c
```

ausgefuehrt wird, und schliesslich die resultierende g-Datei zur Erzeugung von "prog.o" und "a.out" zusammengestellt ist, erzeugt das Kommando

```
what  prog.c  prog.o  a.out
```

die Ausgabe

```
prog.c:
     prog.c:3.4
prog.o:
     prog.c:3.4
a.out:
     prog.c:3.4
```

Das what-Kommando durchsuchte die angegebenen Dateien nach dem Muster @(#) und gibt den SCCS-Archivnamen und den diesem Muster folgenden String, hier also prog.c:3.4, aus.

5.9. sccsdiff

Das sccsdiff-Kommando bestimmt (und gibt auf <stdout> aus) die Unterschiede zwischen zwei spezifizierten Versionen einer oder mehrerer SCCS-Dateien. Die zu vergleichenden Versionen werden mittels der -r Option spezifiziert, dessen Format dasselbe wie fuer das get Kommando ist. Die zwei Versionen muessen als die ersten beiden Argumente fuer dieses Kommando in der Reihenfolge spezifiziert sein, in der sie gebildet wurden, d.h. die aeltere Version wird zuerst spezifiziert. Alle folgenden Optionen werden als Argumente fuer das pr(1) Kommando (das eigentlich die Unterschiede ermittelt) interpretiert und muessen vor allen Dateibezeichnungen erscheinen. Zu bearbeitende SCCS-Dateien werden zuletzt genannt. Verzeichnisbezeichnungen und eine Bezeichnung mit "-" (ein einzelnes Minuszeichen) sind fuer sccsdiff nicht akzeptabel.

Die Unterschiede werden in der Form ausgegeben, die durch diff(1) erstellt wurde. Das folgende ist ein Beispiel eines sccsdiff-Aufrufs:

```
sccsdiff  -r3.4  -r5.6  s.abc
```

5.10. comb

Comb erzeugt eine Shell-Prozedur (siehe sh(1)), die auf <stdout> ausgegeben wird und mit deren Hilfe man die spezifizierten SCCS-Archive reorganisieren kann. Das Ziel ist, geloeschte Deltas zu eliminieren und nicht mehr benoetigte Deltas zusammenzufassen, um den Speicherplatz, den das SCCS-Archiv benoetigt, zu verringern. (In Ausnahmefaellen kann jedoch das reorganisierte SCCS-Archiv mehr Speicherplatz benoetigen als das urspruengliche).

Bezeichnete SCCS-Dateien werden durch Ausrangieren unerwuenschter Deltas und Zusammenschliessen anderer spezifizierter Deltas reorganisiert. Die Verwendung ist fuer jene SCCS-Dateien gedacht, die Deltas enthalten, die so alt sind, dass sie nicht mehr von Nutzen sind. Die routinemaessige Verwendung von comb ist nicht zu empfehlen. Seine Verwendung sollte auf sehr wenige Male im Leben einer SCCS-Datei beschraenkt werden.

Ohne die Angabe von Optionen bewahrt comb nur "Blattdeltas" und die minimale Anzahl von Ahnendeltas auf, die notwendig sind, um die "Gestalt" des SCCS-Dateibaumes zu erhalten. Die Wirkung dessen besteht in der Eleminierung der "mittleren" Deltas auf dem Stamm und allen Zweigen des Baumes. Folglich wuerden im letzten Beispiel die Deltas 1.2, 1.3.2.1, 1.4 und 2.1 eleminiert werden. Einige der Optionen werden wie folgt zusammengefasst:

Die -p Option spezifiziert das aelteste Delta, das in der Rekonstruktion erhalten bleiben soll. Alle aelteren Deltas werden ausrangiert.

Die -c Option spezifiziert eine Liste (siehe get(1) bezueglich der Syntax so einer Liste) der aufzubewahrenden Deltas. Alle anderen Deltas werden ausrangiert.

Die -s Option bewirkt die Erzeugung einer Shell-Prozedur, die am Ende nur einen Bericht herstellt, der den Prozentsatz des einzusparenden Platzes (falls ueberhaupt) durch die Reorganisation jeder bezeichneten SCCS-Datei zusammenfasst. Es wird empfohlen, comb mit dieser Option (zusaetzlich zu den anderen gewuenschten) vor der eigentlichen Reorganisation laufen zu lassen.

Es sollte beachtet werden, dass die mittels comb entstandene Shell-Prozedur nicht die Garantie dafuer leistet, dass ueberhaupt Platz gespart wird. Es ist faktisch moeglich, dass die reorganisierte Datei groesser als die urspruengliche ist. Beachte ausserdem, dass die Gestalt des SCCS-Dateibaums durch den Rekonstruktionsprozess veraendert werden kann.

5.11. val

Val ueberprueft, ob die mitgegebenen Mudulnamen, Modultypen mit denen in den SCCS-Archiven abgespeicherten uebereinstimmen. Zusaetzlich werden Pruefungen, die die Ausgabennummerierung betreffen, durchgefuehrt. Das Ergebnis wird in Form eines 8 bit Returncodes und zusaetzlich, wenn nicht unterdrueckt, in Langtext auf <stdout> ausgegeben.

Val wird verwendet, um festzustellen, ob eine Datei eine SCCS-Datei ist, die mit den Merkmalen, die durch eine Liste von Optionen spezifiziert werden, uebereinstimmt.

Jedes Merkmal, das nicht uebereinstimmt, wird als Fehler betrachtet.

Val prueft die Existenz eines speziellen Deltas, wenn das SID fuer jenes delta explizit mittels -r Option spezifiziert ist. Der der -y oder -m Option folgende Text wird verwendet, um den Wert zu kontrollieren, der durch das t- bzw. m-flag gesetzt wurde (siehe admin(1) bezueglich einer Beschreibung der flags). Val behandelt das spezielle Argument "-" anders als die SCCS-Kommandos (siehe Teil 4). Dieses Argument gestattet val, die Argumentenliste von <stdin> zu lesen, die es sonst von der Kommandozeile erhaelt. <stdin> wird bis zum end-of-file gelesen. Z.B. prueft

```
val -
-yc -mabc s.abc
-mxyz -ypll s.xyz
```

zuerst, ob die Datei "s.abc" einen Wert "c" fuer deren type-flag und der Wert "abc" fuer deren "module Namen"-flag hat. Wenn die Bearbeitung der ersten Datei einmal beendet ist, bearbeitet val dann die verbleibenen Dateien. In diesem Falle "s.xyz", um festzustellen, ob sie mit Merkmalen uebereinstimmen, die mit den Optionen angegeben wurden.

Val gibt einen 8 bit Return-Code zurueck, der eine Aussage ueber die moeglicherweise entdeckten Fehler macht. D.h., dass jedes gesetzte Bit das Auftreten eines spezifischen Fehlers anzeigt (siehe val(1) bezueglich der Beschreibung der moeglichen Fehler und deren Code). Ausserdem wird eine entsprechende Fehlerbeschreibung ausgegeben, sofern diese nicht durch die -s Option unterdrueckt wurde. Ein Return-Code von "0" zeigt alle bezeichneten Dateien an, die mit den spezifizierten Merkmalen uebereinstimmten.

5.12. sact

Sact prueft SCCS-Dateien, ob sie gegenwaertig herausgegeben werden. Das bedeutet, dass ein "get -e" ohne eine nachfolgende Ausfuehrung von delta ausgefuehrt wurde. Wenn z.B. die SCCS-Datei s.abc herausgegeben wurde, wuerde das Kommando

```
sact s.abc
```

eine aehnliche Information, wie die folgende ausgeben :

```
1.1 1.2 otto 88/12/24 15:02:00
```

Das erste Feld ist das SID des fuer s.abc zuletzt hergestellten Deltas, das zweite Feld spezifiziert das SID fuer das neue Delta, das dritte Feld beinhaltet den login-Namen des Nutzers, der das "get -e" ausfuehrte, und das vierte und fuenfte Feld beinhalten das Datum und Zeit der Ausfuehrung von "get -e".

5.13. unget

Wenn ein "get -e" auf eine SCCS-Datei ausgefuehrt wurde, kann das durch das unget-Kommando wieder "rueckgaengig" gemacht werden. Das muss natuerlich vor dem delta Kommando ausgefuehrt werden. Die einfachste Form des unget Kommandos ist:

```
unget s.abc
```

Das Programm antwortet mit dem SID des Deltas, dass gebildet worden waere. Also wuerde unget im obigen Beispiel

```
1.2
```

ausgeben.

Die Option -s unterdrueckt die Ausgabe des beabsichtigten SID. Die -n Option Verhindert, dass die durch get erzeugte Kopie der zu aendernden Ausgabe im Arbeitsdirectory geloescht wird. Durch die Verwendung der -r Optionen kann ein spezielles SID angegeben werden.

6. SCCS-Dateien

Dieser Teil eroertert verschiedene Themen, die betrachtet werden muessen, bevor SCCS eingehend verwendet wird. Diese Themen behandeln die Schutzmechanismen, auf die sich SCCS stuetzt, das Format der SCCS-Dateien und die empfohlenen Verfahren zur Revision der SCCS-Dateien.

6.1. Der Schutz

SCCS stuetzt sich auf die Faehigkeiten des in betrieb befindlichen Systems, das fuer die meisten Schutzmechanismen erforderlich ist, um unbefugte Veraenderungen an den SCCS-Dateien zu verhindern (d.h., Veraenderungen durch Nicht-SCCS-Kommandos). Die einzigen Schutzeigenschaften, die durch SCCS direkt geboten werden, sind das "release lock"-flag, das "release floor"- und das "ceiling"-flag und die "Nutzerliste" (siehe Abschnitt 5.1.3.). Den neuen durch das admin Kommando gebildeten Dateien wird der Dateischutzmode 444 (read only) gegeben. Es wird empfohlen, diesen Mode nicht zu veraendern, da es jede direkte Modifikation der Dateien durch Nicht-SCCS-Kommandos verhindert. Es wird weiterhin empfohlen, dass den Verzeichnissen, die die SCCS-Dateien enthalten, Mode 755 gegeben wird, das nur dem Eigentuemer des Verzeichnisses die Modifizierung deren Inhalte gestattet. SCCS-Dateien sollten in Verzeichnissen aufbewahrt werden, die nur SCCS-Dateien und alle durch SCCS-Kommandos gebildeten vorlaeufigen Kopien enthalten. Das erleichtert den Schutz und die Revision der SCCS-Dateien (siehe Abschnitt 6.3.). Die Inhalte der Verzeichnisse sollten einer geeigneten logischen Anordnung entsprechen, z.B. Subsysteme eines grossen Projekts.

SCCS-Dateien duerfen nur einen Dateiverbinder (Link) haben. Der Grund dafuer ist, dass die Kommandos SCCS-Dateien durch die Bildung einer vorlaeufigen Kopie der Datei (x-Datei genannt, siehe Teil 4) modifizieren und die alte Datei nach Beendigung der Bearbeitung entfernen und die x-Datei umbenennen. Wenn die alte Datei mehr als eine Dateiverbindung hat, wuerde deren Entfernung und die Umbenennung der x-Datei die Dateiverbindungen zerstoeren. Solche Dateien werden von den SCCS-Kommandos nicht bearbeitet und erzeugen eine Fehlernachricht. Alle SCCS-Dateien muessen Dateinamen besitzen, die mit "s." beginnen.

Wenn nur ein Benutzer SCCS verwendet, sind die echten und momentanen Nutzer-IDs dieselben, und diesem Nutzer-ID sind die Verzeichnisse, die die SCCS-Dateien enthalten, eigen. Deshalb kann SCCS direkt verwendet werden, ohne jegliche einleitende Vorbereitung. In jenen Situationen jedoch, in

denen mehreren Benutzern mit einem einzigen ID die Verantwortung fuer eine SCCS-Datei uebertragen wird (z.B. in grossen Softwareentwicklungsprojekten) muss ein Nutzer (entsprechend ein Nutzer-ID) zum "Eigentuemer" der SCCS-Dateien gewaehlt werden und sie "verwalten" (z.B. mittels admin Kommando). Dieser Benutzer wird als "SCCS-Administrator" jenes Projekts bezeichnet. Da andere Benutzer von SCCS nicht dieselben Privilegien und Genehmigungen wie der SCCS-Administrator haben, sind sie nicht in der Lage, jene Kommandos direkt auszufuehren, die eine Schreibberechtigung (write permission) im Verzeichnis, das die SCCS-Datei enthaelt, erfordern. Deshalb ist ein projektabhaengiges Programm erforderlich, um den get-, delta- und falls erwuenscht, den rmdel- und cdc-Kommandos ein Interface zu bieten. Das Interface program muss dem SCCS-Administrator eigen sein und das "setze Nutzer-ID bei Ausfuehrung"-bit gesetzt haben (siehe chmod(1)), so dass das momentane Nutzer-ID das Nutzer-ID des Administrators ist. Diese Programmfunktion soll das gewuenschte SCCS-Kommando aufrufen und bewirkt, dass es die Privilegien des Interfaceprogramms fuer die Dauer dieser Kommandoausfuehrung erbt. In dieser Weise kann der Besitzer einer SCCS-Datei sie auf Wunsch modifizieren. Anderen Benutzern, deren login-Namen oder Gruppen-IDs in der "Nutzerliste" fuer diese Datei enthalten sind (jedoch nicht deren Eigentuemer sind), werden die notwendigen Genehmigungen nur fuer die Dauer der Ausfuehrung des Interfaceprogramms erteilt. Sie sind folglich in der Lage, die SCCS-Dateien nur mittels delta und moeglicherweise rmdel und cdc zu modifizieren. Das projektabhaengige Interfaceprogramm, wie sein Name besagt, muss fuer jedes Projekt angelegt sein.

Mehr Informationen ueber dieses Interfaceprogramm siehe Anhang A.

6.2. Das Format

SCCS-Dateien setzen sich aus ASCII-Textzeilen zusammen. Vorangegangene Versionen von SCCS verwendeten Nicht-ASCII-Dateien. Deshalb sind Dateien, die durch fruehere Versionen von SCCS gebildet wurden mit dieser Version von SCCS unvereinbar. Diese SCCS-Dateien sind, wie folgt, in 6 Teile eingeteilt:

Checksummme: Eine Zeile, die die "logische" Summe aller Dateimerkmale (checksum selbst ist nicht eingeschlossen) enthaelt

Delta-Tabelle: Informationen ueber jedes Delta, wie dessen Typ, dessen SID, Datum und Zeit der Bildung und Kommentar

Nutzernamen: Die Liste der login-Namen und/oder Gruppen-IDs von Nutzern, denen es erlaubt ist, die Datei durch das Hinzufuegen oder Entfernen von Deltas zu modifizieren

Flags: Indikatoren, die bestimmte Handlungen verschiedener SCCS- Kommandos steuern

Beschreibungstext: Ein beliebiger Text, durch den Benutzer geschaffen; gewoehnlich eine Zusammenfassung der Inhalte und des Zwecks der Datei

Koerper: Der eigentliche Text, der von SCCS verwaltet wird, vermischt mit internen SCCS-Steuerzeilen

Detaillierte Informationen ueber die Inhalte der verschiedenen Bereiche der Datei sind in sccsfile(5) zu finden; checksum ist der einzige Teil der Datei, der hier noch erwaehnt wird. Es ist wichtig zu beachten, dass SCCS-Dateien, da sie ASCII-Dateien sind, durch verschiedene WEGA-Kommandos bearbeitet werden koennen, wie vi(1), ed(1), grep(1) und cat(1). Das ist in solchen Faellen sehr geeignet, wo eine SCCS-Datei manuell modifiziert werden muss (d.h. wenn die Zeit und das Datum eines Deltas unkorrekt aufgezeichnet wurden, da die Systemuhr unkorrekt gesetzt wurde), oder wenn es erwuenscht ist, nur einen "Blick" auf die Datei zu werfen.

Anmerkung:

Die Modifizierung von SCCS-Dateien mit Nicht-SCCS-Kommandos sollte mit aeusserster Vorsicht vorgenommen werden.

6.3. Die Revision

In seltenen Faellen, vielleicht durch ein in betrieb befindliches System oder Hardwarefehlfunktionen veranlasst, kann eine SCCS-Datei oder Teile von ihr (z.B. ein oder mehrere "Bloecke") zerstoert werden. SCCS-Kommandos (wie die meisten WEGA-Kommandos) geben eine Fehlernachricht aus, wenn eine Datei nicht existiert. Ausserdem verwenden SCCS-Kommandos die in der SCCS-Datei gespeicherte Checksumme, um festzustellen, ob eine Datei seit dem letzten Zugriff verfaelscht wurde (moeglicherweise durch das Verlieren eines oder mehrerer Bloecke oder z.B. durch veraenderungen mit ed(1)). Kein SCCS- Kommando wird eine verfaelschte SCCS-Datei bearbeiten, ausser das admin Kommando mit der -h oder -z Option, wie unten beschrieben.

Es wird empfohlen, SCCS-Dateien auf einer regulaeren Basis

auf moegliche Verfaelschungen hin zu ueberpruefen. Der einfachste und schnellste Weg zur Durchfuehrung einer Revision ist die Ausfuehrung des admin Kommandos mit der -h Option auf allen SCCS-Dateien.

```
admin -h s.Datei1 s.Datei2 oder
admin -h directory1 directory2 ....
```

Wenn die neue Checksumme irgendeiner Datei nicht mit der Checksumme in der ersten Zeile jener Datei uebereinstimmt, wird die Nachricht

```
ERROR [s.Dateiname] : corrupted file (co6)
```

fuer jene Datei ausgegeben. Dieser Prozess wird fortgesetzt, bis alle Dateien geprueft wurden. Bei der Pruefung von Verzeichnissen (wie im zweiten Beispiel oben), wird der eben beschriebene Prozess keine fehlenden Dateien entdecken. Ein einfacher Weg um herauszufinden, welche anderen Dateien aus dem Directory vermisst werden, ist die periodische Ausfuehrung des ls(1)-Kommandos auf jenem Verzeichnis und das Vergleichen der Ausgaben der jetzigen und der vorangegegangenen Ausfuehrungen. Jede Datei, deren Bezeichnung in der vorangegangenen Ausgabe jedoch nicht in der Gegenwaertigen erscheint, wurde durch gewisse Mittel entfernt.

Wann immer eine Datei verfaelscht wurde, haengt die Art deren Wiederherstellung vom Ausmass der Verfaelschung ab. Wenn der Schaden gross ist, ist die beste Loesung die Kontaktaufnahme mit dem Systemverwalter, um den Wiederaufbau der Datei von einer 'backup copy' zu erbitten. Im Falle eines kleineren Schadens ist die Reparatur durch die Verwendung des Editors ed(1) moeglich. Im letzteren Fall muss das folgende Kommando nach so einer Reparatur ausgefuehrt werden:

```
admin -z s.Datei
```

Der Zweck besteht im Zurueckrechnen der Checksumme, um es mit dem tatsaechlichen Inhalt der Datei in Uebereinstimmung zu bringen. Nachdem dieses Kommando auf der Datei ausgefuehrt wurde, wird keine Verfaelschung, die existiert haben kann, mehr aufzufinden sein.

Anhang A Aufbau eines SCCS-Interface-Programms

Dieser Anhang erlaeutert die Anwendung eines Source Code Control System Interfaceprogramms, das mehr als einem Benutzer gestattet, die SCCS-Kommandos auf demselben Satz von Dateien zu verwenden.

A.1. Einleitung

Um den Nutzern mit unterschiedlichen Nutzer-ID's die Verwendung von SCCS-Kommandos auf derselben Datei zu gestatten, bietet ein SCCS-Interfaceprogramm eine zeitweilige Bewilligung von notwendigen Dateizugriffsrechten fuer diese Benutzer. Dieses Merkblatt eroertert die Bildung und Anwendung so eines Interfaceprogramms.

A.2. Funktion

Siehe Abschnitt 6.1.

A.3. Ein Grundprogramm

Wenn ein Programm ausgefuehrt wird, erhaelt es (als Argument 0) die Bezeichnung, durch die es aufgerufen wird, gefolgt von allen zusaetzlichen von dem Nutzer bereitgestellten Argumenten. Wenn also einem Programm eine Anzahl von Dateiverbindern (Links) gegeben wird, kann es seine Bearbeitung in Abhaengigkeit davon veraendern, welche Dateiverbindung zu dessen Aufruf verwendet wurde. Dieser Mechanismus wird von einem SCCS-Interfaceprogramm verwendet, um zu bestimmen, welches SCCS-Kommando nachfolgend aufgerufen werden sollte (siehe exec(2)). Ein typisches Interfaceprogramm ("inter.c", in C geschrieben) wird im Anhang I dargestellt. Beachte den Hinweis auf die (nichtgelieferte) Funktion "filearg". Dies ist beabsichtigt, um zu demonstrieren, dass das Interfaceprogramm auch als Vorbearbeiter fuer die SCCS-Kommandos verwendet werden kann. Die Funktion "filearg" z.B. koennte verwendet werden, um die Dateiargumente, die zu den SCCS-Kommandos uebertragen werden sollen durch Einfuegen des vollstaendigen Pfadnamens zu veraendern und folglich unnoetiges tippen durch den Nutzer zu vermeiden. Ausserdem koennte das Programm alle zusaetzlich gewuenschten Standardoptionen liefern.

A.4. Verbinden und Anwenden

Um ein SCCS-Interfaceprogramm zu erstellen, sollte der SCCS-Administrator die folgende Schritte ausfuehren. Es wird angenommen, dass das Interfaceprogramm "inter.c" im Verzeichnis "/z/syz/sccs" ansaessig ist. Folglich stellt die Kommandofolge

```
cd /z/xyz/sccs
cc ... inter.c -o inter ...-/pw
```

"inter.c" zusammen, um das ausfuehrbare module "inter" zu erzeugen (Die Punkte repraesentieren andere Argumente, die erforderlich sein koennten). Der korrekte Dateischutzmode und das "setze Nutzer-ID bei Ausfuehrung"-bit werden durch die Ausfuehrung von

```
chmod 4755 inter
```

gesetzt.

Schliesslich werden neue Dateiverbinder (links) gebildet (die Bezeichnung der Verbinder kann beliebig sein; werden sie angegeben, ist das Interfaceprogramm in der Lage, die Bezeichnungen der aufzurufenden SCCS-Kommandos von ihnen zu bestimmen), z.B. durch

```
ln inter get
ln inter delta
ln inter rmdel
```

Anschliessend kann jeder Benutzer, dessen C-Shell-Parameter, path (siehe csh(1) oder Bourne shell Parameter PATH (siehe sh(1)) das Verzeichnis "/z/xyz/sccs" als jenes spezifiert, das zuerst auf ausfuehrbare Kommandos hin ueberprueft werden soll, z.B. kann man

```
get -e /z/xyz/sccs/s.abc
```

von jedem Verzeichnis ausfuehren, um das Interfaceprogramm (durch sein link "get") aufzurufen. Das Interfaceprogramm fuehrt dann "/usr/bin/get" (das eigentliche SCCS get Kommando) auf die angegebene Datei aus. Wie bereits erwaehnt, koennte das Interfaceprogramm verwendet werden, um den Pfadnamen "/z/xyz/sccs" zu generieren, so dass der Benutzer nur

```
get -e s.abc
```

angeben muesste, um dieselben Ergebnisse zu erzielen.

A.5. Schlussfolgerung

Ein SCCS Interfaceprogramm (Tabelle A-1) wird verwendet, um Nutzern, die unterschiedliche Nutzer-ID's haben, die Verwendung der SCCS-Kommandos auf denselben Dateien zu gestatten. Obwohl dies die primaere Absicht des Programms ist, kann es ebenfalls als Vorbearbeiter fuer SCCS-Kommandos verwendet werden, da es Operationen auf dessen Argumenten ausfuehren kann.

Tabelle A-1 - SCCS-Interface-Program "inter.c"

```
#define LENGTH 80
main(argc, argv)
int argc;       /* Anzahl der Parameter */
char *argv[];   /* Zeigerfeld auf Parameter */
{
        register int i;
        char cmdstr[LENGTH];    /* Kommandostring */
        char *filearg(), *sname(); /* benutz. Funktionen */

        /*
        Bearbeiten der Dateiargumente,
        die nicht mit '-' beginnen.
        */
        for (i = 1; i < argc; i++)
                if (argv[i][0] != '-')
                        argv[i] = filearg(argv[i]);

        /*
        Ermitteln des einfachen Dateinamens, der benutzt
        werden soll.
        (z.B. entfernen des Pfadnamens, wenn vorhanden)
        */
        argv[0] = sname(argv[0]);

        /*
        Aufruf des eigentlichen SCCS-Kommandos mit
        Uebergabe der Parameter
        */
        sprintf(cmdstr, "/usr/bin/%s", argv[0]);
        execv(cmdstr, argv);
}

char
*filearg(s)
char *s;
{
        .
        .
        .
}
```

KOMBINAT VEB
ELEKTRO-APPARATE-WERKE
BERLIN-TREPTOW
»FRIEDRICH EBERT«

HEIM-ELECTRIC
EXPORT-IMPORT
Volkseigener Außenhandelsbetrieb
der Deutschen Demokratischen Republik

EAW-Automatisierungstechnik Export-Import

Storkower Straße 97
Berlin, DDR - 1055
Telefon 432010 · Telex 114158 heel dd

VEB ELEKTRO-APPARATE-WERKE BERLIN-TREPTOW
»FRIEDRICH EBERT«
Stammbetrieb des Kombinats EAW
DDR - 1193 Berlin, Hoffmannstraße 15-26
Fernruf: 2760
Fernschreiber: 0112263 eapparate bln
Drahtwort: eapparate bln