

Betriebssystem SINIX

Einführung

Benutzerhandbuch

Ausgabe Mai 1988 (SINIX V5.2)

... und Schulung?

Zu dem nachstehend beschriebenen Produkt, wie zu fast allen PC- und DV-Themen bieten wir Kurse in München – Berlin – Essen – Frankfurt – Hannover – Wien – Zürich

**Zentrale Auskunft und Info-Material:
Telefon (089) 92 75-33 52**

Siemens AG
Schule für Kommunikations-
und Datentechnik
K Schule KE
Postfach 8309 51, D-8000 München 83

Bestell-Nr. U3901-J-Z95-1
Printed in the Federal Republic of Germany
6360 AG 4882. (7950)

SINIX ist der Name der Siemens-Version des Softwareproduktes XENIX. SINIX enthält Teile, die dem Copyright © von Microsoft (1982) unterliegen; im übrigen unterliegt es dem Copyright von Siemens. Die Rechte an dem Namen SINIX stehen Siemens zu. XENIX ist ein Warenzeichen der Microsoft Corporation. XENIX ist aus UNIX-Systemen unter Lizenz von AT & T entstanden. UNIX ist ein Warenzeichen der Bell Laboratories.

Copyright © an der Übersetzung Siemens AG, 1984, alle Rechte vorbehalten.

Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhaltes nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte vorbehalten, insbesondere für den Fall der Patenterteilung oder GM-Eintragung.

Liefermöglichkeiten und technische Änderungen vorbehalten.

Copyright © Siemens AG 1988

Vorwort

Dieses Buch ist für alle Benutzer gedacht, die mit einem SINIX-Rechner arbeiten. Es beschreibt grundlegende Themen zum Betriebssystem SINIX.

Das Buch ist vorwiegend für SINIX-Anfänger, die sich für das *Betriebssystem* interessieren, zum Einarbeiten geeignet, aber auch für erfahrene SINIX- oder UNIX-Kenner zum Nachschlagen.

Als Vorkenntnisse werden lediglich allgemeine Kenntnisse aus der Datenverarbeitung vorausgesetzt, keine speziellen Systemkenntnisse. Das bedeutet, daß Ihnen Begriffe, wie z.B. Prozedur, Parameter, Datei oder Prozeß schon einmal begegnet sein sollten.

Sind Sie SINIX-Anfänger?

Dann beginnen Sie am besten gleich mit dem 1. Kapitel. Arbeiten Sie sich durch bis zum Kapitel: Editoren. Mit dem Durcharbeiten der ersten Kapitel *Einstieg*, *Dateisystem*, *Shell und Kommandos*, können Sie sich einen guten Einblick in die SINIX-„Philosophie“ verschaffen und haben dann schon die Basis, gleich mit den SINIX-Kommandos zu arbeiten. In diesem Buch sind die SINIX-Kommandos nur mit ihren Namen und Anwendungsbereichen zum Kennenlernen aufgelistet. Die detaillierten Beschreibungen der SINIX-Kommandos finden Sie zusammengefaßt im Buch: SINIX-Kommandos. Mit den Editoren sollten Sie sich erst befassen, nachdem Sie die davor beschriebenen Kapitel gelesen haben.

Sind Sie SINIX- / UNIX-Kenner?

SINIX ist eines der vielen UNIX-Kinder. Ein besonders gut gelungenes. Sicherlich haben Sie sich auf dem Weg zum UNIX-Kenner durch einige Literatur gearbeitet und wissen, wie vielseitig dieses System ist. Wir haben uns in diesem Buch bemüht, das Wichtigste zum Betriebssystem SINIX anzusprechen.

Die originalen UNIX-Begriffe haben wir weitgehend ins Deutsche übersetzt. Wir waren der Meinung, daß z.B. der Begriff *Dateiverzeichnis* selbsterklärender ist als *Directory*. Wir sind auch der Meinung, daß sich jemand unter dem Begriff *Prozedur* mehr vorstellen kann als unter dem Begriff *Script*. Um trotzdem eine Verbindung zur Terminologie anderer Literatur zu halten, sind die englischsprachigen Begriffe oft in Klammern angegeben. Außerdem finden Sie am Ende dieses Buchs eine Zusammenstellung von Fachwörtern englisch - deutsch und deutsch - englisch.

Noch eine Bitte an Sie

Eine erklärende Dokumentation kann kaum für jeden individuellen Bedarf optimal sein. Eine Dokumentation lebt. Sie lebt auch von *Ihren* Verbesserungs-Anregungen und Ideen. Helfen Sie uns, indem Sie uns einfach schreiben, wo Sie auf "Stolpersteine" gestoßen sind, damit wir sie aus dem Weg räumen können.

Manualredaktion K D ST QM 2

Otto-Hahn-Ring 6, 8000 München 83

email: ...!ukc!unido!sinix!sieqm2!man
...!ztivax!sinix!sieqm2|man

Inhalt

Seite

Experten-Begriffe finden Sie am schnellsten im Stichwörter-Verzeichnis am Ende des Buches

1	Einstieg	1-1
	Wie können Sie sich an SINIX anschließen?	1-2
	Was passiert durch das Anschließen an SINIX?	1-4
	Was passiert, wenn Sie über die Tastatur etwas eingeben?	1-5
	Wie können Sie ein Kommando eingeben?	1-6
	Wie können Sie einen Eingabefehler korrigieren?	1-7
	Wie können Sie Ihre Benutzerkennung durch ein Kennwort schützen?	1-8
	Wie können Sie SINIX verlassen?	1-8
2	Das Dateisystem	2-1
2.1	Aufbau des Dateisystems	2-2
2.2	Pfadnamen für Dateien und Dateiverzeichnisse	2-4
	Wie sehen Pfadnamen aus?	2-4
2.3	Wichtige Kommandos	2-5
2.4	Was erlaubt die Baumstruktur?	2-7
2.5	Dateiverzeichnisse	2-8
	Was kennzeichnet ein Dateiverzeichnis?	2-8
	Wie kann man ein Dateiverzeichnis erzeugen?	2-8
	Wie kann man ein Dateiverzeichnis löschen?	2-9
	Bezeichnungen für Dateiverzeichnisse	2-9
2.6	Dateien	2-10
	Was kennzeichnet eine Datei?	2-10
	Wie kann man eine Datei erzeugen?	2-10
	Wie kann man eine Datei löschen?	2-11
2.6.1	Dateien für Geräte	2-12
2.6.2	Sonderzeichen für Dateinamen	2-13
	Entwerten von Sonderzeichen	2-17

2.7	Indexnummern für Dateien und Dateiverzeichnisse	2-18
	Identifikationszeichen für Dateitypen	2-18
	Anzahl der Verweise für ein Dateiverzeichnis	2-19
	Anzahl der Verweise für eine Datei	2-19
2.8	Zugriffsschutz für Dateien und Dateiverzeichnisse	2-22
	Warum sind bei SINIX Dateien und Dateiverzeichnisse geschützt?	2-22
2.8.1	Wodurch sind bei SINIX Dateien und Dateiverzeichnisse geschützt?	2-22
	Benutzerkennung und Kennwort	2-22
	Benutzergruppen	2-22
	Benutzerklassen	2-23
2.8.2	Schutzbits für Dateien und Dateiverzeichnisse	2-23
	Wie kann man Schutzbits setzen?	2-24
	Schutzbits gelten nur für ihre Benutzerklasse	2-25
	Beim x-Bit muß man aufpassen	2-28
	Kontrollierter Dateizugriff über Programm mit s-Bit	2-29
2.9	Die Dateien .profile und /etc/profile	2-33
3	Die Kommandoebene Shell	3-1
3.1	Grundsätzliches über Kommandos	3-3
	Wann kann man ein Kommando eingeben?	3-3
	Wie muß bzw. kann man ein Kommando eingeben?	3-4
	Wie verarbeitet die Shell Kommandos?	3-5
3.2	Umleiten der Standard-Ein-Ausgabe	3-6
	Wie leitet man eine Ausgabe um?	3-6
	Wie leitet man eine Eingabe um?	3-7
3.3	Pipeline	3-8
	Welche Kommandos kann man in Pipes benutzen?	3-10
	Wozu kann man eine Pipeline benutzen?	3-10
3.4	Der Apostrophier-Mechanismus	3-12
3.5	Variablen für die Shell	3-16
	Standard-Variablen für die Shell	3-17
3.6	Shell Prozeduren	3-19
3.6.1	Parameter für Shell-Prozeduren	3-21
	Wie kann man in einer Prozedur mit Stellungsparametern arbeiten?	3-22
	Wie kann man in einer Prozedur mit Kennwortparametern arbeiten?	3-24
	Wie kann man in einer Prozedur mit Variablen	

	arbeiten, die in der Login-Shell definiert sind?	3-27
	Wie übergibt man Variablen über die Kommandozeile an eine Prozedur?	3-28
	Das Kommando: set -k hat zwei Effekte:	3-29
	Umwandeln von Kennwort- in Stellungsparameter	3-30
	Parameter vordefinieren	3-31
3.6.2	Shell-Prozeduren und Prozesse	3-34
	Shell-Prozeduren im Hintergrund ablaufen lassen	3-37
3.7	Ablaufanweisungen für die Shell	3-38
3.7.1	Schleifen steuern: Die break- und continue-Anweisung	3-39
3.7.2	Abfragen und Verzweigen: Die case-Anweisung	3-40
3.7.3	Liste in Schleifen abarbeiten: Die for-Anweisung	3-42
3.7.4	Abfragen und Kommando ausführen: Die if-Anweisung	3-44
3.7.5	Schleife mit Abbruchbedingung: Die while- und until-Anweisung	3-46
3.8	Kommandos der Shell	3-51
3.8.1	: Ein Kommando, das "nichts" macht	3-52
3.8.2	%# Kommentare einfügen	3-52
3.8.3	() und {} Kommandos zusammenfassen	3-53
3.8.4	eval Kommandos übergeben und ausführen	3-55
3.8.5	exec Kommando ausführen und Shell ersetzen	3-56
3.8.6	exit Beenden einer Shell-Prozedur	3-59
3.8.7	export Variablen weiterreichen	3-64
3.8.8	punkt Datei starten ohne neuen Prozeß	3-66
3.8.9	read Prozedur anhalten und etwas einlesen	3-67
3.8.10	readonly Variablen schützen	3-69
3.8.11	set und sh Schalter für die Shell setzen	3-70
3.8.12	shift Verschieben von Operanden	3-74
3.8.13	times Prozeßzeiten auflisten	3-76
3.8.14	trap Shell-Prozedur unterbrechen	3-77
3.8.15	umask Standardeinstellung der Schutzbits ändern	3-79
3.8.16	wait Auf Prozeßabschluß warten	3-80
3.9	Übersicht - Sonderzeichen der Shell	3-81

4	Die SINIX-Kommandos	4-1
4.1	Was Sie zu jedem Kommando wissen sollten	4-3
4.2	Kommandos eingeben, aber richtig	4-5
4.3	Anwendungsbereiche für SINIX-Kommandos	4-9
4.4	Alphabetische Liste der Kommandos	4-10
4.5	Welches Kommando für welche Aufgabe?	4-18
5	Editoren	5-1
	Was können Sie mit einem Editor machen?	5-1
	Es gibt mehrere Editoren	5-2
	Was heißt bildschirmorientiert?	5-3
	Wie arbeitet ein Editor?	5-4
5.1	Der Editor ced. Eine Beispielsitzung	5-6
	Welches Ziel hat diese Beispielsitzung?	5-6
	Was können Sie mit dem ced machen?	5-6
	Ein kleines Beispiel	5-7
5.1.1	Welche Tasten können Sie im ced benutzen?	5-9
	Die Funktionstasten F9 bis F17	5-9
	Löschen und Einfügen von Zeichen	5-10
	Löschen und Einfügen von Zeilen	5-10
	Schreibmarke bewegen	5-10
5.1.2	Der ced arbeitet in verschiedenen Modi	5-11
	Die HELP-Taste	5-12
	Ein Tastendruck - ein Kommando	5-12
5.1.3	Beginnen Sie !	5-13
5.1.4	Wie geben Sie Text ein?	5-15
5.1.5	Wie blättern Sie im Dokument?	5-16
	'Schnelle Schreibmarkenbewegungen'	5-16
	Wie springen Sie an den Rand des Sichtbereichs?	5-16
	Wie springen Sie ans Ende oder den Anfang Ihrer Datei?	5-17
5.1.6	Wie verschieben Sie einen Zeilenbereich?	5-18
	Können Sie mit dem ced Rechtecke verschieben?	5-21
	Wie machen Sie weiter?	5-22
	Warum horizontal einfügen?	
	Kann man auch vertikal einfügen?	5-24
	Wie können Sie einzelne Zeilen verschieben?	5-25
5.1.7	Wie können Sie SINIX-Kommandos aufrufen?	5-26
5.1.8	Wie können Sie einzelne Zeilen bearbeiten?	5-30
5.1.9	Daten bewegen von einer Datei in eine andere	5-34

	Wie können Sie in das alte Dokument zurückschalten?	5-35
	Wie können Sie Ihre Datei sichern?	5-36
5.1.10	Beenden Sie Ihre ced-Sitzung	5-38
5.1.11	Sonstige Kommandos	5-39
	Was gibt es noch für Modi und Kommandos?	5-39
	Wie können Sie Texte in einem Dokument suchen?	5-39
	Wie können Sie in der Datei vor der Schreibmarkenposition suchen?	5-40
	Wie können Sie das Fenster über das Dokument verschieben?	5-41
	Wollen Sie das Fenster auf eine bestimmte Zeile positionieren?	5-41
	Wechseln in die Shell	5-42
	Tasten mit beliebiger Zeichenfolge belegen	5-43
	Tastenbelegung anzeigen	5-46
	Tastenbelegung ändern	5-46
	Bildschirm neu aufbauen	5-46
5.2	Der Editor vi. Eine Beispielsitzung	5-49
	Welches Ziel hat diese Beispielsitzung?	5-49
	Was können Sie mit dem vi machen?	5-49
	Beginnen Sie mit einem kleinen Beispiel	5-50
	Ohne Kommando geht gar nichts	5-51
5.2.1	Der vi arbeitet in verschiedenen Modi	5-54
5.2.2	Kommandoformen	5-56
5.2.3	Der Bildschirm als Arbeitsfeld	5-60
5.2.4	Aufrufen des vi	5-63
5.2.5	Beenden des vi und Sichern des Editor-Puffers	5-65
5.2.6	SINIX-Kommandos ausführen	5-66
5.2.7	Wie blättern Sie in einem Dokument?	5-67
	Fenster positionieren	5-67
	Textstelle suchen	5-69
	Wo bin ich?	5-71
5.2.8	Wie geben Sie neuen Text ein?	5-72
	Korrigieren bei Eingabefehlern	5-72
5.2.9	Schreibmarke positionieren	5-74
5.2.10	Wie bearbeiten Sie vorhandenen Text?	5-77
	Textteile löschen und zurückholen	5-77
	Textteile ändern und erweitern	5-79
	Leerzeilen einfügen	5-79
	Textbearbeitung	5-80
5.2.11	Kommando-Mehrfachausführung	5-81

5.2.12	Arbeiten mit mehreren Dateien	5-82
5.2.13	Mehrere Beispiele zu vi	5-83
5.2.14	Voreinstellungen durch vi-Optionen	5-90
6	Programme	6-1
6.1	Das Programm SCCS	6-1
	Was ist SCCS?	6-1
	Wie arbeiten Sie mit SCCS?	6-2
6.1.1	Die Dateien des SCCS	6-8
	Zugriffsschutz für SCCS-Dateien	6-12
6.1.2	Die SID-Nummer und der SID-Baum	6-13
6.1.3	Was Sie beim SCCS beachten sollten	6-17
6.1.4	Übersicht über die SCCS-Kommandos	6-19
6.2	Das Programm UUCP	6-21
6.2.1	Allgemeines	6-21
6.2.2	Beschreibung	6-24
6.2.2.1	Allgemeines zum Übertragen	6-24
6.2.2.2	Beginn einer uucp-Übertragung	6-25
6.2.2.3	Beginn einer uux-Übertragung	6-28
6.2.2.4	Ablauf eines uucico-Aufrufs	6-30
6.2.2.5	Sonstige Dateien	6-35
6.2.3	Installation	6-49
6.2.3.1	Hardware	6-49
6.2.3.2	Software	6-53
	Inbetriebsname einer UUCP-Kopplung	6-53
	Ablauf einer Installation	6-54
	Hinweise zur Übertragung auf andere UNIX-Rechner	6-56
6.2.4	Kommandos	6-57
6.2.4.1	mail, rmail	6-58
6.2.4.2	uulog	6-59
6.2.4.3	uucp	6-60
6.2.4.4	uucico	6-65
6.2.4.5	uux	6-68
6.2.4.6	uuxqt	6-71
6.2.4.7	uuclean	6-72
6.2.5	Was tun, wenn	6-75

Anhang	A-1
Kommandolisten	A-2
Shell-Übersichten	A-4
SINIX-Arbeitsumgebung	A-12
Reguläre Ausdrücke	A-14
Erweiterte reguläre Ausdrücke	A-16
ASCII-Zeichen	A-17

Fachwörter, englisch-deutsch

Fachwörter, deutsch-englisch

Literatur

Stichwörter

)

)

)

)

1 Einstieg

Wenn Sie die Arbeitsweise des Betriebssystems SINIX noch nicht kennen, wird Ihnen dieses Kapitel helfen, für seine Handhabung ein "Gefühl" zu bekommen und SINIX Kommandos anzuwenden.

An Hand einer einfachen Beispielsitzung sollen erst einmal einige Anfangsfragen geklärt werden.

- Wie können Sie sich an SINIX anschließen ?
- Was passiert durch das Anschließen an SINIX?
- Was passiert, wenn Sie über die Tastatur etwas eingeben?
- Wie können Sie ein Kommando eingeben?
- Wie können Sie Eingabefehler korrigieren?
- Wie können Sie Ihre Benutzerkennung durch ein Kennwort schützen?
- Wie können Sie SINIX verlassen?

Am besten lernen Sie SINIX kennen, wenn Sie selbst praktisch damit arbeiten. Setzen Sie sich deshalb möglichst gleich an eine Datensichtstation Ihres Computers und probieren Sie aus, was Ihnen in diesem Buch erklärt wird.

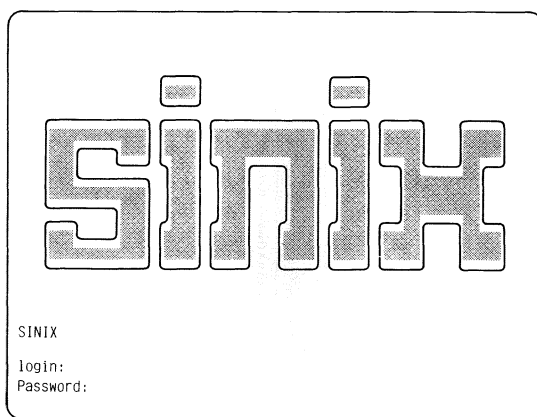


Bild 1-1 Begrüßungsbildschirm

Wenn Sie länger als 10 Minuten nichts eingeben, so verdunkelt sich automatisch der Bildschirm. Diese Funktion ist im SINIX-System eingebaut und verhindert, daß der Bildschirm unnötig beansprucht wird, wenn Sie ihn nicht brauchen. Außerdem sind Sie dann von anderen Arbeiten nicht abgelenkt. Drücken Sie einfach die Taste `↵`, dann zeigt der Bildschirm wieder alles wie vorher. Den gleichen Effekt erzielen Sie mit jeder anderen Taste (z.B.: Leerzeichentaste oder `a`, `x`, `q`, `p` usw.).

In diesem Buch wird Ihnen sehr viel mit Beispielen erklärt, um Ihnen zu zeigen, wie etwas prinzipiell funktioniert oder anwendbar ist. In den Beispielen sind Kommandos verwendet, die Sie wahrscheinlich nicht kennen. Das macht aber nichts. Es sind einfache Kommandos, die Sie aus dem Zusammenhang heraus verstehen lernen. Wenn Sie sich nicht abgelenkt fühlen, können Sie im Zweifelsfall auch im Buch "SINIX-Kommandos" Details zu den Kommandos nachschlagen.

Sollten Sie sich im Laufe dieser Beispielsitzung "verirren" und in eine Situation kommen, in der Sie nicht mehr weiter wissen, drücken Sie die Taste `END`. Damit wird der Begrüßungsbildschirm wieder ausgegeben und Sie können wieder neu beginnen.

Wie können Sie sich an SINIX anschließen?

Wenn Sie mit der Datensichtstation arbeiten wollen, müssen Sie diese als erstes einschalten (Schalter links unten. Leuchtanzeige muß leuchten). Warten Sie einen Augenblick. Wenn Sie jetzt nicht gleich den Begrüßungsbildschirm, sondern eine blinkende Schreibmarke sehen, drücken Sie bitte die Taste `END`. Anschließend bekommen Sie dann den Begrüßungsbildschirm ausgegeben.

Jetzt müssen Sie sich bei SINIX anmelden. Diesen Vorgang nennt man: Login. Dazu benötigen Sie eine *Benutzerkennung*. Haben Sie noch keine, dann lassen Sie sich von Ihrem Systemverwalter eine einrichten und auch ein *Kennwort* geben, oder Sie benutzen einfach die Standard-Benutzerkennung: *gast*.

Die Benutzerkennung: *gast* ist normalerweise durch das Kennwort: *sie-mens* geschützt und steht *jedem* Benutzer des SINIX-Systems zur Verfügung. Sicher wollen Sie aber bald eine eigene Benutzerkennung. Verwenden Sie bitte deshalb die Benutzerkennung *gast* nur zum *Kennenlernen* des SINIX-Systems oder für gelegentliche Testarbeiten, bei denen die Sicherheit Ihrer Daten keine Rolle spielt.

Tippen Sie jetzt die Benutzerkennung ein. Sie wird automatisch ab der Position der blinkenden Schreibmarke am Bildschirm angezeigt. Schließen Sie Ihre Eingabe ab, durch Drücken der Taste ☐.

Falls SINIX Sie jetzt nicht auffordert ein Kennwort einzugeben (Password:), sind Sie jetzt mit SINIX verbunden. Ansonsten geben Sie Ihr Kennwort ein. Es wird am Bildschirm nicht angezeigt.

Am Bildschirm erscheint das Zeichen \$.

Beachten Sie: SINIX arbeitet vorwiegend mit Kleinbuchstaben. Großbuchstaben sind möglich. Wenn Ihre Benutzerkennung oder Ihr Kennwort Groß- und Kleinbuchstaben enthält, sind sie auch genau so einzugeben.

Was passiert durch das Anschließen an SINIX?

Wenn Sie sich, wie oben beschrieben, bei SINIX richtig angemeldet haben, so wird am Bildschirm das Zeichen \$ ausgegeben. SINIX erwartet damit die Eingabe eines Kommandos.

Sie sind jetzt mit der sogenannten Shell verbunden. Die *Shell* ist ein Programm, das von nun an alle Ihre Eingaben empfängt, bewertet und entsprechende Aktionen entweder selbst ausführt oder anstößt. (Stören Sie sich bitte nicht an dem Wort "Shell". Es ist ein traditioneller, fester Begriff für einen Teil des Systems, der wie eine Muschel um den Systemkern liegt.)

Systemintern enthält SINIX zur Zeit für den Benutzer zwei etwas verschiedene Betriebssysteme, die man per Kommando wählen kann. Damit Sie bei der Benutzererkennung gleich auf den richtigen Weg kommen, geben Sie bitte, nachdem \$ am Bildschirm erscheint, erst dieses Kommando ein:

att 

Der Rechner meldet sich wieder mit dem Zeichen \$.

Was passiert, wenn Sie über die Tastatur etwas eingeben?

Erscheint am Bildschirm das Zeichen \$, können Sie dahinter eine Eingabe schreiben. Geben Sie z.B. über die Tastatur ein Kommando ein, wird Ihre Eingabe sofort am Bildschirm abgebildet. Optisch bekommen Sie so den Eindruck, Ihre Eingabe würde von der Tastatur aus direkt zum Bildschirm gelangen. In Wirklichkeit läuft Ihre Eingabe jedoch einen anderen Weg.

Die SINIX-Datensichtstation besitzt keine eigene "Intelligenz". Jedes Zeichen, das Sie über die Tastatur eingeben, geht sofort direkt an den Rechner. Von dort wird es lediglich als "echo" zur Datensichtstation zurückgeschickt und am Bildschirm zur Kontrolle Ihrer Eingabe ausgegeben.

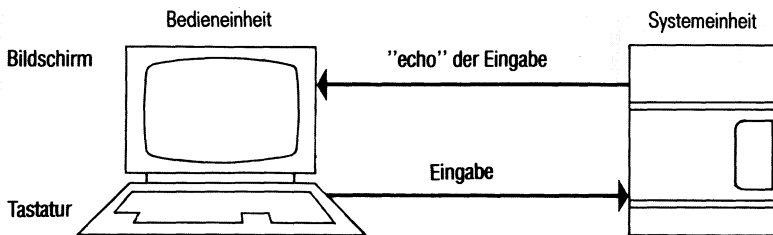


Bild 1-2 Weg einer Eingabe

Wenn Sie Ihre Eingabe abschließen (Taste ) , steht sie bereits schon im Rechner und wird nicht erst vom Bildschirm gelesen.

Weil eine Eingabe immer zuerst an den Rechner geht, bevor sie am Bildschirm angezeigt wird, kann es zu einer geringen zeitlichen Verzögerung zwischen Ihrer Eingabe und der Anzeige am Bildschirm kommen. (Wenn Sie während dieser Zeitspanne "nervös" werden und etwas eingeben, werden diese Eingaben aber auch vom Rechner empfangen und bewertet.)

Wie können Sie ein Kommando eingeben?

Nach dem Login erscheint am Bildschirm das Zeichen \$. Damit signalisiert Ihnen SINIX, daß es bereit ist, Eingaben zu empfangen. Hinter das \$-Zeichen können Sie jetzt ein Kommando schreiben. Die Schreibmarke steht schon richtig.

Probieren Sie einmal diese Eingabe:

date

Drücken Sie dann die Taste .

Mit dieser Taste signalisieren Sie dem Rechner im Klartext:
"Meine Kommando-Eingabe ist beendet".

Wie Sie sehen, wird Ihnen jetzt das aktuelle Datum ausgegeben (in englischer Schreibweise), z.B.

Wed Dec 16 14:59:44 MET 1987 (MET für mitteleuropäische Zeit)

Ein anderes einfaches Kommando ist *echo* (als Kommando). Damit können Sie z.B. einen Text eingeben, der auf den Bildschirm als "echo" zurückgeschrieben wird. Wenn sie folgendes eingeben:

echo Hallo

wird am Bildschirm in der nächsten Zeile ausgegeben:

Hallo

Merke:

Eine Kommandoeingabe muß mit der Taste  abgeschlossen werden. Anschließend wird die entsprechende Aktion ausgeführt.

Wie können Sie einen Eingabefehler korrigieren?

Machen Sie einmal eine "falsche" Eingabe. Geben Sie das Kommando `date` einmal so ein (mit `tt`):

`datte`

Als Antwort bekommen Sie jetzt ausgegeben:

`datte: not found`

Diese Eingabe versteht also SINIX nicht (Kommando nicht gefunden). Solange Sie eine Eingabe noch nicht mit der Taste  abgeschlossen haben, können Sie noch korrigieren.

Benutzen Sie dazu bitte die Taste  und *nicht* die Pfeil-Tasten.

Falls Sie die Bildschirmanzeige mit Hilfe der Tasten dieses Tastenblocks korrigieren wollten, ist die Anzeige am Bildschirm zwar optisch richtig, sie wird aber anders interpretiert, als Sie es auf den ersten Blick annehmen. Warum? Erinnern Sie sich. Jedes Zeichen, das Sie von der Tastatur aus eingeben, geht als erstes an den Rechner; anschließend bekommen Sie es als "echo" am Bildschirm angezeigt. Positionieren Sie die Schreibmarke mit einer Pfeiltaste, so wird dafür jeweils ein Steuerzeichen an den Rechner geschickt. Dort wird dieses Zeichen als Bestandteil der Eingabe interpretiert und nicht als Korrektur Ihrer Eingabe. Am Bildschirm erscheint Ihre Eingabe "optisch richtig", in Wirklichkeit besteht jedoch für den Rechner Ihre Eingabe aus den am Bildschirm angezeigten Zeichen plus dieser Steuerzeichen. Falls Sie Ihre Eingabe jetzt mit der Taste  abschließen würden, übergeben Sie an den Rechner eine andere Eingabe, als der Bildschirm anzeigt.

Die Pfeiltasten können Sie z.B. "gefahrlos" benutzen, wenn Sie mit dem Editor CED oder mit Menüs arbeiten.

Wie können Sie Ihre Benutzerkennung durch ein Kennwort schützen?

Normalerweise werden Sie von SINIX nach der Eingabe Ihrer Benutzerkennung aufgefordert, ein Kennwort einzugeben. Das ist der Fall, wenn der Systemverwalter Ihre Benutzerkennung durch ein Kennwort geschützt hat. Dieses Kennwort können Sie nach dem Login selber ändern. Dazu müssen Sie das Kommando `passwd` benutzen:

`passwd`

SINIX meldet sich mit:

Changing password for xxx	(xxx steht hier für eine Benutzerkennung)
Old password	(tippen Sie Ihr Kennwort, es wird nicht angezeigt)
New password	(neues Kennwort schreiben, es wird nicht angezeigt)
Retype new password	(nochmal neues Kennwort, es wird nicht angezeigt)

Weil ein Kennwort eine wichtige Sache ist, fordert Sie SINIX auf, um ganz sicher zu gehen, Ihr neues Kennwort nochmals einzutippen. Es wird weder das alte Kennwort noch das neue Kennwort am Bildschirm gezeigt.

Ab jetzt haben Sie zum Schutz Ihrer Benutzerkennung ein Kennwort Ihrer Wahl definiert. Nun können Sie sich beim Login nun nur noch mit dem neuen Kennwort bei SINIX anmelden.

Wie können Sie SINIX verlassen?

Das System verlassen Sie durch Drücken der Taste `[END]`. Dabei muß die aktuelle Zeile am Bildschirm leer sein. In der Zeile soll also nur das Zeichen \$ zu sehen sein und die Schreibmarke. Anschließend erscheint am Bildschirm wieder der Begrüßungsbildschirm.

Haben Sie damit keinen Erfolg und Sie erhalten in der selben Zeile ein weiteres Zeichen \$, so drücken Sie nochmals die Taste `[END]`.

2 Das Dateisystem

Dateien kann man sich als Behälter für codierte Information vorstellen. Jede Datei hat zur eindeutigen Identifikation einen Namen. Für den Dateiinhalt gibt es bei SINIX primär keine Vorschriften für eine bestimmte Strukturierung. Jede Datei besteht einfach aus einer Folge von codierten Zeichen. Die Zeichenfolge in der Datei, und was der Rechner damit macht, ist immer von dem Programm abhängig, das die Datei benutzt.

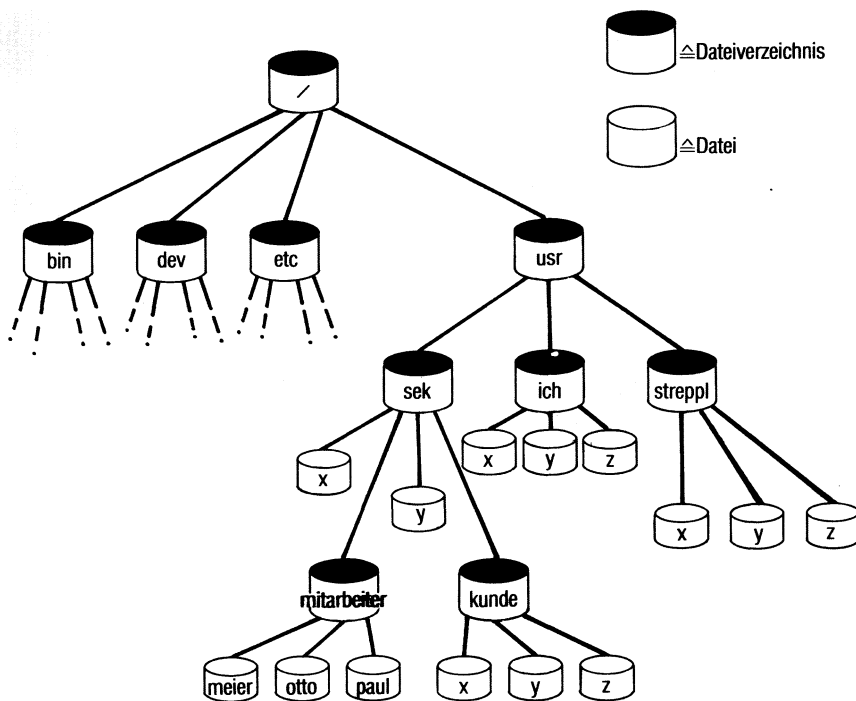


Bild 2-1 Beispiel für ein Dateisystem

2.1 Aufbau des Dateisystems

Wie kann nun SINIX mit diesen "einfachen" Dateien arbeiten? Da man beliebige Einträge in die Dateien machen kann, ist es auch möglich, Namen von Dateien, die logisch zusammengehören, in eine Datei zu schreiben. Der Name einer solchen Datei erhält vom System ein spezielles Kennzeichen. Daran erkennt es dann solche Dateien wieder, in denen nur organisatorische Angaben zu Dateien stehen.

So eine Datei nennt man Dateiverzeichnis (directory).

Mit solchen Mitteln ist das Betriebssystem SINIX so komfortabel aufgebaut, daß man sich mit Hilfe von Dateiverzeichnissen, die auch wieder Namen von weiteren Dateiverzeichnissen enthalten können, jederzeit ein verzweigtes, mehrstufiges und damit hierarchisches Dateisystem einrichten kann. Das gilt für jeden einzelnen Benutzer und für das SINIX-System selbst.

Die Dateien lassen sich so gruppieren, daß sie auf bestimmte Arbeitsgruppen, Firmenabteilungen, Ablagesysteme oder ähnliches abgestimmt sind.

Das Dateisystem hat ein oberstes Dateiverzeichnis, von dem aus es sich verzweigt. Es wird als Wurzel-Dateiverzeichnis bezeichnet (Root-Directory). Es hat als äußerst kurzen Namen nur das Zeichen / .

Das Dateisystem sieht aus wie das Wurzelwerk eines Baums. Es besteht aus Dateien und Dateiverzeichnissen. Von einem Dateiverzeichnis aus sind Verzweigungen zu Dateien oder auch zu weiteren Dateiverzeichnissen möglich. Die Dateiverzeichnisse sind Knotenpunkte des Dateisystems. Eine Datei ist immer das Ende eines Zweiges.

Nach dem Anmelden mit login, befindet sich ein Benutzer in einem Dateiverzeichnis unter usr (User = Benutzer), im Bild 2-1 z.B. bei *ich*.

Von dem Dateiverzeichnis *ich* aus, ist ein direkter Zugriff auf z.B. drei Dateien x, y und z möglich.

Mit den weiteren Dateiverzeichnissen im Bild, mit der gleichen Hierarchie-Stufe wie `usr`, sind Dateiverzeichnisse angedeutet, die SINIX als System-Dateiverzeichnisse benutzt:

- `bin` für allgemeine, oft benutzte Kommandos
- `dev` für sogenannte Geräte-Dateien (Verbindung zu Geräten)
- `etc` für Systemverwaltung

Außer diesen, gibt es noch viele weitere System-Dateiverzeichnisse, die Sie immer mehr interessieren werden, je mehr Sie vom SINIX-System kennen lernen.

Momentan ist wichtig zu erkennen, daß man sich mit Hilfe dieser Struktur des Dateisystems, beliebig benutzungsspezifische Datei-Untersysteme aufbauen kann, wobei auch Verknüpfungen möglich sind.

Theoretisch kann jeder Benutzer, von jedem Dateiverzeichnis aus, in jedes andere Dateiverzeichnis gelangen und auf die Dateien zugreifen. Damit aber geregelte Verhältnisse im Dateisystem herstellbar sind, bietet das System jedem Benutzer mehrere Möglichkeiten, den Zugriff auf seine Dateiverzeichnisse und seine Dateien zuzulassen oder zu sperren.

2.2 Pfadnamen für Dateien und Dateiverzeichnisse

Wie kann man nun von "seinem" Dateiverzeichnis aus in ein anderes gelangen? Ganz einfach. Nachdem jedes Dateiverzeichnis und jede Datei einen eindeutigen Namen hat, gibt man einfach alle Namen an, die den Weg zur Zieldatei beschreiben.

Diesen Weg nennt man *Pfad*.

Ein bestimmter Weg wird mit einem *Pfadnamen* angegeben.

Der Pfad, der zu einer Datei oder einem Dateiverzeichnis führt, ergibt sich automatisch durch die Lage im Dateisystem. Den Pfadnamen gibt man in entsprechenden SINIX-Kommandos an. Je nachdem, von welchem Standpunkt aus, d.h. von welchem Dateiverzeichnis aus man einen Pfadnamen angeben will, gibt es zwei Möglichkeiten:

Entweder vom Anfang des Dateisystems, also vom Wurzel-Dateiverzeichnis aus oder vom aktuellen Dateiverzeichnis aus. Das ist das, mit dem man gerade arbeitet.

Entsprechend unterscheidet man zwei verschiedene Arten von Pfaden:

- Absoluter Pfad
- Relativer Pfad

Wie sehen Pfadnamen aus?

Wie vieles in SINIX, ist auch die Schreibweise für Pfadnamen "kurz und bündig":

Der absolute Pfadname, z.B. im letzten Bild zu der Datei x, die im Dateiverzeichnis *ich* geführt wird, heißt:

`/usr/ich/x`

Ein *absoluter* Pfadname beginnt immer mit dem Schrägstrich. Der erste Schrägstrich ist der *Name* des Wurzel-Dateiverzeichnisses. Dann kommt der Name des nächsten Dateiverzeichnisses. Dann steht wieder ein Schrägstrich zur Trennung zum Namen des nächsten Dateiverzeichnisses.

Der zweite Schrägstrich und alle weiteren, dienen nur zur Abgrenzung der Namen. Das ist in SINIX so festgelegt. Rein äußerlich ergibt sich ein einheitliches Bild der Zeichen, obwohl der erste Schrägstrich etwas anderes bedeutet als alle weiteren.

Ein *relativer* Pfadname beschreibt immer den Weg, vom gerade aktuellen Dateiverzeichnis aus, zu einer Datei oder einem Dateiverzeichnis - vorwärts oder rückwärts im Dateisystem.

Angenommen, Sie arbeiten gerade unter dem Dateiverzeichnis *ich*, wieder auf das letzte Bild bezogen, und wollen in einem Kommando den Pfadnamen zum Dateiverzeichnis *usr* angeben. Der relative Pfadname heißt hierzu:

..

Mit .. gelangt man immer *ein* Dateiverzeichnis *zurück*.

Entsprechend führt ../.. zwei Dateiverzeichnisse zurück.

Vom Dateiverzeichnis *ich* aus, gelangen Sie zur Datei *y* im Dateiverzeichnis *kunde*, mit dem relativen Pfadnamen

../sek/kunde/y

Mit .. gibt man SINIX an: "gehe ein Dateiverzeichnis zurück". Die zwei Punkte führen erst einmal zurück zum Dateiverzeichnis *usr*. Die weiteren Angaben kennzeichnen wieder den Weg vorwärts.

Der relative Pfadname vom Dateiverzeichnis *usr* aus würde so aussehen:

sek/kunde/y

2.3 Wichtige Kommandos

Nach dem *login* befinden Sie sich in dem Dateiverzeichnis, das für Sie im System reserviert wurde (mit Systemverwaltungsfunktionen). Gleich nach dem Login können Sie das überprüfen mit dem Kommando

pwd 

Wenn Sie dieses Kommando eingeben, wird Ihnen der absolute Pfad zu *Ihrem* Dateiverzeichnis ausgegeben, z.B.

/usr/ich

Dateisystem

Mit dem Kommando `pwd` können Sie sich *immer* orientieren, in welchem Dateiverzeichnis Sie gerade sind.

Wenn Sie in ein anderes Dateiverzeichnis wechseln wollen, benutzen Sie das Kommando `cd` und geben zusätzlich den entsprechenden Pfadnamen an, z.B.

`cd ../..` 

Damit landen Sie z.B. im Wurzel-Dateiverzeichnis.

Auch das Kommando `cd` ohne weitere Angaben ist wichtig. Es bringt Sie immer wohlbehalten, von irgendeinem Dateiverzeichnis aus, in *Ihr* Dateiverzeichnis zurück!

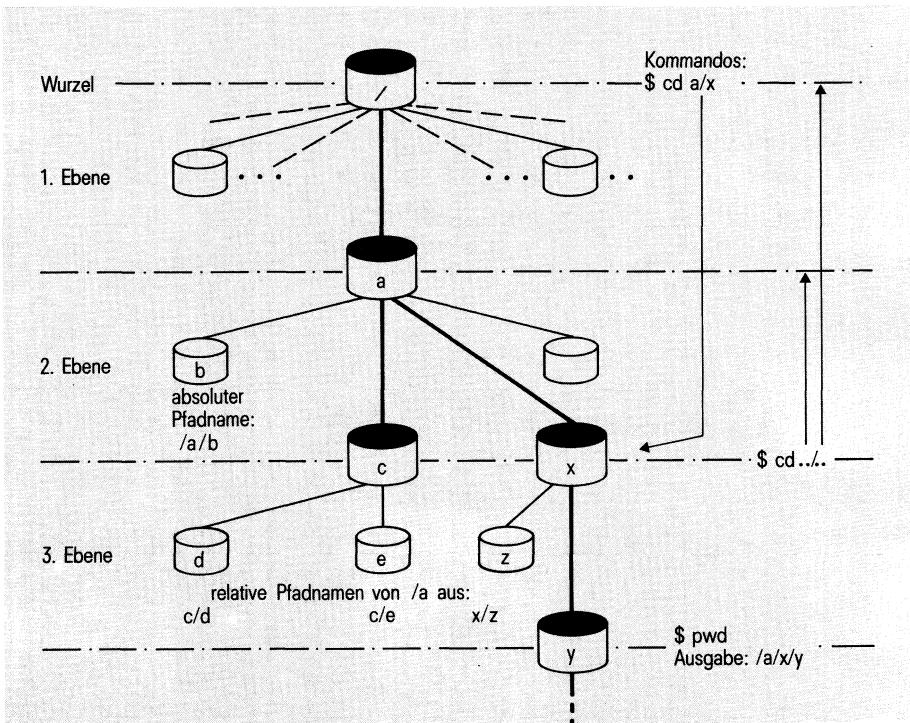


Bild 2-2 Prinzip des Dateisystems

2.4 Was erlaubt die Baumstruktur?

- An einem Mehrplatzsystem können mehrere Benutzer leicht auf die selbe Datei zugreifen. Das ist z.B. für Benutzer einer Arbeitsgruppe, die an einem Arbeitsprojekt arbeiten, sehr hilfreich.
Mit Mitteln zum Einstellen der Zugriffsrechte, kann verhindert werden, daß zur gleichen Zeit, von mehreren Benutzern, verändernd mit einer Datei gearbeitet wird.
- Theoretisch kann jeder Benutzer mit jeder Datei oder jedem Dateiverzeichnis des gesamten Dateisystems arbeiten. Dazu braucht er lediglich die Zugriffsberechtigung für die entsprechende Datei oder das entsprechende Dateiverzeichnis.
- Jeder Benutzer kann seine Dateien in einem Dateiverzeichnis führen oder zusätzlich weitere Dateiverzeichnisse einrichten. Dadurch kann man Dateien übersichtlich und sachorientiert organisieren.
- Ein Benutzer kann mehreren Dateien den gleichen Namen geben. Bedingung: Die einzelnen Dateien müssen in verschiedenen Dateiverzeichnissen eingetragen sein.
- Eine Datei läßt sich leicht einem anderen Dateiverzeichnis zuordnen. Dazu gibt es zwei Möglichkeiten.
Man kann eine Datei physikalisch (d.h. Name plus Inhalt) in ein anderes Dateiverzeichnis kopieren. Die Datei ist dann physikalisch mehrfach vorhanden. (Mit Kopier-Kommando cp)
Man kann aber auch nur den Namen einer Datei zusätzlich in ein anderes Dateiverzeichnis eintragen. Auf eine solche Datei bestehen dann entsprechend mehrere Verweise; die Datei ist physikalisch aber nur einmal vorhanden. (Betrifft "Link" mit Kommando ln)

2.5 Dateiverzeichnisse

Was kennzeichnet ein Dateiverzeichnis?

Die Dateiverzeichnisse braucht man, um ein verzweigtes Dateisystem aufbauen zu können. Ein Dateiverzeichnis kann nur zwei Informationsarten enthalten:

- Namen von Dateien plus Indexnummern
- Namen von Dateiverzeichnissen plus Indexnummern

Jeder neue Name wird vom System automatisch in das aktuelle Dateiverzeichnis eingetragen, wenn eine neue Datei oder ein neues Dateiverzeichnis erzeugt wird. Die Indexnummer wird vom System gegeben. Wenn Sie eine Datei löschen, werden auch die Einträge im Dateiverzeichnis automatisch gelöscht.

Die "Indexnummer" ist einige Seiten weiter, erklärt.

Wie kann man ein Dateiverzeichnis erzeugen?

Mit dem Kommando `mkdir` kann jeder Benutzer ein Dateiverzeichnis erzeugen. Ein neu eingerichtetes Dateiverzeichnis enthält standardmäßig zwei Einträge:

- Den Namen des übergeordneten Dateiverzeichnisses. Er wird durch zwei aufeinanderfolgende Punkte dargestellt (`..`).
- Den eigenen Namen. Er wird durch einen Punkt dargestellt (`.`).

Mit den Zeichen: `..` und `.` kann man arbeiten. Wie? Wollen Sie z.B. aus dem Dateiverzeichnis, in dem Sie sich gerade befinden, in das übergeordnete Dateiverzeichnis wechseln, können Sie das mit dieser Eingabe erreichen:

```
cd ..
```

Das Zeichen: `..` bezeichnet den Namen des übergeordneten Dateiverzeichnisses.

Das Zeichen: `.` kann man verwenden, wenn man z.B. den Inhalt eines anderen Dateiverzeichnisses in das aktuelle Dateiverzeichnis kopieren will. An dieser Stelle soll nicht weiter darauf eingegangen werden.

Wie kann man ein Dateiverzeichnis löschen?

Löschen kann ein Dateiverzeichnis nur sein Eigentümer. Dazu muß er das Kommando `rmdir` benutzen.

Hängen an einem Dateiverzeichnis beschriebene Dateien, müssen zuerst die Dateien gelöscht werden. Auch solange noch weitere Dateiverzeichnisse an ihm hängen, kann es ebenfalls nicht gelöscht werden. Damit wird verhindert, daß man z.B. unbeabsichtigt einen Zweig "abknipst".

Bezeichnungen für Dateiverzeichnisse

Für das Arbeiten mit SINIX, sind zur Orientierung, vier Dateiverzeichnisse besonders wichtig. Sie heißen:

- Login-Dateiverzeichnis
- Home-Dateiverzeichnis
- Aktuelles-Dateiverzeichnis
- Root-Dateiverzeichnis

Was bedeuten diese Bezeichnungen?

Als **Login-Dateiverzeichnis** wird das Dateiverzeichnis bezeichnet, in dem Sie sich automatisch nach dem Login befinden. Für jeden Benutzer definiert der Systemverwalter in der `/etc/passwd`-Datei ein Login-Dateiverzeichnis.

Als **Home-Dateiverzeichnis** wird das Dateiverzeichnis bezeichnet, das mit der Shell-Variable: `HOME` definiert ist. In dieses Dateiverzeichnis werden Sie automatisch gesetzt, wenn Sie das Kommando `cd` ohne weitere Angabe eingeben. (Solange Sie die Variable `HOME` nicht verändern, ist das Login- und das `HOME`-Dateiverzeichnis das selbe.)

Als **Aktuelles-Dateiverzeichnis** bezeichnet man das Dateiverzeichnis, in dem Sie sich jeweils aktuell befinden.

Als **Root-Dateiverzeichnis** (Wurzel-Dateiverzeichnis) wird das Dateiverzeichnis bezeichnet, von dem aus die Baumstruktur des Dateisystems beginnt. Es hat den Namen: `/`

2.6 Dateien

Was kennzeichnet eine Datei?

Eine Datei kann man als einen "Behälter" betrachten, in den man beliebige Einträge machen kann. Einträge können sein:

- Texte
- Shell-Prozeduren
- Programmcodes
- ausführbare Programme

Dateiinhalte speichert das Betriebssystem sequentiell ab. Solange man über die Shell mit Dateien arbeitet, kann man auf Dateiinhalte nur "ganz oder garnicht" zugreifen. Dateien und Dateiverzeichnisse kann man vor unberechtigtem Zugriff schützen (siehe Abschnitt: Zugriffsschutz für Dateien und Dateiverzeichnisse).

Wie kann man eine Datei erzeugen?

Es ist sehr einfach eine Datei zu erzeugen. Entweder man benutzt dazu einen Editor oder das Zeichen `>`. Es gibt kein spezielles Kommando, um eine Datei zu erzeugen. Jeder Dateiname wird in einem Dateiverzeichnis eingetragen. Er wird immer in dem Dateiverzeichnis eingetragen, in dem man sich gerade befindet, wenn man die Datei erstellt.

Wenn Sie z.B. mit dem CED-Editor eine Datei erstellen wollen, um anschließend gleich etwas in sie einzugeben, geben Sie folgendes ein:

```
ced xyz ↵
```

Jetzt wird der CED-Editor aufgerufen und automatisch eine Datei mit dem Namen `xyz` erzeugt. Sie können jetzt etwas in sie eintragen. (Mit dem Abschluß des CED wird die Datei erst gesichert. Wurde mit dem CED nichts eingetragen, so wird die Datei nicht angelegt.)

Durch folgende Eingabe erzeugen Sie mit dem Zeichen `>` eine Datei:

```
> xyz ↵
```

Jetzt haben Sie eine noch leere Datei mit dem Namen `xyz` erzeugt. Allerdings sollten Sie eine Datei nur dann erzeugen, wenn Sie sie wirklich "brauchen", d.h. wenn Sie etwas in Sie hineinschreiben wollen.

Das ist z.B. der Fall, wenn Sie eine Ausgabe anstatt auf die Standard-Ausgabe (z.B. den Bildschirm der Datensichtstation) in eine Datei schreiben wollen. Das können Sie mit dieser Eingabe erreichen:

echo hallo > xyz ↵

Die Ausgabe *hallo* des Kommandos *echo* wird jetzt automatisch in die Datei *xyz* geschrieben. Existierte die Datei *xyz* bereits, so wird sie überschrieben. Gab es die Datei *xyz* noch nicht, wird sie automatisch erzeugt (siehe auch: Umleiten der Standard-Ein-Ausgabe).

Ein Dateiname wird immer automatisch in das Dateiverzeichnis eingetragen, in dem man sich gerade befindet.

Noch ein Beispiel. Erzeugt man mit dem Kommando *cat* eine Datei, geht das wie folgt:

cat > abcd ↵
hallo ↵
END

Anschließend meldet sich die Shell mit dem Zeichen *\$* zurück. Man hat jetzt eine Datei mit dem Namen *abcd* erzeugt, die den Inhalt *hallo* hat. Existierte die Datei *abcd* bereits, wurde ihr Inhalt überschrieben.

Wie kann man eine Datei löschen?

Eine Datei kann man mit dem Kommando *rm* löschen.

Ein Beispiel. Folgende Eingabe löscht die Datei: *abcd*.

rm abcd ↵

Beachten Sie: Gibt man alleine den Dateinamen an, so bezieht man sich nur auf das *aktuelle* Dateiverzeichnis! Ansonsten muß man einen Pfadnamen angeben. (Das heißt, Sie können vom aktuellen Dateiverzeichnis aus an einem fernerem Punkt ein Dateiverzeichnis oder eine Datei lesen, erzeugen oder löschen - wenn Sie dazu die Rechte haben. Der Pfadname ist sozusagen der lange Arm.)

2.6.1 Dateien für Geräte

Jedes Gerät ist mit SINIX über eine Datei verbunden. Diese Dateien sind alle im Dateiverzeichnis: /dev eingetragen (vgl. Bild 2-3).

Dateien für Geräte kann nur der Systemverwalter erstellen oder löschen (Kommando `mknod`). Solange Sie sich jedoch nicht auf der Programmier-ebene befinden, sollten Sie Disketten oder Drucker nur über folgende Kommandos ansprechen (siehe Kapitel Kommandos):

`pr`, `lpr` oder `tar`

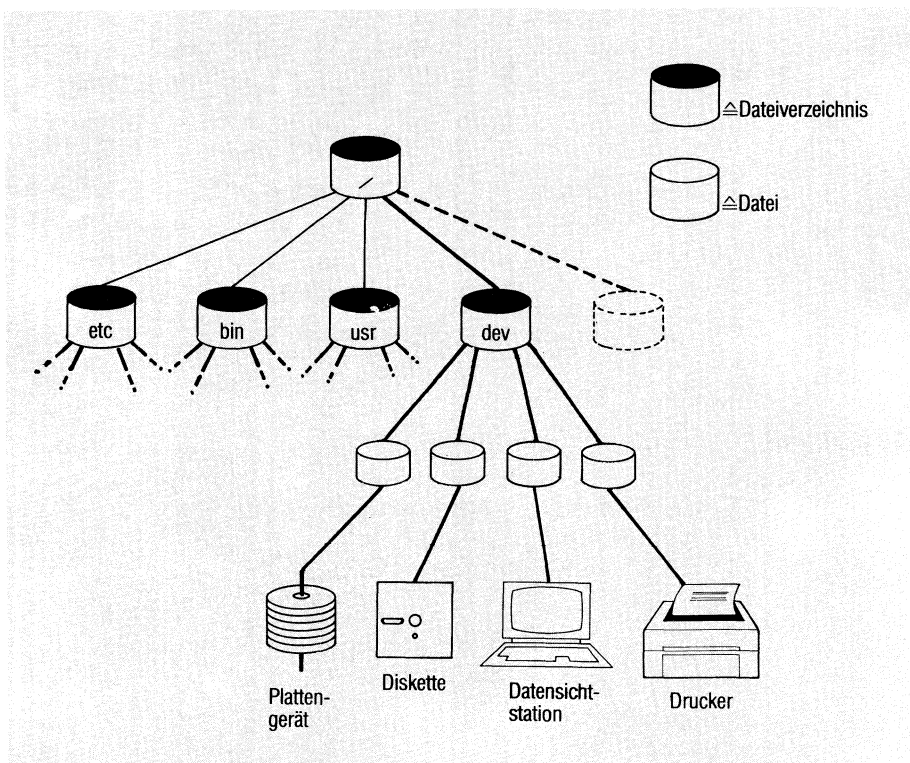


Bild 2-3 Anschluß von Geräten an SINIX

2.6.2 Sonderzeichen für Dateinamen

Es gibt Sonderzeichen, mit denen man sich das Arbeiten mit Datei-*Namen* erleichtern kann:

* ? [] [!]

Sie haben folgende Bedeutung:

* *Zeichenfolge*, ein oder mehrere beliebige Zeichen

? *ein* beliebiges Zeichen

[abc] *ein* Zeichen aus der angegebenen *Menge*, abc als Beispiel

[!abc] *nicht ein* Zeichen aus der angegebenen *Menge*, abc als Beispiel

[a-z] *von-bis ein* Zeichen, a bis z als Beispiel

Um sie optimal auszunutzen, können Sie selbst bereits bei der Definition von Dateinamen etwas dazu tun.

Ein Beispiel: Angenommen, Sie wollen ein längeres Dokument erfassen, z.B. ein Buch. Logisch kann man es in mehrere Teile untergliedern, z.B. in Kapitel. Physikalisch muß man es eventuell unterteilen, wenn ein Editor mit sehr großen Dateien nicht umgehen kann. Sie sollten dann Ihr Dokument nicht in eine einzige Datei schreiben, sondern es in mehrere Dateien aufteilen und ihnen z.B. solche Dateinamen geben:

kap.1

kap.2

kap.3

kap.4

Wie Sie sehen, sind die Dateinamen nach einem Schema aufgebaut. Das bringt zwei Vorteile:

Einmal wird die ganze Sache übersichtlicher und außerdem kann man dann besser mit Sonderzeichen arbeiten.

Mit folgendem Kommando könnte man alle genannten Dateien nacheinander ausgeben. Der Eingabeabschluß mit  ist folgend nicht mehr extra angegeben.

cat kap.1 kap.2 kap.3 kap.4

Das ist aber die umständlichste Art.

Mit Sonderzeichen geht es einfacher und schneller, wenn Sie das Kommando `cat` so eingeben:

```
cat kap*
```

Damit werden die Inhalte aller Dateien ausgegeben, deren Dateinamen mit `kap` beginnen und mit beliebig vielen anderen Zeichen enden. Das wird durch das Sonderzeichen: `*` erreicht.

Das Sonderzeichen `*` kann man auf jede Position eines Dateinamens setzen und innerhalb einer Eingabe auch mehrfach angeben. Beachten Sie aber, daß `*` beliebig viele Zeichen ersetzt!

```
cat k*p*
```

Dieses Kommando würde auch alle Dateien `kap...` ausgeben, ebenso aber auch zusätzlich z.B. eine Datei, die mit dem Namen *kaufpreis* eingetragen wäre.

Sollten Sie unsicher sein, so können Sie sich vorher Klarheit verschaffen. Geben Sie erst das Kommando `ls` (Dateinamen des aktuellen Dateiverzeichnisses auflisten) oder das Kommando `ls k*p*`, das dann genau die betreffenden Dateinamen auflistet.

Wird für Dateinamen nur das Zeichen: `*` alleine angegeben, so werden dafür *alle* Dateinamen des aktuellen Dateiverzeichnisses eingesetzt!

Beachten Sie dabei immer, daß die Eingabe `rm *` alle Dateien Ihres aktuellen Dateiverzeichnisses löscht - ob Sie wollen oder nicht. Eine Ausnahme bilden Dateien, deren Name mit *einem Punkt beginnt*. Das ist in SINIX so festgelegt.

Es gibt also (wie so oft) auch für Sonderzeichen eine Ausnahme. (Nicht zuletzt auch deswegen, weil der eigene Name jedes Dateiverzeichnisses mit einem Punkt beginnt.) Wollen Sie auch alle diese Dateinamen ansprechen, müssen Sie auch den Punkt angeben. Den führenden Punkt kann man nicht durch ein Sonderzeichen ersetzen.

Deshalb bekommt man, z.B. bei der Eingabe:

```
echo *
```

alle die Dateinamen aufgelistet, die nicht mit einem Punkt beginnen.

Wenn man *nur* die Dateien aufgelistet haben will, deren Name mit einem Punkt beginnt, kann man folgende Eingabe machen:

echo .*

Mit der Eingabe von

ls

erhalten Sie alle Einträge des aktuellen Dateiverzeichnisses.

Mit der Eingabe von

ls *

erhalten Sie alle Einträge des aktuellen Dateiverzeichnisses und zusätzlich die Einträge der Unter-Dateiverzeichnisse.

Mit der Eingabe von

ls .*

erhalten Sie alle Einträge des aktuellen Dateiverzeichnisses und zusätzlich die Einträge des übergeordneten Dateiverzeichnisses (dessen Name im aktuellen unter .. eingetragen ist).

Ein weiteres, jetzt wieder einfacheres Beispiel zu * : Das Kommando

ls *.c

listet alle Einträge des aktuellen Dateiverzeichnisses auf, deren Namen mit dem Zeichen: .c *enden*.

Mit den Sonderzeichen: [] und [!] lassen sich Zeichenfolgen "von-bis" zusammenfassen. So listet z.B. die Eingabe:

ls [a-k]*

alle Einträge des aktuellen Dateiverzeichnisses auf, die mit einem der Buchstaben a bis k *beginnen*:

ls [!a-k]*

listet alle Einträge des aktuellen Dateiverzeichnisses auf, die mit einem der Buchstaben a bis k *nicht beginnen*. Die Negierung wird erreicht durch das Sonderzeichen: !.

Mit der Eingabe:

ls [ak]*

listen Sie nur Einträge auf, die mit a *oder* k beginnen. Mit vorgestelltem !, nur Einträge, die nicht mit a oder k beginnen.

Dateisystem

Mit dem Sonderzeichen: ? kann man ein beliebiges *einzelnes* Zeichen in einer Eingabe ersetzen. Die Eingabe:

cat kap.?

gibt die Inhalte aller Dateien aus, deren Namen mit kap. beginnen und denen *ein* beliebiges Zeichen folgt.

Um ein Gefühl für den Umgang mit diesen Sonderzeichen zu bekommen, sollten Sie sie ausprobieren. Das Kommando echo gibt die Einträge am Bildschirm nebeneinander aus, durch Leerzeichen getrennt; das Kommando ls gibt sie untereinander als Liste aus. (Das Kommando echo verwendet man, um sich schnell einen Überblick zu verschaffen, das Kommando ls, da es in vielen Varianten möglich ist, um detailliertere Informationen zu erhalten.)

Ist im Dateiverzeichnis kein Abbild der Kommando-Argumente eingetragen, so wird beim Kommando ls die Meldung *not found* ausgegeben, bei echo das Argument selbst.

Angenommen, im aktuellen Dateiverzeichnis sind folgende Dateien eingetragen:

kap1 kap2 kap3 ed.doc man.doc

Mit folgenden Eingaben kann man mit Sonderzeichen untenstehende Ausgaben erreichen:

Eingabe:	Ausgabe:
echo *	ed.doc kap1 kap2 kap3 man.doc
echo *.doc	ed.doc man.doc
echo kap?	kap1 kap2 kap3
echo kap[2-3]	kap2 kap3
echo kap[23]	kap2 kap3
echo ?? .doc	ed.doc
echo ????	kap1 kap2 kap3
echo ??	?? (kein Name mit zwei Zeichen vorhanden)
ls ??	?? not found
ls *.doc	ed.doc man.doc (ls listet untereinander auf)

Wenn Sie die gleichen Eingaben mit dem Kommando cat machen, bekommen Sie die Inhalte der Dateien ausgegeben.

Entwerten von Sonderzeichen

Neben den Sonderzeichen: * [] [!] und ? haben für die Shell auch noch folgende Zeichen eine spezielle Bedeutung:

< | & && () und { }.

Will man eines dieser Zeichen, z.B. in einem Text, als "normales" Zeichen angeben, d.h. ohne daß es die Shell als Sonderzeichen interpretiert, muß man es entwerten. Das macht man mit dem Zeichen: \

Zum Beispiel schreibt

```
echo \?
```

ein Fragezeichen auf den Bildschirm der Datensichtstation. Das Zeichen \ hat die spezielle Bedeutung des Fragezeichens für die Shell entwertet (siehe auch Abschnitt: Apostrophier-Mechanismus im Kapitel Shell).

Ebenso schreibt

```
echo \*
```

den * auf den Bildschirm!

2.7 Indexnummern für Dateien und Dateiverzeichnisse

Für Zugriffs- und Verwaltungszwecke im Dateisystem, führt SINIX eine Liste, in der alle Dateien eingetragen sind. An den Einträgen ist die Dateiart Dateiverzeichnis oder Datei erkennbar, Zugriffsrechte, der Dateiname usw. Insgesamt sind 64 Byte für jeden sogenannten "Indexeintrag" (inode) vorhanden.

Jeder Indexeintrag hat eine Indexnummer, unter der er vom Betriebssystem verwaltet wird. Wie die Namen von Dateiverzeichnissen und Dateien, stehen auch die Indexnummern in den jeweiligen Dateiverzeichnissen (siehe Bild 2-4). Teile dieser Indexeinträge können Sie sich ausgeben lassen, mit dem Kommando `ls -l`.

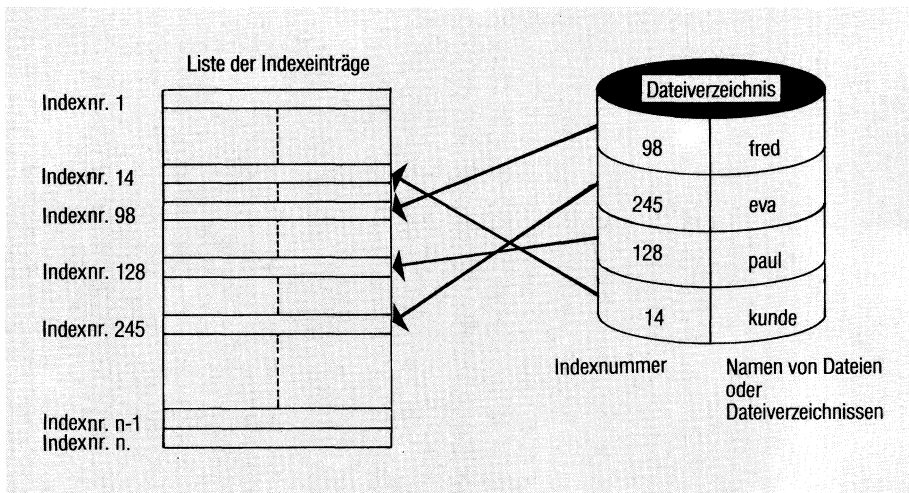


Bild 2-4 Zuordnung von Indexnummer zu Indexeintrag-Liste

Identifikationszeichen für Dateiarten

Zum Erkennen der Dateiart dienen die Identifikationszeichen im Indexeintrag. Je nachdem, ob man eine Datei oder ein Dateiverzeichnis erstellt, wird dort automatisch ein entsprechender Eintrag gemacht. Folgende Einträge sind möglich:

- d für Dateiverzeichnis
- für Datei
- b für Datei für ein Gerät, das blockweise arbeitet
- c für Datei für ein Gerät, das zeichenweise arbeitet

Ein vollständiger Indexeintrag enthält folgende Informationen:

- Ein Identifikations-Zeichen, ob es sich um eine Dateiverzeichnis oder eine Datei handelt.
- Schutzbits (sind im nächsten Abschnitt erklärt)
- Anzahl der existierenden Verweise auf die Datei oder das Dateiverzeichnis, für die dieser Indexeintrag gilt (Erklärung: siehe unten)
- Benutzernummer
- Gruppennummer
- Größe der Datei oder des Dateiverzeichnisses in Byte
- Plattenblockadresse
- Die Zeit, wann das letzte mal zugegriffen wurde
- Die Zeit, wann das letzte mal modifiziert wurde
- Zeitpunkt der Erstellung des Indexeintrags

Anzahl der Verweise für ein Dateiverzeichnis

Bei diesem Eintrag wird folgendes eingetragen: Wieviele andere Dateiverzeichnisse direkt an ihm hängen plus 2. Warum: plus 2? Weil ein Dateiverzeichnis immer mindestens zwei Einträge enthält:

Seinen eigenen Namen (Abkürzung: .) und den Namen des übergeordneten Dateiverzeichnisses, in dem es selbst eingetragen ist (Abkürzung: ..).

Anzahl der Verweise für eine Datei

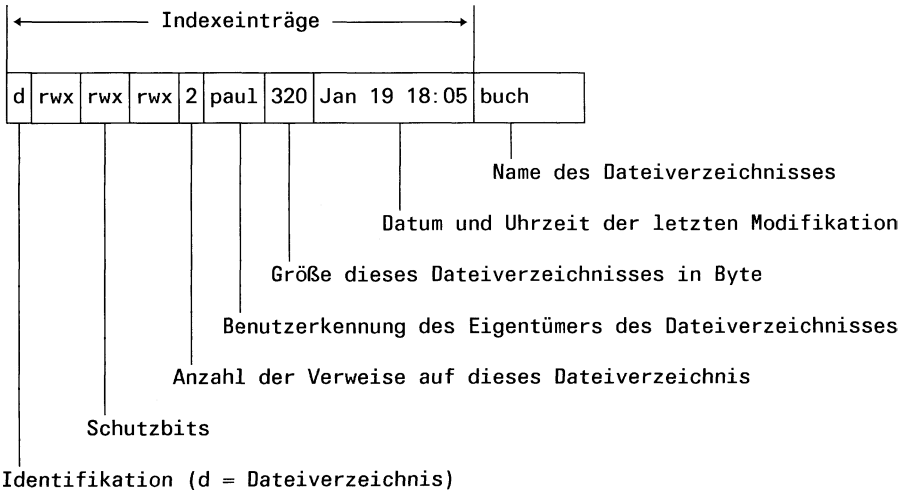
Bei diesem Eintrag wird eingetragen, wie oft die Datei in irgendeinem Dateiverzeichnis eingetragen ist. Auf eine Datei gibt es mehrere Verweise, wenn es mehrere "Links" auf sie gibt (vgl. Kommando ln).

Indexeinträge kann man mit dem Kommando ls lesen. Folgend ein paar Beispiele, wobei das Kommando ls mit einem Kommando-Argument, dem Schalter -l eingegeben werden muß.

Die Ausgaben für Dateien und Dateiverzeichnisse sind unterschiedlich.

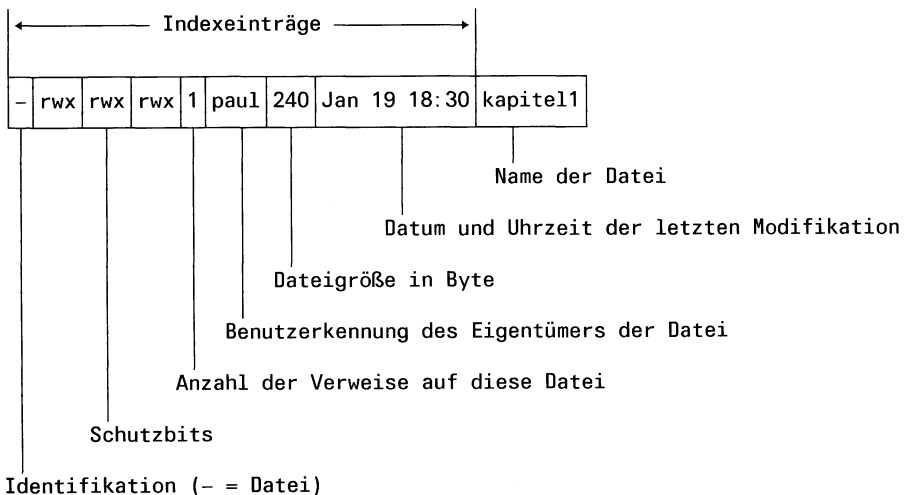
Beispiel für ein *Dateiverzeichnis* mit dem Namen *buch*.

Ausgabe der Indexeinträge mit dem Kommando `ls -l buch`



Beispiel für eine *Datei* mit dem Namen *kapitel1*.

Ausgabe der Indexeinträge mit dem Kommando `ls -l kapitel1`



Beispiel für Indexeinträge einer Datei, über die ein Gerät definiert ist.
Ausgabe der Indexeinträge mit dem Kommando `ls -l /dev/tty01`

Indexeinträge										
c	rw-	-w-	-w-	1	root	8,	1	Aug 21	16:00	/dev/tty01
										Dateiname im Dateiverzeichnis: /dev
										Datum und Uhrzeit der letzten Modifikation
										Gerätenummer (Minornummer)
										Gerätekategorie (Majornummer)
										Benutzerkennung des Eigentümers
										Anzahl der Verweise auf das Gerät
										Schutzbits
Identifikation (c = Datei für zeichenweise arbeitendes Gerät)										

2.8 Zugriffsschutz für Dateien und Dateiverzeichnisse

Warum sind bei SINIX Dateien und Dateiverzeichnisse geschützt?

Theoretisch könnte man auf einen Zugriffsschutz für Dateien und Dateiverzeichnisse verzichten. Jeder Benutzer könnte dann auf alle Dateien und Dateiverzeichnisse des Dateisystems zugreifen und mit ihnen arbeiten. Die Folge wäre wahrscheinlich bald ein Chaos. Denn jeder Benutzer könnte - versehentlich oder absichtlich - unkontrolliert das gesamte Dateisystem verändern. Jeder Benutzer könnte die Dateien anderer Benutzer lesen und verändern. Um das zu verhindern, gibt es Schutzmechanismen. Damit kann man Dateien und Dateiverzeichnisse vor unbefugtem Zugriff schützen.

2.8.1 Wodurch sind bei SINIX Dateien und Dateiverzeichnisse geschützt?

- Jeder Benutzer hat eine individuelle Benutzerkennung
- Jede Benutzerkennung hat ein Kennwort (password)
- Jedes Dateiverzeichnis und jede Datei hat mehrstufigen Zugriffsschutz

Benutzerkennung und Kennwort

Jeder Benutzer, der sich ans System anschließen will (Login), braucht eine Benutzerkennung. Die muß für ihn der Systemverwalter definieren. Außerdem erhält jeder Benutzer vom Systemverwalter auch ein Kennwort für die Benutzerkennung (als Eintrag in die Datei `/etc/passwd`). Der Benutzer kann dieses Kennwort auch selbst definieren oder verändern (Kommando `passwd`). Das Kennwort (password) wird beim Login immer gleich nach der Benutzerkennung abgefragt und muß dann eingegeben werden.

Benutzergruppen

Mit Systemverwalter-Funktionen können auch mehrere Benutzer zu Gruppen zusammenfaßt werden. Der Systemverwalter vergibt dazu eine Gruppennummer (als Eintrag in die Datei `/etc/group`). Die Möglichkeit Gruppen zu bilden, ist eine Eigenheit von SINIX und dient zur Abgrenzung von Zugriffsrechten gegenüber einem "Eigentümer" von Dateien und den sogenannten "Anderen".

Benutzerklassen

In SINIX werden bezüglich der Zugriffsrechte, die in jedem Dateiverzeichnis und in jeder Datei eingestellt werden können, diese drei Benutzerklassen unterschieden:

Eigentümer

Gruppe

Andere

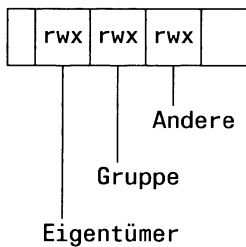
Mit dieser Einrichtung kann z.B. jeder Benutzer, der zu einer Gruppe gehört, seine Dateien oder Dateiverzeichnisse für die Mitglieder seiner Gruppe zugänglich machen und für andere nicht. Dabei gibt es viele Möglichkeiten für die Einstellung.

2.8.2 Schutzbits für Dateien und Dateiverzeichnisse

Sowohl jedem Dateiverzeichnis als auch jeder Datei, werden beim Erstellen Schutzbits automatisch mitgegeben. Diese Schutzbits kann *nur* der Eigentümer verändern und damit seine Dateien oder Dateiverzeichnisse schützen.

Es gibt insgesamt 9 Schutzbits.

Es gehören jeweils 3 Schutzbits zusammen und gelten genau für *eine* der 3 Benutzerklassen:



Für jede der 3 Benutzerklassen kann man folgende Berechtigungen festlegen:

r	für Leseberechtigung	(read)
w	für Schreibberechtigung	(write)
x	für Ausführberechtigung	(execute)

Die Berechtigungen werden für Dateien und Dateiverzeichnisse unterschiedlich interpretiert. Und zwar so:

	Datei	Dateiverzeichnis
read	lesen	Einträge lesen
write	schreiben	Einträge löschen/anlegen
execute	ausführen	durchlaufen/durchsuchen

Wie kann man Schutzbits setzen?

Schutzbits kann man mit dem Kommando `chmod` setzen (siehe auch Kommandobeschreibung). Der Systemverwalter darf die Schutzbits aller Dateien oder Dateiverzeichnisse verändern. Der Eigentümer darf nur die Schutzbits seiner *eigenen* Dateien oder Dateiverzeichnisse verändern. Auch wenn jemand aus der Gruppe oder Andere volle Zugriffsberechtigung auf eine Datei oder ein Dateiverzeichnis besitzt, kann er trotzdem die Schutzbits *nicht* verändern.

Die Schutzbits werden vom Betriebssystem als ein Teil des Indexeintrags abgespeichert. Man kann sie am Bildschirm nicht alleine ausgeben, sondern nur zusammen mit anderen Indexeinträgen (siehe: Kommando `ls`).

Für jede Datei und jedes Dateiverzeichnis gibt es eine *Standardeinstellung* der Schutzbits, die beim Erstellen mitgegeben wird. Die Standardeinstellung kann man mit dem Kommando umask verändern (siehe: Kommando umask im Kapitel Shell).

Die Standardeinstellung sieht so aus:

-	rw-	rw-	r--	

Datei

d	rwX	rwX	r-X	

Dateiverzeichnis

Wie man sieht, kann mit einer Datei standardmäßig der Eigentümer und die Gruppe: lesen und schreiben. Andere können nur: lesen.

Ein Dateiverzeichnis kann standardmäßig vom Eigentümer und der Gruppe durchlaufen werden. Beide dürfen Einträge lesen und löschen bzw. anlegen. Andere dürfen ein Dateiverzeichnis nur durchlaufen und Einträge lesen.

Schutzbits gelten nur für ihre Benutzerklasse

Die Schutzbits beziehen sich streng auf die Benutzerklassen:

Eigentümer,
Gruppe,
Andere.

Was heißt das? Hat z.B. der Eigentümer einer Datei selbst keine Zugriffsberechtigung für seine Datei, aber die Gruppe und Andere (Einstellung ----rwxrwx), dann darf er auch nicht mit ihr arbeiten.

Der Eigentümer kann aber jederzeit die Schutzbits seiner Dateien ändern und sich so wieder selbst Zugang zu seiner Datei verschaffen.

Haben z.B. nur die Anderen eine Zugriffsberechtigung auf eine Datei (Einstellung -----rwx), dann darf weder der Eigentümer noch ein Mitglied seiner Gruppe mit der Datei arbeiten.

Die Schutzbits für: Eigentümer, Gruppe und Andere schließen einander also aus. Ein Eigentümer einer Datei bekommt daher auch keine Zugriffsberechtigung über die Gruppenzugehörigkeit.

Beispiele dafür, wie man Schutzbits definieren kann

In den Beispielen sind die Schutzbits im Kommando `chmod` in der Oktal-Schreibweise angegeben; z.B. 111 110 100 als 764 für `rwX rw- r--`.

Beispiel 1

Eigentümer, Gruppe und Andere sollen volles Zugriffsrecht auf die Datei *text* erhalten.

Schutzbits setzen	<code>chmod 777 text</code>
Lesen der Änderung	<code>ls -l</code>
Ausgabe	<code>-rwxrwxrwx 1 paul 33 Jan 19 10:00 text</code>

Beispiel 2

Eigentümer und Gruppe sollen auf die Datei *text* volles Zugriffsrecht erhalten. Andere sollen aus ihr nur lesen können.

Schutzbits setzen	<code>chmod 774 text</code>
Lesen der Änderung	<code>ls -l</code>
Ausgabe	<code>-rwxrwxr-- 1 paul 33 Jan 19 10:00 text</code>

Beispiel 3

Nur der Eigentümer der Datei *text* soll Lese- und Schreibberechtigung haben.

Schutzbits setzen	<code>chmod 600 text</code>
Lesen der Änderung	<code>ls -l</code>
Ausgabe	<code>-rw----- 1 paul 33 Jan 19 10:00 text</code>

Beispiel 4

Nur der Eigentümer des Dateiverzeichnisses *buch* soll volles Zugriffsrecht haben.

Schutzbits setzen	<code>chmod 700 buch</code>
Lesen der Änderung	<code>ls -l</code>
Ausgabe	<code>-rwx----- 1 paul 33 Jan 19 10:00 buch</code>

Beispiel 5

Nur der Eigentümer des Dateiverzeichnisses *buch* soll volles Zugriffsrecht haben. Die Gruppe und Andere sollen das Dateiverzeichnis durchlaufen können.

Schutzbits setzen	<code>chmod 711 buch</code>
Lesen der Änderung	<code>ls -l</code>
Ausgabe	<code>drwx--x--x 1 paul 33 Jan 19 10:00 buch</code>

Beispiel 6

Der Eigentümer der Datei *test*, die eine Shell-Prozedur enthält, soll volles Zugriffsrecht haben. Gruppe und Andere sollen sie nur ausführen können.

Schutzbits setzen	<code>chmod 755 test</code>
Lesen der Änderung	<code>ls -l</code>
Ausgabe	<code>-rwxr-xr-x 1 paul 33 Jan 19 10:00 test</code>

Beim x-Bit muß man aufpassen

Das Schutzbit für die Ausführberechtigung (execute) wird für Dateien und Dateiverzeichnisse unterschiedlich interpretiert. Da ein Dateiverzeichnis nicht ausgeführt werden kann, wird das x-Bit so interpretiert, daß die entsprechende Benutzerklasse das Dateiverzeichnis durchlaufen darf.

Will man mit dem x-Bit eine Datei, die eine *Shell-Prozedur* enthält, als ausführbar kennzeichnen, muß man zusätzlich durch das r-Bit auch eine Leseerlaubnis vergeben. Das x-Bit alleine reicht dafür nicht, um eine Prozedurdatei für eine Benutzerklasse ausführbar zu machen.

Achtung

Ein nicht gesetztes x-Bit garantiert noch keinen Schutz gegen unbeberechtigtes Ausführen einer Prozedurdatei. Wieso? Eine Datei kann man auf zwei Arten ausführen:

- durch Angabe ihres Namens
- durch das Kommando sh

Will man eine Datei, die eine Prozedur enthält, durch Angabe ihres Namens ausführen, kann man das nur, wenn das entsprechende r-Bit und x-Bit gesetzt ist (siehe oben). Für das Kommando sh gilt dieser Mechanismus nicht.

Was macht das Kommando sh ? Damit ruft man die Shell auf, um eine Datei zu lesen und ihren Inhalt zu interpretieren. Enthält diese Datei eine Shell-Prozedur, führt die Shell diese auch aus. Das x-Bit (Ausführberechtigung) braucht dafür nicht gesetzt zu sein. Es reicht, wenn das r-Bit (Leseberechtigung) gesetzt ist. Was hat das zur Folge? Da jeder Benutzer das Kommando sh aufrufen darf, sollte man sich als Eigentümer einer Datei darüber im klaren sein, daß man mit Vergabe einer Leseberechtigung auch gleichzeitig eine Ausführberechtigung vergibt.

Will man dagegen eine Datei, die einen *ausführbaren Programmcode* enthält, als ausführbar kennzeichnen, reicht es, wenn man das x-Bit alleine setzt.

	Datei mit Shell-Prozedur als Inhalt	Datei mit ausführbarem Programmcode	Dateiverzeichnis (ausführen = durchlaufen)
wird ausführbar durch	x-Bit und r-Bit oder alleine r-Bit	x-Bit	x-Bit

Kontrollierter Dateizugriff über Programm durch das s-Bit

Das s-Bit bindet ein Programm an definierte numerische Identifikationen. Was bedeutet das? Für jeden Benutzer gibt es zwei numerische Identifikationen (siehe auch: Bild 2-5):

- Eine Benutzernummer (UID). Darüber ist jeder Benutzer im SINIX-System eindeutig identifiziert.
- Eine Gruppennummer (GID). Darüber ist jeder Benutzer im SINIX-System einer Gruppe zugeordnet.

Beide Identifikationen vergibt der Systemverwalter.

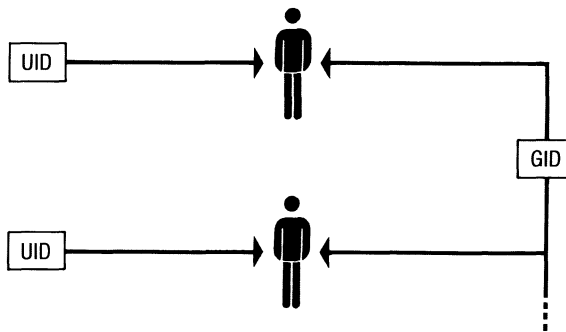


Bild 2-5 Zuordnung von UID und GID zu Benutzern

Startet ein Benutzer ein Programm, läuft es unter diesen numerischen Identifikationen (UID und GID) ab (vorausgesetzt natürlich, der Benutzer hat eine Ausführberechtigung für das entsprechende Programm). Diese Verlagerung der Zugriffswege kann man bewußt ausnutzen - es kann aber auch zu Problemen führen.

Dateisystem

Ein Beispiel. Angenommen, Sie haben als Eigentümer ein Programm, das auf eine Datei zugreift, die ebenfalls Ihnen gehört. Die Schutzbiteinstellung soll so sein (siehe auch: Bild 2-6):

- für das Programm: `-rwxr-xr-x`
- für die Datei: `-rwx-----`

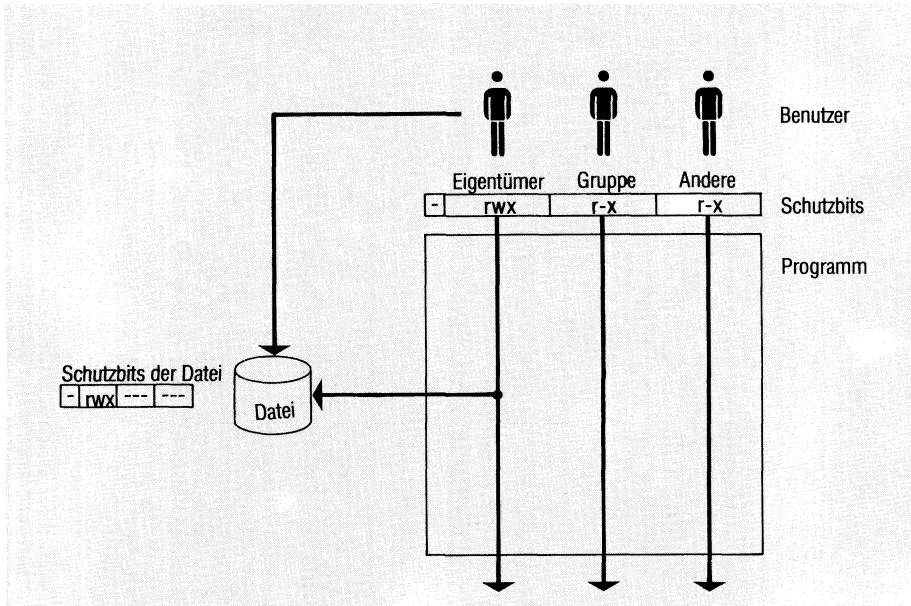


Bild 2-6 Dateizugriff über ein Programm

Damit können Sie als Eigentümer Ihre Datei und Ihr Programm lesen, schreiben und ausführen. Die Mitglieder Ihrer Gruppe und Andere können zwar Ihr Programm ausführen, aber nicht auf die Datei zugreifen.

Wie kann man den Benutzern: Gruppe und Anderen einen Zugriff auf die Datei erlauben? Erst einmal dadurch, daß Sie die Schutzbits für Ihre Datei so einstellen: `-rwxr-xr-x`. Damit könnten die Benutzer: Gruppe und Andere jedoch auch direkt auf die Datei zugreifen, ohne Ihr Programm zu benutzen. Der Zugriff auf die Datei wäre dann nicht mehr über Ihr Programm kontrollierbar.

Soll der Zugriff auf die Datei nur über Ihr Programm möglich sein, muß man ein s-Bit setzen. An Stelle des x-Bits kann ein s-Bit nur für eine UID oder GID gesetzt werden.

Was macht das s-Bit? Es hat zur Folge, daß ein Programm immer nur unter einer fest definierten *Nummer*, der UID oder GID abläuft - egal welcher Benutzer es aufruft.

Beispiel: Angenommen, Sie haben als Eigentümer folgende Schutzbits gesetzt:

- für das Programm: `-rwsr-xr-x`
- für die Datei: `-rwx-----`

Dann haben Sie für sich als Eigentümer das s-Bit gesetzt (siehe auch: Kommando `chmod`). Jetzt läuft das Programm (intern) immer unter Ihrer Benutzernummer (UID), egal ob es ein Mitglied Ihrer Gruppe oder Andere ausführen. Das *Programm* kann damit unabhängig davon, welcher Benutzer es ausführt, auch auf die Datei zugreifen.

Dateisystem

Noch ein Beispiel für das s-Bit. Angenommen, Sie haben als Eigentümer folgende Schutzbits gesetzt (siehe auch Bild 2-7):

- für das Programm: `-rwx--s--x`
- für die Datei1: `-rwxrw----`
- für die Datei2: `-rwx-----`

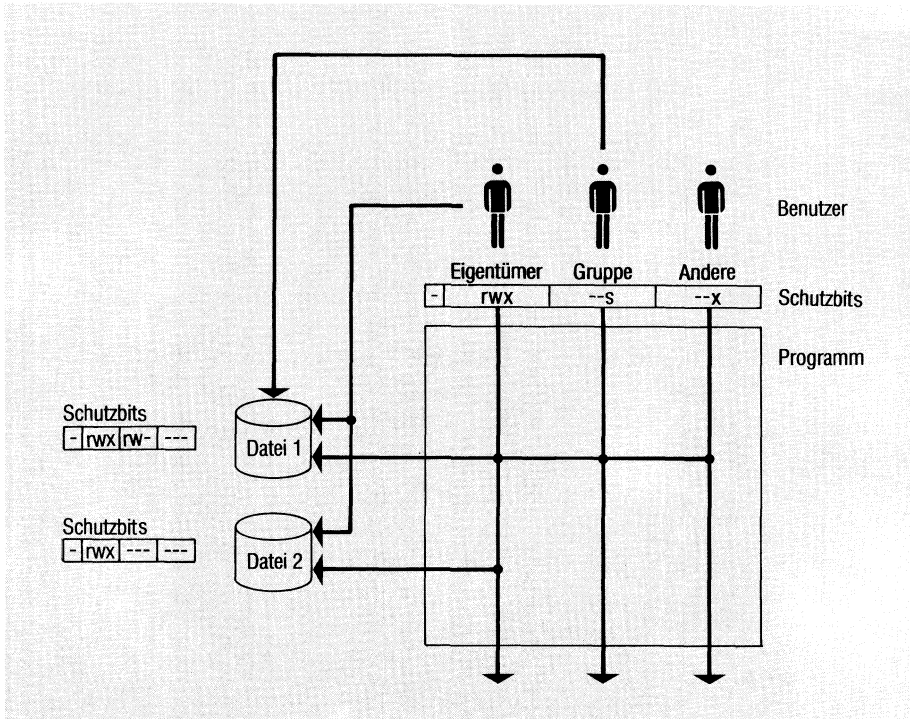


Bild 2-7 Dateizugriff über ein Programm mit s-Bit

Jetzt können alle, Sie als Eigentümer, die Gruppenmitglieder und Andere das Programm ausführen und darüber auf die Datei1 zugreifen. Das wird durch Setzen des s-Bits bei der Gruppe erreicht. Andere können auf die Datei1 nur über das Programm zugreifen; auf die Datei2 haben diese weder direkt, noch über das Programm einen Zugriff.

Sie als Eigentümer können, ohne das Programm zu benutzen, die Datei1 und Datei2 lesen, schreiben und ausführen. Mitglieder Ihrer Gruppe können, ohne das Programm zu benutzen, die Datei1 lesen und schreiben.

2.9 Die Dateien .profile und /etc/profile

Die Dateien .profile und /etc/profile sind besondere Dateien. Nach diesen Dateien sucht die Shell gleich nach dem Login. Wenn sie vorhanden sind, werden sie sofort ausgeführt.

Die Datei /etc/profile kann nur der Systemverwalter erstellen. Sie wird für *jeden* Benutzer des Systems beim Login ausgeführt.

Die Datei .profile kann sich jeder Benutzer selbst erstellen. Bedingung ist nur, daß der Name .profile verwendet wird und daß Sie diese unter Ihrem Login-Dateiverzeichnis erstellen. (Dazu können Sie z.B. einen Editor verwenden oder die Kommandos `cat > ...` oder `echo`.) Die Datei .profile gilt nur für Ihre Kennung.

Wenn im System beide Dateien existieren, eine Datei /etc/profile und Ihre .profile, so werden bei Ihrem Login beide Dateien ausgeführt, erst die /etc/profile und unmittelbar danach Ihre .profile.

Die .profile können Sie als komfortables Hilfsmittel nutzen, um nach dem Login, aber noch vor Beginn Ihrer Arbeiten z.B. Variablen zu setzen, zu verändern oder dort eingetragene Kommandos auszuführen. (Auch `echo GUTEN MORGEN` ist möglich.)

Näheres zu Variablen etc. ist im Kapitel Shell erklärt.

.profile

Wie funktioniert die Datei *.profile* und was läuft ab? Das Besondere ist, daß die Shell nur bei *Ihrem* Login nach diesem Dateinamen sucht. Nach dem Login befinden Sie sich in Ihrem Login-Dateiverzeichnis. Bevor die Shell Sie dort "absetzt", durchsucht sie dieses Dateiverzeichnis nach der Datei *.profile*. Wenn sie eine Datei mit Namen *.profile* findet, führt die Shell diese Datei aus. Was kann man dadurch erreichen? Sie können durch diese Datei z.B. Variablen an die Shell übergeben, mit denen Sie während Ihrer Sitzung arbeiten wollen oder Shell-Variablen verändern. Wenn Sie in Prozeduren mit Kennwortparametern arbeiten wollen, können Sie z.B. das Kommando: `set -k` in die Datei *.profile* schreiben und ausführen lassen.

Ein Beispiel für eine *.profile*-Datei:

```
HOME=/usr/lager
PATH=: /bin: /usr/bin: /etc
set -k
name=$USER
echo Hallo
export HOME PATH name
```

Wenn das Ihre *.profile*-Datei wäre, würde bei Ihrem Login folgendes passieren: Sie würden die Standardwerte der Shell-Variablen *HOME* und *PATH* Ihrer Shell umdefinieren, das Kommando `set` ausführen, eine Variable `name=$USER` definieren und außerdem am Bildschirm ausgegeben bekommen: *Hallo*. Zudem hätten Sie die Variablen *HOME*, *PATH* und *name* exportiert.

Die *.profile*-Datei bietet sich auch während einer Sitzung an, wenn man z.B. Variablen der Shell verändert und nicht will, daß diese Änderungen mit dem Ende der Sitzung verloren gehen. Schreiben Sie dann einfach Ihre gewünschten Änderungen in die *.profile*-Datei und übergeben Sie diese mit dem Punkt-Kommando an die Shell. Damit stehen Ihnen die Änderungen sofort in der aktuellen Sitzung zur Verfügung und werden auch bei Ihrem nächsten Login gleich automatisch gesetzt.

/etc/profile

Diese Datei kann nur der Systemverwalter erstellen, weil "normale Benutzer" im Dateiverzeichnis */etc* keine Dateien erstellen dürfen. Die */etc/profile*-Datei funktioniert im Prinzip genauso wie die *.profile*-Datei. Einziger Unterschied: Die */etc/profile*-Datei wird für *jeden* Benutzer beim Login ausgeführt.

3 Die Kommandoebene Shell

Die Kommandoebene Shell verbindet die Benutzer des Systems mit dem Systemkern. Die Shell und der Systemkern sind voneinander getrennt und kommunizieren über Systemaufrufe miteinander (siehe auch: Bild 3-1). Auf den Systemkern haben Benutzer keinen direkten Zugriff.

Die Hauptkomponenten des Systemkerns sind:

- das Dateiverwaltungssystem
- die Prozeßverwaltung
- die Speicherverwaltung
- das Ein-Ausgabesystem

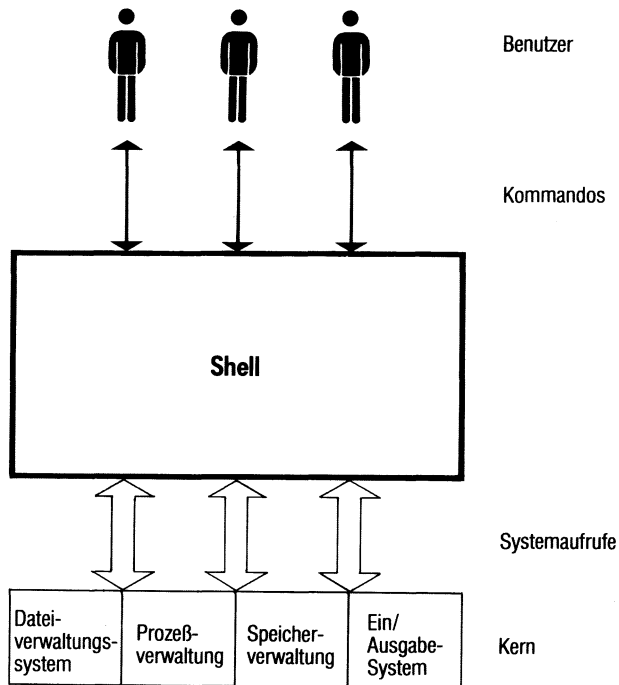


Bild 3-1 Die Kommandoebene Shell im SINIX-System

Die Kommandoebene Shell bietet dem Benutzer eine umfangreiche Kommandosprache. Im Gegensatz zu anderen dialogorientierten Kommandosprachen hat sie den Vorteil, daß sie sich wie eine Programmiersprache anwenden läßt. Gleichzeitig erlaubt es die Shell, eigene neue Kommandos zu erstellen bzw. Kommandos miteinander zu verknüpfen.

Die Shell liest Kommandos entweder von einer Datensichtstation oder aus einer Datei und startet entsprechende Programme. In einer Datei stehende Kommandos heißen Shell-Prozeduren. Wie Sie solche Prozeduren erstellen können und auf was dabei zu achten ist, wird in diesem Kapitel beschrieben. Ebenso alle anderen Möglichkeiten, die Ihnen die Shell im Umgang mit Kommandos bietet.

3.1 Grundsätzliches über Kommandos

Wann kann man ein Kommando eingeben?

Nach dem Login (anschießen an SINIX) können Sie grundsätzlich jederzeit ein Kommando eingeben. Ihre Eingabe wird von SINIX immer entgegengenommen, aber nur dann verarbeitet, wenn die Shell empfangsbereit ist. Das ist sie, wenn Sie am Bildschirm das Empfangsbereitsymbol ausgegeben bekommen. Das ist in der Regel das Zeichen: \$. Geben Sie nach einer Eingabe sofort wieder etwas ein, ohne vorher das Zeichen \$ bekommen zu haben, so wird Ihre Eingabe zwar von SINIX angenommen und am Bildschirm abgebildet, jedoch erst verarbeitet, wenn die Shell das nächste Zeichen \$ auf den Bildschirm schreibt. Eine Ausnahme ist Taste DEL, sie wird immer sofort ausgeführt. Mit ihr kann man z.B. eine laufende Prozedur abbrechen.

Merke:

Die Shell verarbeitet ein Kommando erst, wenn sie empfangsbereit ist. Das ist sie, wenn am Bildschirm das Zeichen \$ erscheint.

Erscheint an Ihrem Bildschirm das Zeichen: >, erwartet die Shell weitere Eingaben zum gerade eingegebenen Kommando. Das passiert, wenn Sie:

- mit Kommandos für Shell-Prozeduren im Dialog arbeiten
- eine Zeichenkette in der Kommandozeile mit einem Apostroph begonnen haben, ohne sie mit einem Apostroph zu beenden.
- eine Kommandozeile mit dem Zeichen: \ abschließen. Dadurch entwerfen Sie das Zeichen <Neue Zeile> und die Shell nimmt an, daß Sie mit der Kommandoeingabe in der nächsten Zeile fortfahren wollen.

Wie muß bzw. kann man ein Kommando eingeben?

Standardmäßig folgt nach dem Empfangsbereitschaftszeichen ein Leerzeichen. Direkt an das Leerzeichen anschließend, kann ein Kommando folgen. Abschließen muß man eine Kommando-eingabe mit der Taste .

Beispiel: cat datei 

Mehrere Kommandos kann man zusammen in einer Kommando-Liste eingeben. Wie kann eine Kommando-Liste aussehen? Sie kann bestehen aus:

- einem oder mehreren Kommandos
- einer oder mehreren Pipelines

Mehrere Angaben, in einer Kommando-Liste, müssen durch eines der folgenden Zeichen voneinander getrennt sein:

; & && oder ||

Die Zeichen && und || haben eine höhere Priorität als die Zeichen ; und &. Die einzelnen Zeichen haben folgende Wirkung:

- ; bewirkt, daß die einzelnen Kommandos oder Pipelines einer Kommando-Liste *nacheinander* ausgeführt werden.
- & bewirkt, daß die einzelnen Kommandos oder Pipelines einer Kommando-Liste als *Hintergrundprozeß* ablaufen.

&& und || fragen den *Ende-Status* eines Kommandos oder einer Pipeline ab.

Auf && folgende Kommandos oder Pipelines werden nur dann ausgeführt, wenn der Ende-Status *Null* gemeldet wird (d.h. das vorhergehende Kommando oder die Pipeline wurde erfolgreich ausgeführt).

- Auf || folgende Kommandos oder Pipelines werden nur dann ausgeführt, wenn als Ende-Status *ungleich Null* gemeldet wurde (d.h. das Kommando oder die Pipeline wurde nicht erfolgreich ausgeführt).

Wie verarbeitet die Shell Kommandos?

Bis auf "eingebaute Kommandos" (siehe: Kommandos der Shell) ist jedem Kommando eine eigene Datei zugeordnet. Die Namen der meisten Kommandodateien sind in den Dateiverzeichnissen `/bin` und `/usr/bin` eingetragen. Zur Ausführung liest die Shell den Namen des Kommandos und sucht in diesen Dateiverzeichnissen nach dem entsprechenden Dateinamen. Die gesuchte Datei enthält ein Programm, das das eingegebene Kommando ausführt.

Um ein Kommando auszuführen, erzeugt die Shell meist einen neuen Prozeß und wartet anschließend, bis er fertig ist. Danach gibt die Shell das Zeichen `$` aus und kann ein neues Kommando verarbeiten.

Dazu ein Beispiel:

Auf die Eingabe

`date`

bekommt man umgehend das aktuelle Datum ausgegeben, anschließend schickt die Shell das Zeichen `$` auf den Bildschirm und ist damit bereit, ein neues Kommando zu verarbeiten.

Bei einem Kommando wie `date` ist die Zeitspanne klein, die zwischen der Eingabe des Kommandos und der erneuten Empfangsbereitschaft der Shell vergeht. Es könnte jedoch zu längeren Wartezeiten an der Datensichtstation führen, wenn man z.B. eine Datei übersetzt. Denn die Shell meldet sich ja erst zurück, nachdem sie ein Kommando ausgeführt hat.

Um längere Wartezeiten zu vermeiden, gibt es das Sonderzeichen: `&`. Wenn man mit diesem Zeichen eine Kommandoeingabe beendet, meldet sich die Shell sofort mit dem Empfangsbezeichner: `$` am Bildschirm zurück, anstatt erst auf die Beendigung des eingegebenen Kommandos zu warten. Man kann dann sofort ein neues Kommando eingeben und braucht nicht zu warten. Warum? Weil das gestartete Kommando als Hintergrundprozeß gestartet wurde, um den Sie sich nicht weiter zu kümmern brauchen.

Beispiel:

`cc pgm.c&`

ruft den C-Compiler auf, um die Datei `pgm.c` als Hintergrundprozeß zu übersetzen. Mit dem Kommando `ps` kann man sich eine Liste der aktiven Prozesse ausgeben lassen. Auf diese Art kann man Hintergrundprozesse im Auge behalten.

3.2 Umleiten der Standard-Ein-Ausgabe

Die Shell nimmt an, daß im Normalfall Ein- und Ausgaben mit der Datensichtstation verbunden sind, von der aus ein Kommando eingegeben wurde. Im Klartext heißt das: Die Eingabe kommt von der Tastatur und die Ausgabe geht auf den Bildschirm. Diesen Ein-Ausgabefluß kann man ändern. Durch Umleitungen. So kann z.B. eine Eingabe an die Shell anstatt von der Tastatur aus einer *Datei* kommen. Eine Ausgabe kann man ebenfalls anstatt auf den Bildschirm in eine *Datei* oder auf einen *Drucker* leiten.

Wie leitet man eine Ausgabe um?

Ein Beispiel: Gibt man das folgende Kommando ein:

```
ls
```

bekommt man die Dateinamen des aktuellen Dateiverzeichnisses am Bildschirm ausgegeben. Wenn man die folgende Eingabe macht:

```
ls > dateiliste
```

werden die Einträge des aktuellen Dateiverzeichnisses in die Datei *dateiliste* geschrieben. Diese Datei wird erzeugt, falls es sie noch nicht gab oder aber überschrieben, wenn sie schon existierte.

Die *Umleitung* der Ausgabe wird durch das Zeichen: `>` erreicht. Das Zeichen `>` bedeutet: Schreibe die durch dieses Kommando erzeugte Ausgabe in die, nach dem Zeichen `>` angegebene Datei, anstatt auf den Bildschirm der Datensichtstation.

Mit dem Zeichen `>` und dem Kommando `cat` kann man auch mehrere Dateien in eine Datei zusammenfassen:

```
cat d1 d2 d3 > sammel
```

Durch dieses Kommando werden die Dateien `d1`, `d2` und `d3` in der Datei *sammel* zusammengefaßt. Existierte die Datei *sammel* bereits, wird ihr alter Inhalt überschrieben. Soll ein alter Inhalt erhalten bleiben, muß man die Eingabe so machen:

```
cat d1 d2 d3 >> sammel
```

Das Zeichen: `>>` bedeutet: *Datei erweitern*; in anderen Worten: Schreibe die durch das Kommando `cat` zusammengefaßten Dateien `d1`, `d2` und `d3` hinter einen in der Datei *sammel* stehenden Inhalt.

Wie leitet man eine Eingabe um?

⌋ Eine Eingabe leitet man um mit dem Zeichen: <. Es bedeutet: Nimm die Eingabe für das Kommando aus der dem Zeichen < folgenden Datei.

Beispiel: Mit dem Kommando `wc` kann man Zeilen, Worte und Zeichen zählen:

`wc <eric`

liest seine Eingabedaten aus der Datei *eric* und gibt am Bildschirm die Anzahl der gefundenen Zeilen, Worte und Zeichen aus.

⌋ Hinweis

Die Standard-Eingabe läßt sich auch mit dem Kommando `exec` umleiten (siehe Abschnitt 3.8.5).

3.3 Pipeline

Man kann zwei Kommandos zu einer *Pipe* ketten und auch mehrere Kommandos zu einer *Pipeline*. Mit dem äußerlich sehr einfachen Pipe-Mechanismus läßt sich die Ausgabe eines Programms mit dem Eingang eines nachfolgenden Programms *verbinden*. Was heißt das?

Ein Beispiel. Angenommen, Sie wollen wissen, wieviele Einträge ein Dateiverzeichnis enthält. Das könnten Sie dadurch erreichen, indem Sie folgende Kommandos nacheinander eingeben:

```
ls > temp
wc -l < temp
rm temp
```

Jetzt haben Sie die Ausgabe des Kommandos `ls` in die Datei `temp` geschrieben, anschließend durch das Kommando `wc -l` die Anzahl der Zeilen (`wc` = word count, `-l` steht für lines = Anzahl der Einträge) ermittelt und als drittes, mit dem Kommando `rm`, die "Schmierdatei": `temp` wieder gelöscht. Wenn man die gleichen Kommandos durch eine Pipeline miteinander verbindet, geht das ganze viel schneller und einfacher - mit der gleichen Wirkung!

```
ls|wc -l
```

| ist das Symbol zum Verbinden von zwei Kommandos zu einer Pipe.

Im obigen Beispiel wird damit erreicht, daß die vom Kommando `ls` erzeugte Ausgabe unmittelbar dem Kommando `wc` als Eingabe übergeben wird.

Mit dem Verbinden zweier Kommandos durch das Pipe-Symbol `|`, läßt sich die Standard-Ausgabe des ersten Kommandos direkt als Standard-Eingabe des nächsten Kommandos verwenden.

Auf diese Art können Sie bis zu 10 Kommandos verbinden.

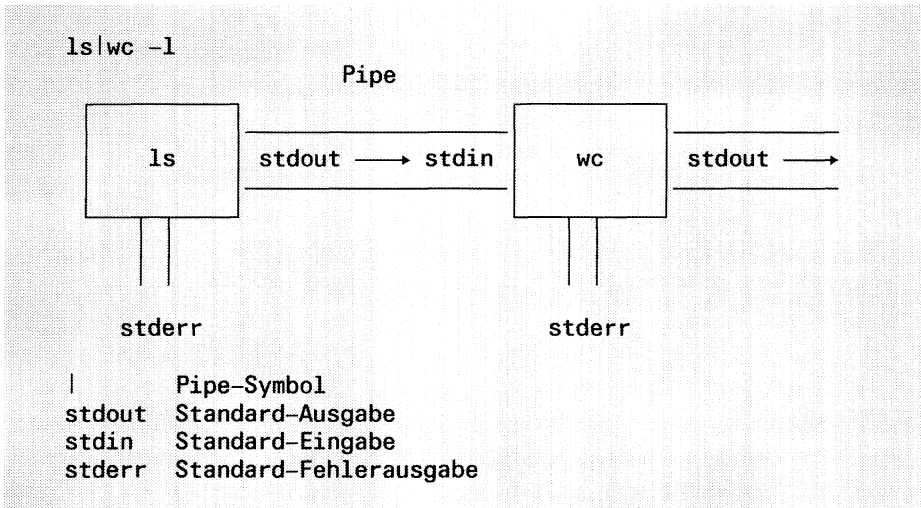


Bild 3-2 Pipe-Mechanismus

Alle in einer Pipeline stehenden Teilaufgaben startet die Shell gleichzeitig als quasiparallele Prozesse und synchronisiert sie über einen Puffer. Ein Kommando liest und verarbeitet Daten, sobald das vorhergehende Kommando sie auf seine Standard-Ausgabe schreibt. Dadurch braucht ein Kommando nicht unbedingt darauf zu warten, bis das vorhergehende Kommando voll beendet ist. Wenn Sie das folgende Kommando eingeben:

```
ls /usr/bin
```

wird am Bildschirm der Inhalt des Dateiverzeichnisses `/usr/bin` (das die Namen der Benutzerkommandos enthält) am Bildschirm ausgegeben. Es sind dort mehr Einträge vorhanden, als auf einem Bildschirm darstellbar sind; die Folge ist, daß die Zeilen durchlaufen werden bis zum letzten Eintrag.

Geben Sie jetzt folgendes ein:

```
ls /usr/bin|pg
```

Das Kommando `pg` kann für seitenweise Bildschirmausgabe verwendet werden (`page`). Sobald jetzt das Kommando `ls` über seine Standard-Ausgabe etwas in die Pipeline eingibt, liest es das Kommando `pg` und gibt einen Bildschirm aus, obwohl das Kommando `ls` noch nicht abgeschlossen ist.

Mit Betätigen der Taste  wird der nächste Bildschirm ausgegeben usw. Nach der letzten Zeile steht am Bildschirm (EOF);, was das Ende der Datei angibt (end of file). Dann ist auch das Kommando ls abgeschlossen.

Welche Kommandos kann man in einer Pipeline benutzen?

In einer Pipeline kann man alle Kommandos benutzen, die von der Standard-Eingabe lesen und auf die Standard-Ausgabe ausgeben. Am Anfang oder Ende können auch andere Kommandos stehen (z.B. ls, who, date).

Wozu kann man eine Pipeline benutzen?

Mit dem Pipe-Mechanismus kann man leistungsfähige Routinen formulieren und damit kleinere bis mittlere Aufgaben schnell, einfach und elegant lösen.

Angenommen es sollen aus zwei Dateien bestimmte Zeilen gesucht und sortiert ausgegeben werden.

Inhalt der Datei: *kunde1*

hartmann	muenchen	kundennummer: 160349
richter	muenchen	kundennummer: 300350
schmidt	muenchen	kundennummer: 190459
bap	koeln	kundennummer: 310846
borg	koeln	kundennummer: 301462

Inhalt der Datei: *kunde2*

roman	muenchen	kundennummer: 160856
strepel	muenchen	kundennummer: 300461
blumann	muenchen	kundennummer: 190851
zelting	koeln	kundennummer: 311249
buchber	koeln	kundennummer: 302368

Beispiel 1: Aus beiden Dateien sollen am Bildschirm aufgelistet werden, alle Kunden aus muenchen, alphabetisch sortiert.

Eingabe:
`cat kunde1 kunde2 | grep muenchen | sort`

Ausgabe:
blumann muenchen kundennummer: 190851
hartmann muenchen kundennummer: 160349
richter muenchen kundennummer: 300350
roman muenchen kundennummer: 160856
schmidt muenchen kundennummer: 190459
strepel muenchen kundennummer: 300461

Beispiel 2: Aus beiden Dateien sollen am Bildschirms aufgelistet werden, alle Kunden mit Kundennummer 30..., alphabetisch sortiert.

Eingabe:
`cat kunde1 kunde2 | grep :30 | sort`

Ausgabe:
borg koeln kundennummer: 301462
buchber koeln kundennummer: 302368
richter muenchen kundennummer: 300350
strepel muenchen kundennummer: 300461

Was wurde gemacht?

In beiden Fällen wurden die Dateien *kunde1* und *kunde2* durch das cat-Kommando gelesen. Durch Angabe des Pipe-Zeichens: | wurden sie an das grep-Kommando übergeben und dort nach den gewünschten Begriffen durchsucht. Jede Zeile, die den entsprechenden Suchbegriff enthielt, wurde an das sort-Kommando übergeben und von diesem in alphabetischer Reihenfolge ausgegeben. - So einfach geht das!

3.4 Der Apostrophier-Mechanismus

An die Shell kann man Zeichenketten übergeben und durch festgelegte Anfangs- und Ende-Kennzeichnungen angeben, was die zu verarbeitende Zeichenkette bewirken soll. Die Shell arbeitet dabei nach ihrem Apostrophier-Mechanismus und interpretiert die Zeichenketten entsprechend.

Es gibt 3 Möglichkeiten, eine Zeichenkette durch Apostrophe zu kennzeichnen:

"...."

Steht eine Zeichenkette zwischen Anführungszeichen, dann:

- liest die Shell den Inhalt,
- ersetzt vorhandene Stellungs- oder Kennwortparameter,
- Shell interpretiert nur Sonderzeichen

\$ `...` \

- *führt Kommando aus* mit der Zeichenkette als Argument

`....`

Steht eine Zeichenkette innerhalb Akzent Gravis, dann:

- interpretiert Shell den *Inhalt als Kommando* und führt es aus
- Ergebnis der Kommandoausführung ersetzt die Zeichenkette

'....'

Steht eine Zeichenkette zwischen Apostrophen, liest sie die Shell

- *ignoriert* jede Sonderbedeutung, ersetzt nichts
(= 1 Argument)

Beispiele für den Apostrophier-Mechanismus:

Angenommen, es wurde folgende Eingabe gemacht: name=Meier. Die Shell merkt sich diese Eingabe in ihrem aktuellen Umfeld (als Variable). Damit kann man mit dieser Eingabe als Kennwortparameter arbeiten.

Die folgenden Beispiele zeigen, wie man den Apostrophier-Mechanismus darauf anwenden kann.

Eingabe: echo "\$name"

Ausgabe: Meier

Eingabe: echo ' \$name '

Ausgabe: \$name

Eingabe: echo ` \$name `

Ausgabe: Meier: not found

Eingabe: echo "Der Name des Kunden ist: \$name"

Ausgabe: Der Name des Kunden ist: Meier

Eingabe: echo "\$name `date`"

Ausgabe: Meier plus aktuelles Datum

Eingabe: echo date

oder: echo "date"

oder: echo 'date'

Ausgabe: date

Eingabe: echo `date`

Ausgabe: Aktuelles Datum

In den bisherigen Beispielen liest die Shell von der Standard-Eingabe (Tastatur) und gibt auf die Standard-Ausgabe (Bildschirm) aus. Die Shell behandelt die eingegebenen Zeichenketten entsprechend dem angegebenen Apostrophier-Mechanismus.

Weitere Beispiele mit Kennwortparameter: `name = Meier`

Eingabe:

```
echo "echo Der Name ist: $name"> text
```

schreibt in die Datei: `text`

```
echo Der Name ist: Meier
```

Wie man sieht, wird der Kennwortparameter `$name` durch seinen definierten Wert (Meier) *ersetzt* und in die Datei `text` geschrieben.

Bei Aufruf dieser Datei würde sie ausgegeben: Der Name ist: Meier

Dieser Aufruf entspricht den Konventionen, wenn man die Zeichenkette in Anführungszeichen `"...."` einschließt.

Gibt man die gleiche Zeichenkette in Apostrophe `'....'` ein:

```
echo 'echo Der Name ist: $name'> text
```

werden die Apostrophe nur entfernt, nichts ersetzt und in die Datei `text` folgendes eingetragen:

```
echo Der Name ist $name
```

Wie man sieht, wird jetzt der Kennwortparameter `$name` nicht sofort durch seinen definierten Wert ersetzt. Erst wenn man die Datei `text` ausführt, wird `$name` durch den Wert: Meier ersetzt (substituiert).

Ebenso würde bei Eingabe von:

```
echo ``date``> text
```

in die Datei `text`

```
`date`
```

eingetragen. Erst beim Ausführen der Datei `text` wird `date` als Kommando erkannt und am Bildschirm das aktuelle Datum angezeigt.

Bei Eingabe von:

```
echo `date`> text
```

wird in die Datei `text`

```
`date`
```

eingetragen. Ruft man dann die Datei `text` auf, erwartet die Shell ausführbare Kommandos. Was passiert? Als erstes liest die Shell die Zeichenkette

``date`` und substituiert darin das date-Kommando.

An Stelle von ``date`` steht dann das aktuelle Datum und die Shell versucht, es als ausführbares Kommando zu behandeln. Das geht schief. Es ist kein ausführbares Kommando - man erhält die Meldung:
text: execute permission denied (Kommando abgewiesen).

3.5 Variablen für die Shell

Die Shell erlaubt die Verwendung von Variablen. Variablen kann man z.B. dazu benutzen, um Abläufe zu steuern. Eine Variable besteht aus einem *Variablennamen* und einem *Wert*. So definiert man eine Variable in der Shell:

Eingabe: `name = Hans`

Durch diese Eingabe hat man in der Shell eine Variable mit dem Wert *Hans* definiert. Will man mit dem *Wert* einer Variablen arbeiten, z.B. im Dialog oder in einer Prozedur, so muß man den Namen der Variablen angeben und davor das *Zeichen* `$` stellen. (Siehe auch: `export`- und `readonly`-Kommando in diesem Kapitel).

`$Variablenname`

Beispiel

Eingabe 1: `name = fred`

Eingabe 2: `echo $name`

Ausgabe: `fred`

Was wurde gemacht? Es wurde die Variable *name* mit dem Wert *fred* definiert. Anschließend wurde der Variablenwert *fred* durch das `echo`-Kommando aufgerufen und am Bildschirm ausgegeben.

Den Namen für Variablen kann man frei wählen. Im SINIX-System sind lediglich die Namen einiger Standard-Variablen festgelegt. Bis auf die Variable `MAIL`, werden diesen Standardwerte von der Shell zugewiesen. Alle Standardwerte können vom Benutzer geändert werden.

Aber Achtung: Ändert man den Standardwert einer Standard-Variablen, kann man mit dem geänderten Wert in einer anderen als der Login-Shell nur dann arbeiten, wenn man ihn vorher *exportiert* hat (siehe Kommando `export` in diesem Kapitel). Andernfalls arbeitet man außerhalb der Login-Shell immer nur mit den zugewiesenen Standardwerten der Standard-Variablen, d.h. mit den "nicht geänderten" Werten.

Standard-Variablen für die Shell

Erklärung einiger Standard-Variablen für die Shell:

Variable	Bedeutung
HOME= ...	Dieser Variablen weist die Shell den in der /etc/passwd-Datei definierten Namen des Login-Dateiverzeichnisses zu. Gibt man das cd-Kommando ohne weitere Angaben an, benutzt es den durch diese Variable definierten Wert.
IFS= ...	Durch IFS kann man Feld-Trennzeichen definieren. Das sind die Zeichen, die die Shell als Leerzeichen interpretiert. Standardwerte sind: Leerzeichen, Tabulator und Neue Zeile
MAIL= ...	Durch diese Variable gibt man den Pfadnamen einer Post-Datei an. Jedesmal wenn etwas in diese Post-Datei geschrieben wird, bekommt der Eigentümer während der Sitzung eine Nachricht am Bildschirm ausgegeben. Hinweis: Die Variable hat keinen Einfluß auf das mail-Kommando. Das mail-Kommando schreibt Post immer in die Standard-Post-Datei (Name: /usr/spool/mail/\$USER).
PATH= ...	Durch diese Variable gibt man einen oder mehrere Suchpfade an, die von der Shell durchsucht werden sollen, um ein Kommando zu finden. Die Suchpfade werden von der Shell in der Reihenfolge ihrer Angabe durchsucht. Die folgenden Suchpfade werden von der Shell als Standardwerte definiert: :/usr/bin/local:/bin:/usr/bin. Die Doppelpunkte trennen die einzelnen Suchpfade voneinander. Beginnt eine Suchpfadangabe mit einem Doppelpunkt, wird dieser Doppelpunkt jedoch als aktuelles Dateiverzeichnis interpretiert, d.h. die Shell beginnt die Suche nach einem Kommando im aktuellen Dateiverzeichnis. Soll das aktuelle Dateiverzeichnis als letztes durchsucht werden, muß man es so schreiben: PATH=/usr/bin/local:/bin:/usr/bin::
PS1= ...	Durch PS1 wird ein Zeichen oder eine Zeichenfolge festgelegt, womit die Shell einen Benutzer zur Eingabe auffordert. Als Standardwert trägt die Shell ein \$-Zeichen mit anschließendem Leerzeichen ein. Diese Angabe kann man jederzeit ändern.
PS2= ...	Durch PS2 wird ein Fortsetzungszeichen definiert, das die Shell an den Anfang der nächsten Zeile setzt, falls eine Kommandoeingabe länger als eine Zeile ist. Als Standardwert trägt die Shell ein >-Zeichen mit anschließendem Leerzeichen ein.
TERM=...	Durch diese Variable wird ein Datensichtstationstyp definiert. Als Standardwert trägt die Shell den in der Datei: ttytype eingetragenen Wert ein (z.B. 97801).
USER=...	Durch diese Variable gibt man eine Benutzererkennung an. Als Standardwert trägt die Shell die aktuell an der Datensichtstation eingegebene Benutzererkennung ein.

Shell

Die Shell versteht auch einige Sonderzeichen. Gibt man sie zusammen mit dem Zeichen \$, z.B. echo \$? im Dialog an der Datensichtstation ein oder ruft man sie in einer Prozedur auf, ersetzt sie die Shell durch definierte Werte. Das sind sie:

Zeichen	Bedeutung
#	Gibt die Anzahl der angegebenen Stellungsparameter dezimal an.
-	Schalter, mit denen die Shell aufgerufen wurde oder durch das set-Kommando gesetzt wurden.
?	Der zurückgemeldete Wert (dezimal) des zuletzt ausgeführten Kommandos (Ende-Status eines Kommandos).
\$	Die Prozeßnummer dieser Shell.
!	Die Prozeßnummer des zuletzt aufgerufenen Hintergrundkommandos.
* oder @	Dafür werden an der angegebenen Stelle alle vorhandenen Stellungsparameter eingesetzt, beginnend bei \$1 und voneinander durch ein Leerzeichen getrennt.

Beispiel:

Eingabe:	unsinn	(als Kommando)
Ausgabe:	unsinn: not found	(Meldung von SINIX)
Eingabe:	echo \$?	(Abfrage des Rückmeldewertes)
Ausgabe:	1	(Ausgabe von SINIX, 1 = ohne Erfolg)

3.6 Shell Prozeduren

Als Programmierer sind Sie es gewohnt, Probleme mit Hilfe von Programmen zu lösen, die in einer Programmiersprache geschrieben wurden. Bevor ein Programm ablauffähig ist, müssen Sie Tätigkeiten ausführen, die mit der eigentlichen Problemlösung noch nichts zu tun haben, z.B. Programme übersetzen und binden. Das kostet Zeit, erfordert Organisationsaufwand und kann auch recht langweilig sein.

Die *Shell* bietet Ihnen Kommandos und Ablaufanweisungen, die sich wie eine Programmiersprache nutzen lassen, wie in den nachfolgenden Abschnitten beschrieben ist. Gegenüber "normalen" Programmiersprachen besitzt sie jedoch den großen Vorteil, daß Sie von "lästigem" Administrationsaufwand (z.B. Übersetzen) befreit sind. Sie können sich voll auf Ihre Problemlösung konzentrieren.

Zu einem Programm zusammengefaßte Kommandos bezeichnet man als Shell-Prozedur. Um eine Shell-Prozedur zu erstellen, müssen Sie einfach die gewünschte Kommandofolge in eine Datei schreiben. Anschließend können Sie diese Kommandofolge sofort ausführen, nur mit mit Eingabe des Dateinamens.

Beispiel für eine Shell-Prozedur:

Setzen Sie sich am besten an eine Datensichtstation und probieren das folgende einfache Beispiel.

1. Schritt: Geben Sie ein

```
cat > auswahl    (Taste  drücken)
who              (Taste  drücken)
date            (Taste  drücken)
mail            (Taste  drücken)
                (Taste  drücken)
```

Der Eingabe-Abschluß mit z.B. ist nachfolgend nicht extra angegeben.

2. Schritt: Datei *auswahl* ausführen

Dazu muß für die Datei *auswahl* eine *Ausführungserlaubnis* bestehen. Die vergibt man durch das Kommando *chmod*. Geben Sie folgendes ein:

```
chmod +x auswahl    (x-Bit setzen)
```

Shell

Wenn Sie jetzt den Dateinamen eingeben:

`auswahl`

werden die in der Datei *auswahl* stehenden Kommandos *ausgeführt*. Sie haben damit 3 Kommandos durch eine Eingabe gestartet:

Am Bildschirm erscheint eine Liste aller mit dem System verbundenen Benutzer, das aktuelle Datum sowie die für Sie vorhandene Post.

Besteht für die Datei *auswahl* keine Ausführungserlaubnis, kann man das in ihr enthaltene Programm durch folgende Eingabe ebenfalls starten:

`sh auswahl`

Das Kommando `sh` ist u.a. im Kapitel: Dateisystem erklärt. Jetzt wissen Sie, wie man grundsätzlich eine einfache Shell-Prozedur erstellt und zum Ablauf bringt. Die folgenden Kapitel enthalten detaillierte Informationen darüber, was man mit Shell-Prozeduren alles machen kann.

3.6.1 Parameter für Shell-Prozeduren

Jede Prozedur hat einen Variablen- und Parameterbereich (siehe: Bild 3-3). In diesen Bereich können Sie:

- Stellungsparameter eintragen
- Variablen eintragen, deren Werte man als Kennwortparameter aufrufen kann

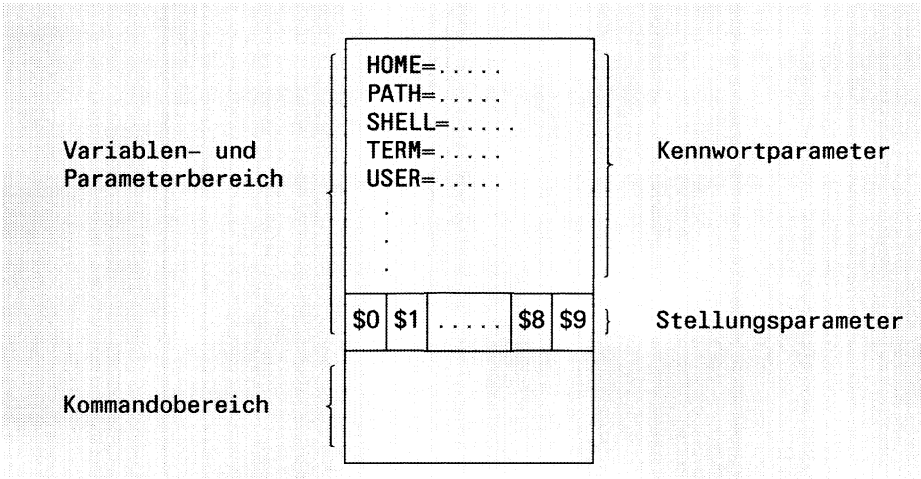


Bild 3-3 Der Variablen- und Parameterbereich für eine Prozedur

In der Login-Shell sind für jeden Benutzer Variablen mit Standardwerten vorhanden (siehe: Variablen der Shell). Einige dieser Variablen stehen mit ihren Standardwerten jeder Prozedur in ihrem Variablen- und Parameterbereich automatisch zur Verfügung. Welche das sind, können Sie mit dem Kommando env abfragen.

Wie kann man in einer Prozedur mit Stellungsparametern arbeiten?

Wird eine Prozedur von einer Datensichtstation aus oder aus einer Prozedur heraus aufgerufen, numeriert die Shell die Angaben in der Kommandozeile. Der Name der Prozedur bekommt die Nummer 0 zugewiesen. Die erste folgende Angabe die Nummer 1, die zweite Angabe die Nummer 2 usw.. Die durchnummerierten Werte einer Kommandozeile kann man als Stellungsparameter aufrufen. Dazu muß man das Zeichen \$ benutzen. Den Namen einer Prozedur kann man durch die Angabe \$0 aufrufen, die erste folgende Angabe durch \$1 usw.. Man kann auf bis zu 9 Stellungsparameter direkt zugreifen. Will man mit mehr als 9 Stellungsparametern arbeiten, muß man das shift-Kommando (siehe: Kommandos der Shell) oder \$* benutzen. Mit \$* liest man alle Stellungsparameter einer Kommandozeile gleichzeitig.

Nur die erste Ziffer hinter dem \$-Zeichen wird als Stellungsparameter interpretiert. Gibt man z.B. \$10 an, interpretiert die Shell diese Angabe im Klartext so: Lese Stellungsparameter \$1 und hänge eine 0 an.

Stellungsparameter sind nicht exportierbar, d.h. in einer Prozedur gesetzte Stellungsparameter können nicht durch das export-Kommando für eine andere Prozedur verfügbar gemacht werden. Das ist ein Gegensatz zu Kennwortparametern. Direkt kann man mit Stellungsparametern nur in der Prozedur arbeiten, für die sie bestimmt sind.

Beispiele, wie Sie beim Aufruf einer Prozedur über die Kommandozeile Stellungsparameter übergeben können.

Beispiel 1: Angenommen, Sie wollen von einer Datenstation aus, an eine Prozedur einen Stellungsparameter übergeben. Name der Datei in der die Prozedur steht: *kunde*. Inhalt der Datei:

```
echo Der Name des Kunden ist: $1
```

Wenn Sie diese Prozedur durch folgendes Kommando aufrufen:

```
kunde Huber
```

erscheint am Bildschirm folgende Ausgabe:

```
Der Name des Kunden ist: Huber
```

Für den in der Prozedurdatei *kunde* angegebenen Stellungsparameter \$1, wurde beim Aufrufen der Prozedur der in der Kommandozeile in Position 1 stehende Parameter (d.h. Huber) eingesetzt und durch das echo-Kommando ausgegeben.

Beispiel 2: Angenommen, Sie wollen aus einer Prozedur heraus eine andere Prozedur aufrufen und dabei einen Stellungsparameter übergeben. Namen der Dateien, in denen die Prozeduren stehen:

Aufrufende Prozedur: *kunde*
Aufgerufene Prozedur: *ausgabe*

Inhalt von: *kunde*
echo Der Name des Kunden ist: \$1
ausgabe 21479

Inhalt von: *ausgabe*
echo Seine Kundennummer ist: \$1

Wenn Sie jetzt folgende Eingabe machen:

kunde Meier

erscheint am Bildschirm folgende Ausgabe:

Der Name des Kunden ist: Meier
Seine Kundennummer ist: 21479

Was ist passiert? An die Prozedurdatei *kunde* wurde aus der an der Datensichtstation eingegebenen Kommandozeile der Stellungsparameter \$1 übergeben (d.H. Meier) und durch das echo-Kommando auf den Bildschirm geschrieben. Durch die daran anschließende Kommandozeile wird die Prozedur *ausgabe* aufgerufen. Diese Kommandozeile enthält zusätzlich noch die Angabe: 21479, die in der Prozedur *ausgabe* durch den Stellungsparameter \$1 gelesen und durch das echo-Kommando ausgegeben wird.

Wie kann man in einer Prozedur mit Kennwortparametern arbeiten?

Einer Prozedur stehen beim Aufruf alle Variablen zur Verfügung, die in ihrem Variablen- und Parameterbereich eingetragen sind. In den Variablen- und Parameterbereich einer Prozedur werden automatisch alle Variablen eingetragen, die: (siehe auch: Bild 3-4)

- (1) in einer Prozedur direkt definiert werden
- (2) in der Login-Shell definiert und exportiert wurden
- (3) beim Aufruf der Prozedur über die Kommandozeile an die Prozedur übergeben wurden
- (4) in einer Aufrufhierarchie von Prozeduren (Prozedur1 ruft Prozedur2 auf, Prozedur2 ruft Prozedur3 auf usw.) in einer "unterhalb" der aktuellen Prozedur abgelaufenen Prozedur definiert und exportiert wurden.

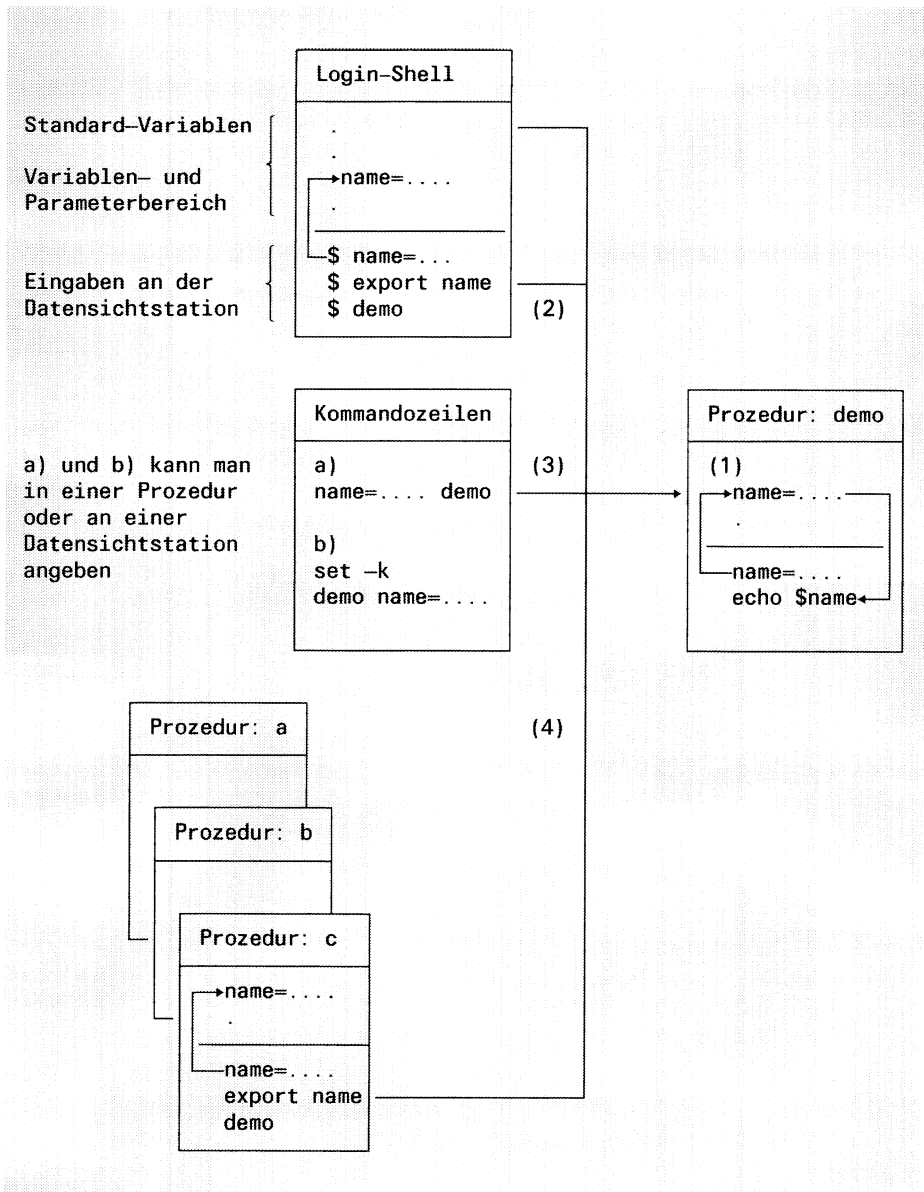


Bild 3-4 Eintrag von Variablen in den Variablen- und Parameterbereich

Den Wert einer Variablen kann man als Kennwortparameter lesen. Dazu muß man den Variablennamen mit dem Zeichen \$ aufrufen. Auf den folgenden Seiten sind dazu weitere Erklärungen und einige Beispiele angegeben.

Wie kann man in einer Prozedur eine Variable definieren und mit ihrem Wert als Kennwortparameter arbeiten?

Ein Beispiel. In der Prozedur *register* soll die Variable *name* = *SINIX* definiert werden (siehe auch: Bild 3-5).

```
Prozedurdatei:  register
Inhalt:         name=SINIX
                echo $name

Aufruf:        sh register
Ausgabe:       SINIX
```

Was wurde gemacht? In den Variablen- und Parameterbereich der Prozedur *register* wurde die Variable *name* = *SINIX* eingetragen. Durch den Kennwortparameter \$name wurde ihr Wert gelesen und durch das echo-Kommando am Bildschirm ausgegeben.

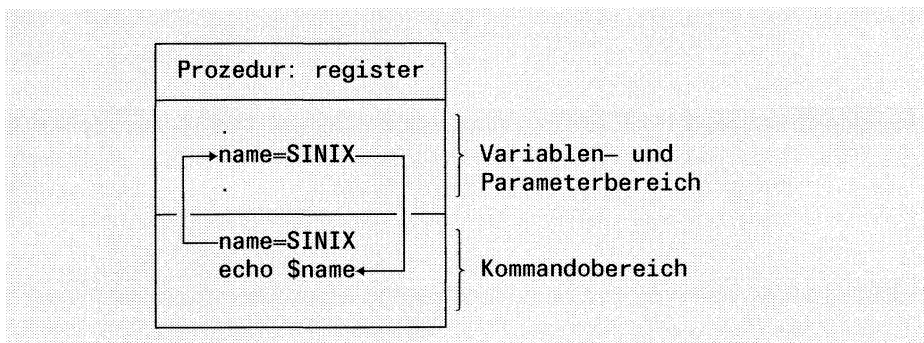


Bild 3-5 Definieren einer Variablen in einer Prozedur

Wie kann man in einer Prozedur mit Variablen arbeiten, die in der Login-Shell definiert sind?

In der Login-Shell definieren Sie eine Variable, indem Sie an der Datensichtstation z.B. folgendes eingeben:

```
name = eric
```

Jetzt haben Sie eine Variable *name = eric* definiert. Um mit dieser Variable auch außerhalb der Login-Shell arbeiten zu können, müssen Sie sie *exportieren* (siehe auch: export-Kommando). Geben Sie folgendes ein:

```
export name
```

Jetzt steht jeder Prozedur, die Sie aufrufen, in ihrem Variablen- und Parameterbereich die Variable *name* zur Verfügung. Angenommen, Sie haben folgende Prozedur:

```
Prozedurdatei: kartei
Inhalt:      echo $name

Aufruf:      kartei
Ausgabe:     eric
```

Was wurde gemacht? Beim Aufruf der Prozedur *kartei* wurde in ihren Variablen- und Parameterbereich die in der Login-Shell definierte und exportierte Variable *name = eric* eingetragen. Durch den Kennwortparameter *\$name* wurde ihr Wert gelesen und durch das echo-Kommando am Bildschirm ausgegeben (siehe auch: Bild 3-6).

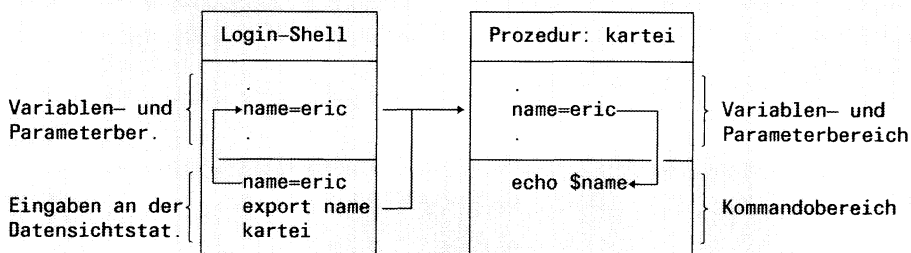


Bild 3-6 Übergabe von Variablen der Login-Shell an eine Prozedur

Wie übergibt man Variablen über die Kommandozeile an eine Prozedur?

Eine Prozedur wird durch eine Kommandozeile aufgerufen, die Sie entweder im Dialog von der Datensichtstation aus eingeben oder die aus einer Prozedur heraus ausgeführt wird. Eine Kommandozeile kann Variablen enthalten, die in den Variablen- und Parameterbereich der aufgerufenen Prozedur geschrieben werden sollen. Das können Sie auf zwei Arten erreichen.

Beispiel: Angenommen, Sie haben folgende Prozedur (siehe auch: Bild 3-7):

Prozedurdatei: *archiv*
Inhalt: `echo $kunde`

Wenn Sie die Prozedurdatei *archiv* so aufrufen:

```
kunde=miller archiv
```

erhalten Sie als Ausgabe am Bildschirm: `miller`.

Wenn Sie die Prozedurdatei *archiv* so aufrufen:

```
archiv kunde=miller
```

erhalten Sie am Bildschirm keine Ausgabe. Warum?

Was wurde gemacht? Beim ersten Aufruf beginnt die Kommandozeile mit der Angabe der Variablen `kunde=miller` gefolgt vom Namen der Prozedur *archiv*. Bei dieser Eingabeform wird die Variable automatisch exportiert, d.h. in den Variablen- und Parameterbereich der aufgerufenen Prozedur *archiv* eingetragen. Der Kennwortparameter `$kunde` kann den Wert lesen. Das `echo`-Kommando gibt ihn am Bildschirm aus.

Beim zweiten Aufruf wurde als erstes der Name der Prozedur (*archiv*) eingegeben und anschließend die Variable (`kunde=miller`). Eine solche Eingabeform "versteht" die Shell nicht. Man muß der Shell mitteilen, wie sie diese Eingabe "zu verstehen hat". Dazu muß man das Kommando `set` benutzen. Geben Sie ein:

```
set -k  
archiv kunde=miller
```

erhalten Sie als Ausgabe am Bildschirm: `miller`

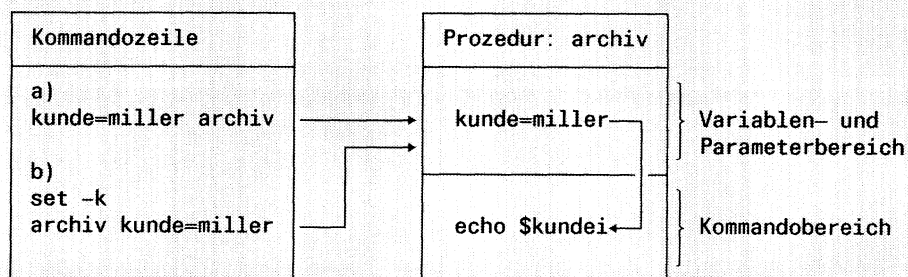
Das Kommando: set -k hat zwei Effekte:

- Alle Angaben in der Kommandozeile, die ein Gleichheitszeichen enthalten, interpretiert die Shell als Variablen.
- Alle Variablen, die Sie in einer Kommandozeile angeben, werden automatisch exportiert. Und zwar für alle Prozeduren, die direkt oder indirekt durch diese Kommandozeile ausgeführt werden.

Aber beachten Sie beim set-Kommando:

Die Wirkung des set-Kommandos ist auf die Shell begrenzt, in der es angegeben wird. Was bedeutet das? Geben Sie das set-Kommando im *Dialog* von der Datensichtstation aus ein, bezieht sich seine Wirkung nur auf Kommandozeilen, die Sie direkt von der Datensichtstation aus eingeben. Wenn Sie innerhalb einer *Prozedur* mit Kommandozeilen arbeiten wollen, müssen Sie in die entsprechende Prozedur ein set-Kommando schreiben. Ein an der Datensichtstation eingegebenes set-Kommando, hat also keine Wirkung auf Kommandozeilen innerhalb einer Prozedur.

Ein im Dialog an einer Datensichtstation eingegebenes set-Kommando gilt nur für die Dauer einer Sitzung. Anschließend ist seine Wirkung aufgehoben. Nach einem erneuten Login muß man deshalb das set-Kommando neu eingeben, wenn man mit Kennwortparametern arbeiten will. Schreibt man das set-Kommando in die Datei *.profile*, wird es bei jedem Login automatisch ausgeführt (vgl. am Ende von Kapitel: Dateisystem).



a) und b) kann man in einer Prozedur oder im Dialog an einer Datensichtstation angeben

Bild 3-7 Übergabe von Variablen über die Kommandozeile

Umwandeln von Kennwort- in Stellungsparameter

Man kann Kennwortparameter in Stellungsparameter umwandeln. Dazu muß man das set-Kommando benutzen. Das kann sehr nützlich sein, wenn man aus dem Wert einer Variablen etwas herausfiltern möchte.

Ein einfaches Beispiel. Angenommen, Sie wollen aus der Ausgabe des Kommandos `date` nur die Uhrzeit ausgegeben bekommen. Über eine Shell-Prozedur (*zeit*) können Sie das wie folgt erreichen:

Inhalt der Datei *zeit*:

```
date >x
exec <x
read xyz
set $xyz
echo $3
```

Was passiert, wenn Sie die Shell-Prozedur *zeit* aufrufen? (Eingabe: `sh zeit`). Als erstes wird das Ergebnis des Kommandos `date` in die Datei *x* geschrieben. Es könnte z.B. so aussehen:

Di 5.Jun.1984, 12:37:02 MEZ. Anschließend wird durch das `exec`-Kommando die Standard-Eingabe auf die Datei *x* umgelenkt. Von dort liest das `read`-Kommando und weist das Ergebnis der Variablen *xyz* als Wert zu. Das `set`-Kommando macht diesen Wert für Stellungsparameter zugreifbar. Mit dem `echo`-Kommando wird der Stellungsparameter `$3` gelesen und ausgegeben. Am Bildschirm erscheint die gewünschte Uhrzeit: 12:37:02 .

Parameter vordefinieren

(Beispiele gelten bei `set -k` gesetzt)

- Standardwerte setzen: `${parameter-wort}` oder `${parameter:-wort}`

Für die Angabe `-` gilt:

Ist für **parameter** beim Aufruf der Prozedur ein Wert gesetzt, wird er eingesetzt, wenn nicht, wird das angegebene **wort** eingesetzt.

Geben Sie `:-` an, wird zusätzlich abgefragt, ob `parameter=""` ist.

Beispiel 1 (mit Stellungsparameter)

Inhalt der Datei `test1`:

```
echo Alex ist: ${1-klein}
```

Aufruf: `test1`

Ausgabe: `Alex ist: klein`

Aufruf: `test1 jung`

Ausgabe: `Alex ist: jung`

Beispiel 2 (mit Kennwortparameter)

Inhalt der Datei `test2`:

```
echo Das Auto ist: ${farbe-schwarz}
```

Aufruf: `test2`

Ausgabe: `Das Auto ist: schwarz`

Aufruf: `test2 farbe=rot`

Ausgabe: `Das Auto ist: rot`

- Standardwerte setzen: `${parameter=wort}` oder `${parameter:=wort}`

Der Wert von `wort` wird eingesetzt wenn:

- bei der Angabe `=` kein parameter angegeben wird
- bei der Angabe `:=` kein parameter oder `parameter = ""` angegeben wird

Beispiel 3

Inhalt der Datei `test3`:

```
echo Der Kunde ist: ${a=Neu}  
echo Die Kundennummer ist: $a
```

```
Aufruf:  test3  
Ausgabe: Der Kunde ist: Neu  
         Die Kundennummer ist: Neu
```

```
Aufruf:  test3 a=21479  
Ausgabe: Der Kunde ist: 21479  
         Die Kundennummer ist: 21479
```

- Prüfen, ob ein Wert gesetzt ist:
`${parameter?wort}` oder `${parameter:?wort}`

Für die Angabe `?` gilt: Der für **parameter** gesetzte Wert wird ersetzt. Ist kein Wert gesetzt, wird **wort** ausgedruckt und die Prozedur beendet. Geben Sie `:?` an, wird zusätzlich abgefragt, ob `parameter = ""` ist. Wird kein **wort** angegeben bekommt man eine Standardmeldung ausgegeben.

Beispiel 1 (mit Stellungsparameter)

Inhalt der Datei `test4`:

```
echo Flugzeuge fliegen: ${1?Abgebrochen}
```

```
Aufruf:  test4 hoch  
Ausgabe: Flugzeuge fliegen: hoch
```

```
Aufruf:  test4  
Ausgabe: test4: 1: Abgebrochen
```

Beispiel 2 (mit Kennwortparameter)

Inhalt der Datei *test5*:

```
echo Flugzeuge fliegen: ${speed?Abbruch}
```

Aufruf: `test5 speed=schnell`

Ausgabe: `Flugzeuge fliegen: schnell`

Aufruf: `test5`

Ausgabe: `test5: speed: Abbruch`

- Parameter durch feste Werte ersetzen:
`${parameter+ wort}` oder `${parameter:+ wort}`

Für die Angabe `+` gilt:

Ist ein **parameter** gesetzt, wird der Wert von **wort** eingesetzt. Ist kein **parameter** gesetzt, passiert garnichts. Geben Sie `:+` an, wird zusätzlich abgefragt, ob `parameter = ""` ist.

Beispiel

Inhalt der Datei *test6*:

```
echo Diese Datei ist ein: ${name+Dateiverzeichnis}
```

```
echo Sie haben für das Dateiverzeichnis: $name keine Leseerlaubnis
```

Aufruf: `test6 name=Kundendatei`

Ausgabe: `Diese Datei ist ein: Dateiverzeichnis`

`Sie haben für das Dateiverzeichnis: Kundendatei keine Leseerlaubnis`

Aufruf: `test6`

Ausgabe: `Diese Datei ist ein:`

`Sie haben für das Dateiverzeichnis.....keine Leseerlaubnis`

3.6.2 Shell-Prozeduren und Prozesse

Dieser Abschnitt beschreibt Ihnen die Auswirkungen des SINIX-Prozeßkonzepts, wenn Sie mit der Shell arbeiten, d.h. wenn Sie Kommandos eingeben oder mit Shell-Prozeduren arbeiten.

Ein laufendes SINIX-System (Ein- oder Mehrplatz), an dem Benutzer arbeiten, besteht aus einer baumartigen Struktur von *Prozessen* (siehe Bild 3-8).

Ein- oder Mehrplatzsysteme haben ein prinzipiell gleiches Prozeßkonzept. Einziger Unterschied: Da an einem Einplatzsystem nur ein Benutzer arbeiten kann, gibt es nur eine Login-Shell und damit auch nur einen Prozeßbaum.

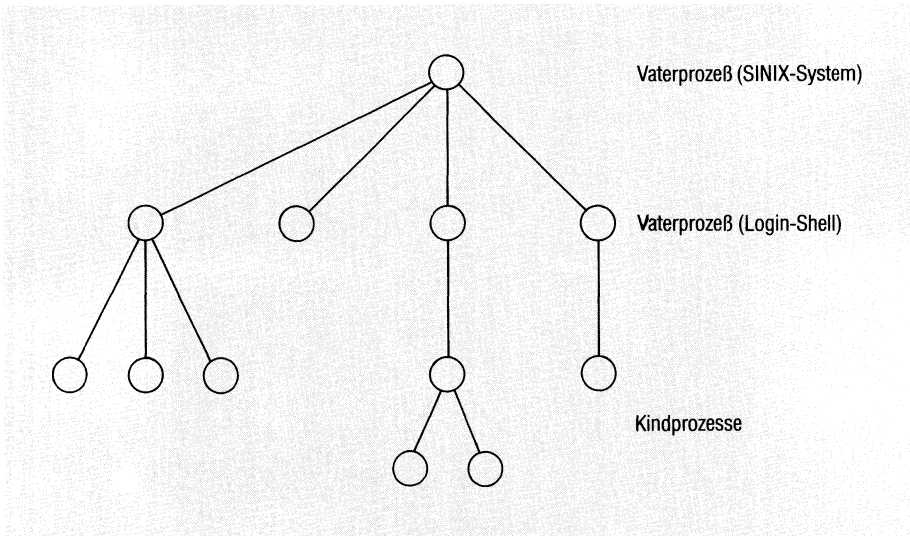


Bild 3-8 SINIX-Prozeßstruktur

Beim Starten des Systems wird ein Vaterprozeß mit der Prozeßnummer 1 (PID 1) erzeugt. Er ist der Ursprung für alle weiteren Prozesse im SINIX-System. Direkt an ihn angeschlossen ist für jeden Benutzer ein Kindprozeß, in dem seine Login-Shell abläuft. Alle Eingaben, die ein Benutzer macht, nimmt die Login-Shell in Empfang und führt entsprechende Aktionen aus. Bis auf wenige Ausnahmen (siehe: Kommandos der Shell), wird für jedes eingegebene Kommando ein eigener Prozeß erzeugt. Der Prozeß in dem die Login-Shell abläuft, wird damit selbst zu einem Vaterprozeß.

Grundsätzlich gilt: Jeder Prozeß kann einen neuen Prozeß erzeugen. Solange ein Prozeß keinen anderen Prozeß erzeugt, ist er der Kindprozeß eines Vaterprozesses. Sobald ein Kindprozeß einen anderen Prozeß erzeugt, wird er selbst vom Kind- zum Vaterprozeß.

Die Vater-Kind-Prozeßhierarchie müssen Sie beachten, wenn Sie in Shell-Prozeduren Variablen weiterreichen wollen. Warum? Jeder Benutzer kann für seine Login-Shell (= Vaterprozeß) ein Umfeld definieren, d.h. Variablen festlegen, mit denen die Login-Shell arbeiten soll (siehe auch: Abschnitt 3.6 Shell-Prozeduren). Dieses Umfeld ist aber grundsätzlich erst einmal nur der Login-Shell bekannt. Soll in einem Kindprozeß mit Variablen aus dem Umfeld des Vaterprozesses (= Login-Shell) gearbeitet werden, muß dieses Umfeld an den Kindprozeß exportiert werden. Dazu muß man das Kommando `export` benutzen (siehe: Abschnitt 3.8.7). Dieser Mechanismus gilt sinngemäß auch für Prozesse, die von einem Kindprozeß aufgerufen werden.

Beachten Sie: Variablen lassen sich durch Exportieren nur vom Vater- an einen Kindprozeß weiterreichen. Will man vom Kind- an seinen Vaterprozeß Variablen zurückgeben, geht das nur mit "Tricks".

Ein Beispiel dafür, wie man Variablen "zurückgeben" kann

Aus der Datei */etc/passwd* sollen die Benutzerkennungen herausgefiltert und auf den Bildschirm geschrieben werden. Dazu soll aus einer Prozedur (Name: *start*) eine andere Prozedur (Name: *filter*) aufgerufen werden, in der die Benutzerkennungen herausgefiltert werden. Die Benutzerkennungen sollen als Variablen an die aufrufende Prozedur zurückgegeben und von ihr ausgegeben werden.

Inhalt von *start*:

```
xyz=`filter`  
echo $xyz
```

Inhalt von *filter*:

```
IFS=:  
exec </etc/passwd  
while read lines  
do    set $lines  
      echo $1  
done
```

Was passiert, wenn man die Prozedur *start* aufruft? Eingabe: *sh start*
Die Prozedur *filter* wird gestartet und ihr Ergebnis wird der Variablen *xyz* zugewiesen. Das sind die gesuchten Benutzerkennungen.

Was passiert in der Prozedur *filter*? Als erstes wird durch die Variable *IFS* das Feldtrennzeichen definiert auf das Zeichen *:* (Doppelpunkt). Das ist notwendig, um später auf die einzelnen Elemente jeder Zeile in der Datei */etc/passwd* zugreifen zu können. Durch das Kommando *exec* wird die Standard-Eingabe auf die Datei */etc/passwd* umgeleitet. Das in der *while*-Schleife angegebene *read*-Kommando liest zeilenweise von der Standard-Eingabe, d.h. aus der Datei */etc/passwd*, und weist das Ergebnis der Variablen *lines* als Wert zu. Das Kommando *set* macht diese Werte so verfügbar, daß man auf sie über Stellungsparameter zugreifen kann. Das *echo*-Kommando liest mit dem Stellungsparameter *\$1* das erste Element. Das ist die gesuchte Benutzerkennung, die als Ergebnis der Prozedur *filter* an die Prozedur *start* übergeben wird.

Shell-Prozeduren im Hintergrund ablaufen lassen

Wenn man nach dem Aufruf einer Prozedur nicht auf die Beendigung der Prozedur warten will, kann man sie im Hintergrund ablaufen lassen. Dazu muß man den Prozeduraufruf mit folgendem Zeichen *abschließen*: **&**. Jetzt bekommt man unmittelbar nach dem Aufruf der Prozedur, die Prozeßnummer des Prozesses ausgegeben, in dem die Prozedur im Hintergrund abläuft.

Anschließend erscheint am Bildschirm das Eingabeaufforderungszeichen. Man kann sofort weitere Kommandos eingeben und braucht nicht auf die Beendigung des Hintergrundprozesses warten. Die ausgegebene Prozeßnummer braucht man, um mit dem Kommando **kill** den Hintergrundprozeß eventl. abbrechen zu können (siehe auch: **#!**, Variablen für die Shell).

3.7 Ablaufanweisungen für die Shell

Mit den Anweisungen:

break	(Schleifen steuern)
case	(Abfragen und Verzweigen)
continue	(Schleife erneut ausführen)
for	(Liste in Schleifen abarbeiten)
if	(Abfragen und Kommando ausführen)
while	(Schleife mit Abbruchbedingung)

können Sie eine Shell-Prozedur strukturieren. Alle Ablaufanweisungen können Sie auch im Dialog von der Datensichtstation aus eingeben. Wie Sie innerhalb einer Ablaufanweisung eine Kommando-Liste angeben können, ist im Abschnitt 3.1 beschrieben.

3.7.1 Schleifen steuern: Die break- und continue-Anweisung

Formate der Anweisungen:

```
break [n]      continue [n]
```

Mit der break-Anweisung kann man eine z.B. **while**- oder **for**-Schleife beenden. Mit der continue-Anweisung gibt man an, daß die Schleife noch einmal durchlaufen werden soll. Beide Anweisungen funktionieren nur, wenn sie zwischen **do** und **done** stehen.

Eine break-Anweisung beendet die Schleife, in der sie angegeben wird. Die Prozedur wird hinter dem nächsten, nicht erreichten done, fortgesetzt. Will man n Schleifen beenden, muß man eingeben: break n.

Eine continue-Anweisung führt noch einmal die Schleife aus, in der sie gegeben wurde, d.h. die Verarbeitung geht an den Anfang einer for-, while- oder until-Schleife zurück. Bei verschachtelten Schleifen kann man auch bei der n-ten einschließenden Schleife fortsetzen; dazu muß man angeben: continue n.

Beispiel für die break- und continue-Anweisung:

Eine Prozedur soll ein gewünschtes Dateiverzeichnis nach leeren Dateien durchsuchen. Sobald die erste leere Datei gefunden wird, soll eine Meldung am Bildschirm erscheinen.

Name der Prozedurdatei sei *konti*. Der Inhalt sieht so aus:

```
letzte=`ls $1 | tail -1`      #letzte Datei im Dateiverzeichnis merken
cd $1
for x in *
do
if test -d $x
then if test $x = $letzte
    then echo "Das Dateiverzeichnis $1 enthält keine leeren Dateien"
    else continue
    fi
elif test ! -s $x
then echo "Das Dateiverzeichnis $1 enthält leere Dateien"
break
else if test $x = $letzte #ist das die letzte Datei im Dateiverzeichnis?
    then echo "Das Dateiverzeichnis $1 enthält keine leeren Dateien"
fi
fi
done
```

3.7.2 Abfragen und Verzweigen: Die case-Anweisung

Mit der case-Anweisung kann man eine Eingabe prüfen und entsprechend dem Ergebnis etwas ausführen.

Die Anweisung case hat das allgemeine Format:

```
case name in  
muster) Kommando-Liste;;  
.  
.  
.  
muster) Kommando-Liste  
esac
```

Die unterstrichenen Begriffe sind Konstanten der case-Anweisung.

Die Shell vergleicht den für *name* stehenden Wert mit den unter *muster* stehenden Werten.

Dafür gelten dieselben Konventionen wie für Dateinamen. Bei einer Übereinstimmung zwischen *name* und *muster* wird eine hinter *muster* stehende Kommando-Liste ausgeführt. Bis auf die letzte Kommando-Liste muß jede andere mit den Zeichen ;; abgeschlossen sein. Es kann immer nur ein *muster* ausgeführt werden. Die *muster* werden der Reihe nach geprüft.

Beachten Sie: Wird das erste *muster* mit * angegeben, wird kein weiteres *muster* mehr geprüft. Es ist möglich, vor eine Kommando-Liste mehrere *muster* zu setzen. Mehrere *muster* müssen durch | voneinander getrennt sein.

Beispiel für die Anweisung case:

Es soll entsprechend einer bestimmten Eingabe jeweils ein Text ausgegeben werden. Die Prozedurdatei *meldung* soll dazu folgendes enthalten:

```
case $1 in
    alex) echo "alex ist klein";;
    pascal) echo "pascal ist jung";;
    peter|werner) echo "$1" ist klug";;
    *) echo "Diese Parametereingabe ist nicht erlaubt"
esac
```

Eingabe: meldung alex
Ausgabe: alex ist klein

Eingabe: meldung pascal
Ausgabe: pascal ist jung

Eingabe: meldung peter oder meldung werner
Ausgabe: peter ist klug oder werner ist klug

Jede andere Eingabe z.B.

Eingabe: meldung auto

erzeugt die

Ausgabe: Diese Parametereingabe ist nicht erlaubt

3.7.3 Liste in Schleifen abarbeiten: Die for-Anweisung

Mit der for-Anweisung kann man für jeden Eintrag in einer Liste, die gleiche Kommando-Liste ausführen. Mit der for-Anweisung kann man z.B. mehrere unterschiedliche Dateien mit dem gleichen Kommando bearbeiten.

Die Anweisung for hat das allgemeine Format:

```
for name [in Liste]  
do Kommando-Liste  
done
```

Die unterstrichenen Begriffe sind Konstanten der for-Anweisung. Um erkannt zu werden, müssen sie entweder (außer in) am Beginn einer Zeile oder nach einem ;-Zeichen stehen.

name Name einer Variablen, die erzeugt werden soll (z.B. x oder eric). Dieser Variablen werden nacheinander die Werte zugewiesen, die in der hinter in angegebenen Liste stehen. Mit diesen Werten kann man in der Kommando-Liste als Kennwortparameter arbeiten.

Liste Dafür kann man eine Liste von Werten angeben, die durch Leerzeichen voneinander getrennt sein müssen. Für jeden Wert in dieser Liste wird die hinter do angegebene Kommando-Liste ausgeführt. Fehlt die Liste-Angabe, wird \$@ als Standard angenommen, d.h. es werden alle vorhandenen Stellungsparameter (\$1, \$2, \$3, usw.) eingesetzt.

Kommando-Liste

Dafür kann man eine abzuarbeitende Kommando-Liste angeben.

done Damit muß jede for-Schleife abgeschlossen sein.

Beispiele für die Anweisung for:

Angenommen, Sie haben 3 Dateien mit Kundennamen *kunde1*, *kunde2*, *kunde3*. Durch eine for-Schleife können Sie alle 3 Dateien nach gleichen Kundennamen durchsuchen. Erstellen Sie dazu als erstes eine Prozedurdatei, z.B. *kundenname*, mit folgendem Inhalt:

```
for x in kunde1 kunde2 kunde3  
do grep $1 $x  
done
```

Wenn Sie jetzt folgende Eingabe machen:

```
kundenname huber
```

passiert im Klartext folgendes: Der Variablen *x* wird der erste Wert aus der hinter in folgenden Liste zugewiesen, d.h. der Dateiname *kunde1*. Anschließend wird das nach do angegebene Kommando ausgeführt. Als erstes ersetzt die Shell die hinter dem *grep*-Kommando stehenden Parameter. Für *\$1* wird der erste Stellungsparameter aus der Kommandozeile eingesetzt (d.h. *huber*) und für *\$x* wird der erste Wert der der Variablen *x* zugewiesen wurde eingesetzt (d.h. der Dateiname *kunde1*).

Das *grep*-Kommando durchsucht jetzt die Datei *kunde1* nach dem Kundennamen *huber*. Anschließend wird die *for*-Schleife erneut durchlaufen.

Der Variablen *x* wird als Wert der Dateiname *kunde2* zugewiesen. Das *grep*-Kommando übernimmt mit *\$x* diesen Wert, d.h. die Datei *kunde2* wird nach dem Kundennamen *huber* durchsucht. Das gleiche passiert mit der Datei *kunde3*. Am Bildschirm bekommen Sie eine Auflistung darüber, welche der Dateien den Kundennamen *huber* enthält.

Das gleiche Ergebnis können Sie mit einer anderen Schreibweise einer *for*-Anweisung erreichen:

```
for eingabe
do grep $eingabe kunde1 kunde2 kunde3
done
```

Was ist daran anders? Die *for*-Zeile enthält keine Liste-Angabe. Dafür wird als Standardwert *\$@* eingesetzt. Alle in der Kommandozeile angegebenen Stellungsparameter werden jetzt "unsichtbar" an diese Stelle gesetzt.

Damit sieht die *for*-Zeile in diesem Fall eigentlich so aus:

```
for eingabe in $1 $2
```

Die Eingabe:

```
$ kundenname huber meier
```

bewirkt, daß der hinter *for* stehende Name *eingabe* auf *huber* gesetzt wird und das *grep*-Kommando *huber* übernimmt, um die 3 Dateien *kunde1*, *kunde2* und *kunde3* danach zu durchsuchen. Das Gleiche passiert mit dem Kundennamen *meier*.

Am Bildschirm wird aufgelistet, welche Dateien die Kundennamen *huber* oder *meier* enthalten.

3.7.4 Abfragen und Kommando ausführen: Die if-Anweisung

Mit der if-Anweisung kann man eine Kommando-Liste abarbeiten und entsprechend dem Ergebnis in zwei alternative Richtungen verzweigen. Wie kann das Ergebnis aussehen? Jedes Kommando meldet nach seiner Ausführung einen Ende-Status zurück, den die if-Anweisung abfragt. Der Ende-Status kann entweder null oder ungleich null sein. Null bedeutet *wahr*, ungleich null bedeutet *unwahr*. Ist der Ende-Status gleich null, wird in die folgende then-Abfrage verzweigt. Ist der Ende-Status ungleich null wird zum nächsten elif verzweigt. Mit elif kann man erneut eine Bedingung abfragen und entsprechend dem Ergebnis zum nächsten then oder else verzweigen.

Das Kommando if hat das allgemeine Format:

```
if   Kommando-Liste
then Kommando-Liste
elif Kommando-Liste
then Kommando-Liste
else Kommando-Liste
fi
```

Die unterstrichenen Begriffe sind Konstanten der if-Anweisung. Um erkannt zu werden, müssen sie entweder am Beginn einer Zeile oder nach einem ;-Zeichen stehen.

Beispiel für die Anweisung if:

Es soll abgefragt werden, ob ein Datei- oder ein Dateiverzeichnisname eingegeben wurde.

Bei Eingabe eines Dateinamens soll der Inhalt der Datei ausgegeben werden. Bei Eingabe eines Dateiverzeichnisnamens sollen die Einträge des Dateiverzeichnisses ausgegeben werden. Wird weder ein Datei- noch ein Dateiverzeichnisname eingegeben, soll eine Meldung am Bildschirm erscheinen. Name der Prozedurdatei *check*

Inhalt der Datei *check*:

```
if test -f "$1"
then cat $1
elif test -d "$1"
then (cd $1;ls -l)
else echo $1 "ist weder ein Datei- noch ein Dateiverzeichnisname"
fi
```

Angenommen, es gibt eine Datei mit dem Namen *richter* und ein Dateiverzeichnis mit dem Namen *kundendatei*, dann kann man mit der Prozedurdatei *check* wie folgt arbeiten:

Eingabe: *check richter*

Ausgabe: ...Inhalt der Datei *richter*

Eingabe: *check kundendatei*

Ausgabe: ...Alle Einträge des Dateiverzeichnisses *kundendatei*

Eingabe: *check*

Ausgabe: ...Alle Einträge des HOME-Dateiverzeichnisses. Warum? Weil dann an das in der Prozedurdatei stehende *cd*-Kommando kein Parameter übergeben wird und es dann automatisch das HOME-Dateiverzeichnis annimmt (siehe auch: Abschnitt 3.5, HOME-Variable).

3.7.5 Schleife mit Abbruchbedingung: Die while- und until-Anweisung

Mit der while-Anweisung kann man den Ende-Status eines Kommandos oder einer Kommando-Liste auf null (d.h. *wahr*) und mit der until-Anweisung auf ungleich null (d.h. *unwahr*) abfragen. Anschließend kann man eine zweite Kommando-Liste solange ausführen, wie die Bedingung erfüllt ist. Der Ende-Status einer while- oder until-Anweisung ist der des letzten Kommandos in der zweiten Kommando-Liste.

Die Anweisung **while** fragt den Ende-Status *null* ab und hat das allgemeine Format:

```
while  Kommando-Liste  
do    Kommando-Liste  
done
```

Die unterstrichenen Begriffe sind Konstanten der while-Anweisung. Um erkannt zu werden, müssen sie entweder am Beginn einer Zeile oder nach einem ;-Zeichen stehen.

Die Kommandos in der nach while stehenden Kommando-Liste werden ausgeführt. Solange die hinter while stehende Kommando-Liste den Ende-Status null (d.h. *wahr*) meldet, wird die hinter do stehende Kommando-Liste ausgeführt. Diese Kommando-Liste wird solange in einer Schleife ausgeführt, wie die erste Kommando-Liste den Ende-Status null meldet.

Beispiel für die Anweisung while:

Es soll die Datei *datei1* in die Dateien *datei2* *datei3* und *datei4* kopiert werden. Die angenommene Prozedurdatei *Kopiere* muß folgenden Inhalt haben:

```
while test "$2" != ""
do    cp $1 $2
    shift
done
```

Durch die Eingabe:

```
$ Kopiere datei1 datei2 datei3 datei4
```

wird nun die *datei1* in die Dateien *datei2*, *datei3* und *datei4* kopiert. Diese while-Anweisung liest sich im Klartext so: Solange ("while") in der Kommandozeile unter Stellungsparameter \$2 ein Wert steht, führe die hinter do angegebene Kommando-Liste aus. Der Stellungsparameter \$2 wird durch das test-Kommando abgefragt. Solange der Stellungsparameter \$2 ungleich null ist, ist die im test-Kommando abgefragte Bedingung erfüllt, d.h. das test-Kommando hat den Ende-Status null. Da die while-Anweisung auf den Ende-Status null abfragt, wird die hinter do stehende Kommando-Liste ausgeführt. Nach dem ersten Lesen der Kommandozeile steht für \$1 der Dateiname *datei1* und für \$2 der Dateiname *datei2*. Die while-Bedingung ist erfüllt, also kopiert das cp-Kommando die *datei1* in *datei2*. Anschließend verschiebt das shift-Kommando die Werte in der Kommandozeile.

Unter dem Stellungsparameter \$1 steht dann *datei2*, unter \$2 *datei3* und unter \$3 *datei4*. Die while-Bedingung ist immer noch erfüllt, denn der Wert unter \$2 ist immer noch ungleich null. Also kopiert das cp-Kommando \$1 nach \$2. In diesem Fall *datei2* nach *datei3*. Da in *datei2* beim ersten Durchlauf der Inhalt von *datei1* kopiert wurde, steht in *datei3* ebenfalls der Inhalt von *datei1*. Dieser Ablauf wiederholt sich solange, bis das test-Kommando unter \$2 null liest, die Abfragebedingung deshalb nicht mehr erfüllt ist und es den Ende-Status eins (d.h. *unwahr*) meldet.

while und until

Die Anweisung **until** fragt den Ende-Status auf *ungleich null* ab und hat das allgemeine Format:

```
until  Kommando-Liste  
do    Kommando-Liste  
done
```

Die unterstrichenen Begriffe sind Konstanten der until-Anweisung. Um erkannt zu werden, müssen sie entweder am Beginn einer Zeile oder nach einem ;-Zeichen stehen.

Die Kommandos in der nach until stehenden Kommando-Liste werden ausgeführt. Solange ("until") diese Kommando-Liste einen Ende-Status ungleich null meldet, wird die hinter do stehende Kommando-Liste ausgeführt. Die hinter do stehende Kommando-Liste wird solange ausgeführt, wie die erste Kommando-Liste einen Ende-Status ungleich null meldet.

Beispiel für die Anweisung until:

Angenommen, Sie wollen aus einem Dateiverzeichnis alle leeren Dateien löschen. Vorausgesetzt, Sie sind der Eigentümer des Dateiverzeichnisses und der entsprechenden Dateien, dann können Sie das durch eine Prozedur machen, die eine until-Anweisung enthält. Die Prozedurdatei *loesche* muß dann folgendes enthalten:

```
for x in *
do until test -s "$x"
do      rm $x
      echo Die Datei $x wurde geloescht
      break
done
done
```

Was passiert, wenn Sie jetzt die Prozedurdatei *loesche* aufrufen?

Eingabe: sh loesche

Die for-Anweisung stellt der until-Anweisung jeden Eintrag des aktuellen Dateiverzeichnisses zur Abfrage zur Verfügung. Das test-Kommando hinter der until-Anweisung fragt ab: "Die Datei existiert und ist nicht leer". Handelt es sich um eine leere Datei, wird der Ende-Status 1 gemeldet (d.h. *unwahr*). Die Abfragebedingung für die until-Anweisung ist erfüllt und die folgenden Kommandos werden ausgeführt. Das rm-Kommando löscht die durch \$x gelesene Datei. Das echo-Kommando gibt eine Meldung aus, welche Datei gelöscht wurde und das break-Kommando beendet die until-Schleife. Durch das for-Kommando wird der nächste Eintrag gelesen, an die until-Anweisung zur Abfrage übergeben usw..

—

—

—

—

3.8 Kommandos der Shell

Die Shell stellt neben den im Buch SINIX-Kommandos beschriebenen Kommandos, folgende "eingebaute" Kommandos und Anweisungen zur Verfügung:

break	case	cd	continue	eval
exec	exit	export	for	if
newgrp	read	readonly	set	sh
shift	times	trap	umask	until
wait	while			
.	: und #	()	{ }	[...]

Mit den Anweisungen: break, continue, case, for, if und while kann man eine Shell-Prozedur steuern (vgl. Abschnitt 3.7). Die Kommandos cd, sh, newgrp, umask und wait, sind auch im Buch SINIX-Kommandos beschrieben. Alle übrigen Shell-Kommandos sind hier im Kapitel 3 beschrieben, beginnend mit den Sonderzeichen; für das . -Kommando siehe unter 'punkt'.

Der Unterschied zwischen "eingebauten" Kommandos bzw. Anweisungen und allen anderen Kommandos ist der, daß die Shell "eingebaute" Kommandos und Anweisungen selbst interpretiert und verarbeitet. Alle anderen Kommandos führt die Shell ungeprüft aus. Für "eingebaute" Kommandos oder Anweisungen erzeugt die Shell keinen eigenen Prozeß, deshalb sind sie schneller. Diese Kommandos geben Sie genauso ein, wie alle anderen Kommandos.

3.8.1 : Ein Kommando, das "nichts" macht

Das Leere-Kommando tut "nichts" und sein Ende-Status ist 0 (d.h. *wahr*).

Das Leere-Kommando hat das Format:

```
:
```

Beispiel für das :-Kommando:

Man kann das Kommando `:` in einer Prozedur dazu benutzen, um "nichts" zu machen, um z.B. den Zweig einer `if`- oder `case`-Anweisung zu füllen.

```
if test -f $1
then :
else echo "Guten Tag"
fi
```

In diesem einfachen Beispiel wird jede Eingabe, die kein Dateiname ist, mit der Meldung *Guten Tag* quittiert.

3.8.2 # Kommentare einfügen

Mit dem Zeichen `#` kann man in einer Prozedur Kommentarzeilen schreiben.

Das `#`-Zeichen verlangt das Format:

```
# ␣
```

Beispiel für einen Kommentar:

```
if test -f $1      # Ist die Eingabe ein Dateiname?
```

Hinweis

Beachten Sie, daß dem `#` ein *Leerzeichen* folgen muß. Der Kommentar endet mit dem Zeilenende.

3.8.3 () und {} Kommandos zusammenfassen

Durch runde und geschweifte Klammern kann man Kommandos zusammenfassen. Die Auswirkungen sind unterschiedlich. Für runde und geschweifte Klammern gilt: Ihr Ende-Status ist der des zuletzt ausgeführten Kommandos.

Stehen Kommandos zwischen zwei runden Klammern (), erzeugt die Shell für diese Kommandofolge eine Sub-Shell (d.h. einen eigenen Prozeß). Was kann man damit anfangen? Das kann man beim Arbeiten mit Variablen ausnutzen. Werden Kommandos in einer Sub-Shell ausgeführt, werden Variablen in der eigenen aktuellen Shell davon nicht berührt. Verändert man also in einer Sub-Shell Variablen (die z.B. vorher von der aktuellen Shell an die Sub-Shell übergeben wurden), haben diese Änderungen keine Rückwirkung auf die aktuelle Shell.

Beispiel für runde Klammern:

Beispiel 1: Angenommen, man hat für das Umfeld der aktuellen Shell die Variable *name=meier* definiert. Durch folgende Eingabe sieht man, daß diese Variable in einer Sub-Shell verändert werden kann ohne daß sie deshalb in der aktuellen Shell verändert wird:

```
(echo $name;name=richter;echo $name);echo $name
```

Die Ausgabe sieht so aus:

```
meier
richter
meier
```

Was wurde gemacht? Die Shell liest die erste runde Klammer und erzeugt eine Sub-Shell. Das erste echo-Kommando liest *\$name* und gibt *meier* aus. Die Angabe *name=richter* ändert die Variable *name*, sie wird durch das zweite echo-Kommando ausgegeben. Die zweite runde Klammer beendet die Sub-Shell, man ist wieder in der eigenen aktuellen Shell. Dort liest das dritte echo-Kommando die Variable *name* und gibt *meier* aus.

() und {}

Beispiel 2: Runde Klammern kann man auch noch dazu benutzen, um Kommandos in einem anderen als dem eigenen Dateiverzeichnis auszuführen, ohne daß man dafür extra einen Rücksprung angeben muß.

Die Eingabe `(cd ../ls -l)`

gibt alle Einträge des Dateiverzeichnisses aus, das dem eigenen Dateiverzeichnis übergeordnet ist. Anschließend befindet man sich automatisch wieder in seinem aktuellen Dateiverzeichnis.

Die Eingabe `cd ../ls -l`

hätte zwar die gleiche Ausgabe zur Folge, gleichzeitig würde man jedoch auch das Dateiverzeichnis wechseln.

Stehen Kommandos zwischen geschweiften Klammern { }, werden die Ausgaben aller Kommandos zusammengefaßt. Die Summe dieser Ausgaben kann man zur Weiterverarbeitung an ein anderes Kommando übergeben. Die eingeschlossenen Kommandos werden von der Shell gelesen und verarbeitet.

Arbeitet man mit geschweiften Klammern, muß die öffnende Klammer als erstes Zeichen in der Kommandozeile angegeben werden, gefolgt von einem Blank. Vor einer schließenden geschweiften Klammer muß ein Semikolon stehen. Eine Eingabe für geschweifte Klammern:

`{ ls -l;ls -l ..;}|grep meier`

Was wird gemacht? Mit dieser Eingabe werden alle Dateien ausgegeben, die *meier* heißen oder *meier* gehören. Durchsucht wird dafür das aktuelle und das übergeordnete Dateiverzeichnis.

3.8.4 eval Kommandos übergeben und ausführen

Mit dem eval-Kommando kann man der Shell Kommandos übergeben und ausführen. Ausgaben des eval-Kommandos lassen sich nicht umleiten. Es wird eine Kommando- und / oder Parameter-Substitution durchgeführt.

Das Kommando eval hat das Format:

```
eval [Kommando]
```

Beispiel für das Kommando eval:

Dateiname: testeval

Inhalt : kommando1=who
kommando2=';date'
eval \$kommando1 \$kommando2

Aufruf : testeval

Ausgabe : 1) Alle z.Zt. ans System angeschlossenen Benutzer
2) Das aktuelle Datum

Was wurde gemacht? Die in der eval-Kommandozeile angegebenen Kennwortparameter *\$kommando1* und *\$kommando2* wurden ersetzt durch *who;date*. Diese beiden Kommandos hat die Shell ausgeführt und deren Ausgabe auf die Standard-Ausgabe geschrieben.

3.8.5 exec Kommando ausführen und Shell ersetzen

Mit dem Kommando exec kann man:

- ein Kommando oder eine Shell-Prozedur ausführen, wodurch die Shell ersetzt wird.
- die Standard-Eingabe für eine Shell-Prozedur auf eine Datei umlenken.

Das Kommando exec hat das Format:

```
exec name
```

name Name einer Shell-Prozedur, eines Kommandos oder einer Datei.

Beispiele für das Kommando exec:

Beispiel 1: Was passiert, wenn man das exec-Kommando im Dialog von der Datensichtstation aus eingibt?

```
exec date
```

Durch das Kommando exec wird als erstes die Shell beendet und anschließend das Kommando date ausgeführt. Am Bildschirm erscheint die Ausgabe des aktuellen Datums. Da die Shell beendet wurde, sind Sie anschließend nicht mehr mit dem System verbunden. Sie werden automatisch aufgefordert, ihre Benutzerkennung einzugeben.

Beispiel 2: Was passiert, wenn man mit dem Kommando `exec` in einer Prozedur eine andere Prozedur aufruft? Angenommen, es gibt folgende Dateien:

Datei-1:

Name: prozedur1

Inhalt: echo prozedur1 hat die Prozessnummer \$\$
sh prozedur2

Datei-2:

Name: prozedur2

Inhalt: echo prozedur2 hat die Prozessnummer \$\$
exec prozedur3

Datei-3:

Name: prozedur3

Inhalt: echo prozedur3 hat die Prozessnummer \$\$

Wenn jetzt `prozedur1` gestartet wird durch:

```
sh prozedur1
```

bekommt man am Bildschirm eine Ausgabe, die so aussehen kann:

```
prozedur1 hat die Prozessnummer 16834
prozedur2 hat die Prozessnummer 16840
prozedur3 hat die Prozessnummer 16840
```

Was zeigt diese Ausgabe? Sie zeigt, daß *prozedur1* unter einer eigenen Prozeßnummer läuft und *prozedur2* sowie *prozedur3* unter einer gemeinsamen Prozeßnummer ablaufen. Warum? Weil *prozedur3* aus der *prozedur2* heraus durch `exec` aufgerufen wurde. Deshalb wird für *prozedur3* kein neuer Prozeß erzeugt, er bekommt die gleiche Prozeßnummer wie *prozedur2*.

Beachten Sie, daß eine durch das Kommando `exec` aufgerufene Prozedur als *ausführberechtigt* gekennzeichnet sein muß (siehe: `chmod`-Kommando).

Beispiel 3: Dieses Beispiel zeigt Ihnen, wie man mit dem Kommando `exec` die Standard-Eingabe für eine Prozedur auf eine Datei umlenkt. Eine Prozedur soll ihre Eingabe also nicht mehr von der Datensichtstation, sondern aus einer Datei lesen.

Der Name der Prozedur sei: *extest*. Wenn Sie für diese Prozedur eine Eingabe von der Datensichtstation lesen wollen, müssen Sie folgendes in sie eintragen:

```
read baba
echo $baba
```

Was passiert, wenn Sie die Prozedur aufrufen? Eingabe: `sh extest`. Die Prozedur wird durch das Kommando `read` gestoppt und wartet auf eine Eingabe von Ihnen, die Sie an der *Datensichtstation* eingeben müssen. Angenommen, Sie geben *auto* ein, dann wird diese Eingabe dem hinter `read` angegebenen Variablennamen (*baba*) als Variablenwert zugewiesen. Durch das Kommando `echo` wird dieser Wert am Bildschirm ausgegeben. Es erscheint *auto*.

Wollen Sie für die gleiche Prozedur, eine Eingabe anstatt von der Datensichtstation, aus einer *Datei* lesen, müssen Sie folgendes in die Datei *extest* eintragen:

```
exec </etc/passwd
read baba
echo $baba
```

Wenn Sie jetzt diese Prozedur aufrufen, passiert folgendes. Durch das Kommando `exec` wird die Standard-Eingabe für die Prozedur *extest* auf die Datei */etc/passwd* umgelenkt, d.h. das dem `exec`-Kommando folgende `read`-Kommando liest jetzt nicht mehr von der Datensichtstation, sondern aus der Datei */etc/passwd*. Konkret liest es den ersten Satz und weist ihn der Variablen *baba* als Wert zu. Das `echo`-Kommando gibt diesen Satz aus. Im Normalfall müßten das die Benutzer-Einträge für den Systemverwalter sein.

3.8.6 exit Beenden einer Shell-Prozedur

Mit dem Kommando exit kann man eine Shell-Prozedur beenden und wahlweise einen Ende-Status setzen. Als Ende-Status kann man eine Zahl von 0 bis 32768 angeben.

Das Kommando exit hat das Format:

```
exit [n]
```

Wie arbeitet das exit-Kommando?

Das exit-Kommando beendet immer die Shell, in der es angegeben wird. Was heißt das? Gibt man an der Datensichtstation z.B. ein Kommando ein, um eine Shell-Prozedur zu starten, wird von der Login-Shell dafür eine Sub-Shell gestartet. Die Sub-Shell verarbeitet die aufgerufene Shell-Prozedur. Wird in der aufgerufenen Prozedur ein exit-Kommando ausgeführt, wird die Sub-Shell beendet und man befindet sich wieder in der Login-Shell. Den Mechanismus Shell / Sub-Shell muß man verstehen, um einen durch das exit-Kommando gesetzten Ende-Status anwenden zu können.

Was ist der Ende-Status? Es gibt zwei Möglichkeiten. Wurde ein Kommando erfolgreich ausgeführt, wird eine 0 (*wahr*) gesetzt. Konnte ein Kommando nicht erfolgreich ausgeführt werden, wird eine Ziffer ungleich null (*unwahr*) gesetzt. Beendet man eine Prozedur durch das exit-Kommando, kann man einen beliebigen Ende-Status setzen. Den Ende-Status kann man auswerten. Wie? Mit Hilfe des Zeichens: \$? .

Beispiel für das Kommando exit:

Probieren Sie einmal folgendes: Lesen Sie Ihr Dateiverzeichnis

Eingabe: `ls`

Ausgabe: Einträge des aktuellen Dateiverzeichnisses

Wenn Sie jetzt den Ende-Status des `ls`-Kommandos abfragen, durch die Eingabe:

`echo $?`

erhalten Sie als Ausgabe: 0, d.h. das `ls`-Kommando wurde erfolgreich ausgeführt. Machen Sie jetzt einmal irgendeine "Phantasieeingabe", z.B. :

`unsinn`

dann erscheint als Ausgabe: `unsinn: not found` (nicht gefunden)

Wenn Sie jetzt den Ende-Status abfragen, durch die Eingabe:

`echo $?`

erhalten Sie als Ausgabe: 1, d.h. die "Phantasieeingabe": `unsinn` konnte nicht ausgeführt werden. Probieren Sie jetzt einmal folgende Eingaben hintereinander:

1. Eingabe: `unsinn`
Ausgabe: `unsinn: not found`
2. Eingabe: `echo $?`
Ausgabe: 1
3. Eingabe: `echo $?`
Ausgabe: 0

Dieses Beispiel zeigt, daß für die Shell-Variable: `$?` immer der Ende-Status des zuletzt ausgeführten Kommandos eingetragen wird. Deshalb kann man den von der "Phantasieeingabe" erzeugten Ende-Status auch nur einmal abfragen. Anschließend wird für `$?` der Ende-Status der zweiten Eingabe gesetzt, d.h. der des erfolgreich ausgeführten `echo`-Kommandos. Die dritte Eingabe hat demzufolge die Ausgabe: 0 und nicht mehr die 1 des "Phantasiekommandos". Will man einen Ende-Status auswerten, muß man darauf achten, daß zwischen Setzen des Ende-Status und Auswertung, er durch kein anderes Kommando verändert wird.

Wie kann man den Ende-Status in einer Prozedur auswerten?

Als erstes, ein Beispiel dafür, wie es nicht geht. In einer Prozedur soll abgefragt werden, ob eine Eingabe ein Datei- oder Dateiverzeichnisname ist. Entsprechend dem Abfrageergebnis soll durch ein Kommando exit verzweigt werden. Die Prozedur sieht so aus und ihr Name sei *abfrage*:

```
Inhalt der Datei abfrage
if test -f "$1"
then exit 2;
elif test -d "$1";
then exit 3;
else exit 4;
fi
case $? in
  2) cat $1;echo "Das war der Inhalt der Datei: "$1;;
  3) cd $1;ls -l;echo "Das war das Dateiverzeichnis: "$1;;
  4) echo "Die Eingabe hat für diese Prozedur keinen Sinn";;
esac
```

Was passiert, wenn man diese Prozedur aufruft?

Gibt man z.B. ein: *abfrage kunde* (*kunde* soll der Name einer Datei sein), passiert folgendes: Die Login-Shell erzeugt für diese Eingabe eine Sub-Shell um die Prozedur auszuführen. Die Sub-Shell fragt durch das if-Kommando die Datei *kunde* ab und beendet sich durch das Kommando: *exit 2*. Der Ende-Status der Prozedur ist auf *\$?=2* gesetzt. Weil die Sub-Shell durch das *exit*-Kommando beendet wurde, wird das dem if-Kommando folgende *case*-Kommando nicht mehr ausgeführt. Der Ende-Status *\$?=2* wird in dieser Prozedur nicht mehr weiterverarbeitet. Den Ende-Status kann man jetzt nur noch im Dialog an der Datensichtstation abfragen (z.B. durch das *echo*-Kommando) oder weiterverarbeiten. Wobei man, wie bereits erwähnt, aufpassen muß, daß man ihn nicht durch die Eingabe eines anderen Kommandos verändert.

exit

Was kann man tun, um den Ende-Status in einer Datei auswerten zu können?

Antwort: Man setzt im obigen Beispiel das if-Kommando in runde Klammern. Die Prozedur sieht dann so aus:

```
Inhalt der Datei abfrage
  (if test -f "$1"
  then exit 2;
  elif test -d "$1";
  then exit 3;
  else exit 4;
  fi)
  case $? in
    2) cat $1;echo "Das war der Inhalt der Datei: "$1;;
    3) cd $1;ls -l;echo "Inhalt des Dateiverzeichnisses: "$1;;
    4) echo "Eingabe hat keinen Sinn für diese Prozedur"
  esac
```

Was ist daran anders? Für ein in runde Klammern gesetztes Kommando wird eine Sub-Shell erzeugt (siehe auch: Abschnitt 3.8.3).

Welche Auswirkungen hat das in unserem Beispiel? Macht man wieder die Eingabe: `$ abfrage kunde`, erzeugt die Login-Shell als erstes wieder eine Sub-Shell, um die Prozedur *abfrage* auszuführen. Diese Sub-Shell liest als erstes die runde Klammer und erzeugt dadurch eine weitere Sub-Shell, um das if-Kommando auszuführen. Die "zweite" Sub-Shell wird durch das Kommando `exit 2` beendet. Der Ende-Status wird auf `$? = 2` gesetzt. Dieser Ende-Status wird an die "erste" Sub-Shell weitergereicht. Da durch das `exit`-Kommando nur die "zweite" Sub-Shell und nicht gleichzeitig die "erste" Sub-Shell beendet wurde, wird nun als nächstes von der "ersten" Sub-Shell das dem if-Kommando folgende case-Kommando ausgeführt. Das case-Kommando kann dann den Ende-Status `$? = 2` auswerten. Wie man an diesem Beispiel sieht, kann man in einer Prozedur einen durch ein `exit`-Kommando gesetzten Ende-Status, nur in der aufrufenden und nicht in der den Ende-Status erzeugenden Sub-Shell weiterverarbeiten.

Eine andere Möglichkeit dieses Beispiel zu realisieren, kann so aussehen:
Man schreibt die beiden Kommandos `if` und `case` in zwei Dateien eines Dateiverzeichnisses:

Datei 1

Inhalt der Datei *abfrage1*:

```
if test -f "$1"
then exit 2;
elif test -d "$1";
then exit 3;
else exit 4;
fi
```

Datei 2

Inhalt der Datei *abfrage2*:

```
abfrage1 $1
case $? in
  2) cat $1;echo "Das war der Inhalt der Datei: "$1;;
  3) cd $1;ls -l;echo "Inhalt des Dateiverzeichnis: "$1;;
  4) echo "Die Eingabe hat für diese Prozedur keinen Sinn"
esac
```

Ruft man jetzt die Datei: *abfrage2* zusammen mit dem Namen einer Datei als Parameter auf (z.B. durch: `abfrage2 kunde`), erzeugt die Login-Shell als erstes wieder eine Sub-Shell um die Prozedur *abfrage2* auszuführen. Diese Sub-Shell liest die erste Zeile (Inhalt: `abfrage1 $1`). Nachdem die Prozedur *abfrage1* ebenfalls in einer neuen Sub-Shell ausgeführt wurde, bekommt die erste Sub-Shell den Ende-Status `$?=2` zurückgemeldet (er wurde durch `exit 2` in der Prozedur: *abfrage1* gesetzt). Mit diesem Ende-Status wird jetzt in der Prozedur *abfrage2* das `case`-Kommando ausgeführt.

3.8.7 export Variablen weiterreichen

Mit dem export-Kommando kann man Variablen von einer Shell an eine Sub-Shell weiterreichen.

Das Kommando export hat das Format:

```
export [variable]
```

Beispiel für das Kommando export:

Angenommen, Sie haben in Ihrer Login-Shell eine Variable definiert, z.B. durch die Eingabe:

```
gruss = hallo
```

Auf diese Variable können Sie jetzt z.B. mit Kommando echo zugreifen:

```
Eingabe: echo $gruss
```

```
Ausgabe: hallo
```

Warum funktioniert das? Weil der Login-Shell die Variable *\$gruss* bekannt ist und deshalb von ihr substituiert werden kann. Schreiben Sie das gleiche echo-Kommando in eine Datei, z.B.:

```
Inhalt der Datei text:
    echo $gruss
```

Rufen Sie diese Datei auf, durch die Eingabe:

```
text
```

Es wird nichts ausgegeben. Warum? Weil die Login-Shell eine so aufgerufene Datei (und damit das in ihr stehende echo-Kommando) immer in einer Sub-Shell ausführt. Dieser Sub-Shell ist jedoch die Variable *\$gruss* nicht bekannt, weil sie nur für die Login-Shell definiert wurde. Was tun? Man macht die Variable systemweit bekannt, indem man sie durch das Kommando export "exportiert".

```
export gruss
```

Nach dieser Eingabe wird in der Prozedur *text* die Variable *\$gruss* ersetzt. Die Eingabe: *text* erzeugt die Ausgabe: hallo.

Hinweis 1

Im Dialog eingegebene Variablen sind nach dem Abmelden vom System gelöscht. Will man Variablen dauerhaft definieren, muß man sie in die Datei *.profile* schreiben und exportieren. Durch die Eingabe: export erhält man eine Liste von allen als exportierbar deklarierten Variablen, sowie eine Liste der Variablen, die "nur lesbar" sind (siehe auch: readonly-Kommando).

Durch die Eingabe: set, kann man sich eine Liste der "aktuellen Umwelt" ausgeben lassen (siehe auch: set-Kommando).

Hinweis 2

Über weitere Möglichkeiten Variable zu exportieren, informiert Sie Abschnitt 3.6.1 .

3.8.8 punkt Datei starten ohne neuen Prozeß zu erzeugen

Mit dem Punkt-Kommando veranlassen Sie, daß die Shell eine Datei ausführt, ohne daß sie dafür einen eigenen Prozeß erzeugt. An eine, durch das Punkt-Kommando auszuführende Datei, kann man aus der Kommandozeile nur Kennwort-Parameter übergeben.

Das Kommando punkt hat das Format:

```
. dateiname
```

dateiname Name einer Datei, die ausgeführt werden soll.

Das Punkt-Kommando bietet sich an, wenn Sie z.B. in der Datei *.profile* Variablen verändern und diese Änderungen an die Shell übergeben möchten.

Beispiel für das .-Kommando:

Eingabe:

```
..profile
```

Was passiert? Die aktuellen (eventl. geänderten) Variablen der Datei *.profile* werden an die Shell übergeben.

Achtung

Das Punkt-Kommando ist etwas anderes als ein Punkt vor einem Dateinamen (z.B. *.profile*).

3.8.9 read Prozedur anhalten und etwas einlesen

Mit dem read-Kommando kann man eine Prozedur anhalten, um sie entsprechend einer Eingabe von der Standard-Eingabe aus, weiterlaufen zu lassen.

Das Kommando read hat das Format:

```
read [name...]
```

name Dafür kann man einen oder mehrere Namen angeben, denen in der Standard-Eingabe angegebene Eingaben zugewiesen werden. Auf **name** kann man durch `$name` zugreifen.

Wie arbeitet das read-Kommando? Das read-Kommando liest eine Zeile der Standard-Eingabe. Die einzelnen Werte der Eingabe werden der Reihe nach den Variablen: **name** zugewiesen. Überzählige Werte in der Standard-Eingabe werden der letzten Variablen des read-Kommandos zugewiesen.

Beispiel für das Kommando read:

Inhalt der Prozedurdatei *readtest*:

```
echo Bitte geben Sie Parameter ein:
read kunde1 kunde2 kunde3
echo $kunde1
echo $kunde2
echo $kunde3
```

Eingabe: sh readtest

Ausgabe: Bitte geben Sie Parameter ein:

Eingabe: Meier Miller Beck

Ausgabe: Meier
Miller
Beck

Was wurde gemacht? Die Prozedur *readtest* wurde gestartet und nach der Ausgabe des Textes des ersten echo-Kommandos gestoppt. Die Eingabe der Worte *Meier Miller Beck* wurde vom read-Kommando gelesen und den Namen *kunde1 kunde2 kunde3* zugewiesen. Anschließend wurde jeder Name mit einem \$-Zeichen aufgerufen und durch ein echo-Kommando ausgegeben. Gibt man für die gleiche Prozedur nachfolgendes ein, so erhält man eine andere Ausgabe:

Eingabe: Meier Miller Beck Eric Joe

Ausgabe: Meier
Miller
Beck Eric Joe

Warum? Der letzten Variablen des read-Kommandos (*kunde3*) werden jetzt die überzähligen Werte der Standard-Eingabe zugewiesen und damit auch vom Kommando: echo \$kunde3 ausgegeben.

3.8.10 readonly Variablen schützen

Mit den readonly-Kommando kann man Variablen als "nur lesbar" deklarieren. Die Werte dieser Variablen lassen sich durch Zuweisungen nicht mehr ändern. Wird readonly alleine eingegeben, bekommt man eine Liste aller readonly-Variablen ausgegeben.

Das Kommando readonly hat das Format:

```
readonly [variable ... ]
```

3.8.11 set und sh Schalter für die Shell setzen

Durch das set-Kommando kann man Schalter setzen, die das Verhalten der Shell ändern. Besonders die beiden Schalter -x und -v können sehr nützlich sein. Diese beiden Schalter kann man auch wieder zurücksetzen - alle anderen nur durch Beenden der Sitzung. Außerdem kann man die Reihenfolge von in der Kommandozeile stehenden Angaben ändern oder eine Kommandozeile neu aufbauen (siehe auch: Abschnitt 3.6.1 sowie: Beispiele).

Das Kommando set hat das Format:

```
set [-xvektu] [name]
```

Schalter kann man z.B. so eingeben: set -xv

Gibt man das set-Kommando alleine ein (Eingabe: set), bekommt man alle Variablen ausgegeben, die für die aktuelle Shell gesetzt sind.

Bedeutung der Schalter:

- v Alle Eingabe-Zeilen, die von der Shell gelesen werden, erscheinen am Bildschirm. Mit diesem Schalter kann man Syntax-Fehler finden. Die Kommandos jeder Zeile werden ausgeführt, nachdem diese Zeile ausgegeben ist.
- x Die Kommandos und ihre Parameter werden so ausgegeben (mitgeschrieben), wie sie ausgeführt werden (Shell-Kommandos, wie z.B. for oder while werden allerdings nicht ausgegeben).
- e Die Shell wird sich sofort beenden (exit), sobald ein von ihr ausgeführtes Kommando mit einem Ende-Status ungleich null abschließt.
- k Diesen Schalter muß man setzen, wenn man in einer Kommandozeile hinter dem Prozeduraufruf stehende Kennwortparamater übergeben will.
- n Dieser Schalter überprüft eine Prozedur auf Syntaxfehler, ohne die Prozedur-Kommandos auszuführen.
- t Die Shell soll die Kommandos der aktuellen Eingabezeile lesen, ausführen und sich anschließend beenden.

-u Ist dieser Schalter gesetzt, behandelt die Shell eine nicht zugewiesene Variable als Fehler, sobald sie den Wert dieser Variable substituieren soll. Mit diesem Schalter kann man Variable global prüfen.

— Schaltet die -x und -v Schalter aus.

name Angaben für die Kommandozeile (siehe Beispiele). Diese Angaben gelten nur für die Prozedur, in der man sie angibt, d.h. sie sind nicht exportierbar (siehe: export-Kommando).

Die obenstehenden Schalter kann man auch alle beim Aufruf einer neuen Shell (durch das sh-Kommando) direkt mit angeben.

Schalter **nur für sh-Kommando:**

-c pfadname

Mit diesem Schalter kann man der Shell einen Dateinamen oder Pfadnamen einer ausführbaren Datei mitteilen, von dem sie Kommandos lesen soll.

-s Ist dieser Schalter gesetzt, liest die Shell Kommandos von der Standard-Eingabe. Ausgaben erfolgen über Dateideskriptor 2 (Standard-Fehlerausgabedatei) auf den Bildschirm. Der -s Schalter ist für die nach Login geladene Shell automatisch gesetzt.

-i Die Shell befindet sich im Dialog, wenn dieser Schalter gesetzt oder die Ein-/Ausgabe der Datensichtstation zugeordnet ist. Von so einer Shell wird das Signal 2 (INTERRUPT) angenommen aber nicht bewertet.

Die Signale 3 (QUIT) und 15 (TERMINATE) werden ignoriert.

Beispiele für das Kommando set:

Beispiel 1: An die Prozedurdatei *settest* soll eine Kommandozeile übergeben werden. Die Reihenfolge der Angaben in der Kommandozeile soll in der Prozedur geändert werden.

Inhalt der Prozedurdatei *settest*:

```
echo $1 $2 $3
set $3 $2 $1
echo $1 $2 $3
```

Aufruf: *settest* SINIX IST TOLL
Ausgabe: SINIX IST TOLL
TOLL IST SINIX

Was wurde gemacht? Beim Aufruf der Prozedurdatei: *settest* wurde an die Prozedur übergeben: SINIX IST TOLL. Durch das erste echo-Kommando wurde diese Zeichenfolge ausgegeben. Durch das set-Kommando wurde die Reihenfolge der Angaben in der Kommandozeile geändert in: TOLL IST SINIX und anschließend durch das zweite echo-Kommando ausgegeben.

Beispiel 2: An die Prozedurdatei *settest1* soll eine Kommandozeile übergeben werden. Der Inhalt der Kommandozeile soll in der Prozedur durch einen anderen Inhalt ersetzt werden.

Inhalt der Prozedurdatei *settest1*:

```
echo $1 $2 $3
set WER IST DA
echo $1 $2 $3
```

Aufruf: *settest1* HALLO IHR DORT
Ausgabe: HALLO IHR DORT
WER IST DA

Was wurde gemacht? Die Prozedur wurde gestartet durch die Eingabe: *settest*. Das erste echo-Kommando liest die Eingabe: HALLO IHR DORT und gibt sie aus. Das set-Kommando ersetzt den Inhalt der Kommandozeile durch: WER IST DA. Das zweite echo-Kommando liest die Kommandozeile und gibt den neuen Inhalt aus.

Beispiel 3: Die Eingabe für das in der Prozedurdatei *settest2* stehende Kommando *read*, soll als Stellungsparameter verfügbar gemacht werden.

⌋ Inhalt der Prozedurdatei *settest2*:

```
read namen
set $namen
echo $1 $2 $3
echo $3 $1 $2
echo $2 $3 $1
```

Aufruf: *settest2*
Eingabe: *Miller Beck Plant*
Ausgabe: *Miller Beck Plant*
Plant Miller Beck
Beck Plant Miller

⌋ Was wurde gemacht? Die Prozedur wurde gestartet durch die Eingabe: *settest2*. Das *read*-Kommando erwartet eine Eingabe von der Tastatur. Eingegeben wurde *Miller Beck Plant*. Diese Eingabe wurde als Wert der Variablen *namen* zugewiesen. Das *set*-Kommando macht diesen Wert durch *\$namen* im Variablen- und Parameterbereich der Prozedur als Stellungsparameter verfügbar (siehe auch: Abschnitt 3.6.1). Die drei *echo*-Kommandos gaben entsprechende Kombinationen aus.

3.8.12 shift Verschieben von Operanden

Am Bildschirm kann man in der Kommandozeile maximal 9 Stellungspareparameter (\$1-\$9) direkt angeben und abarbeiten. Will man mehr als 9 Stellungspareparameter bearbeiten, kann man das mit dem Kommando shift machen.

Das Kommando shift verschiebt die Zuordnung der Werte zu den Stellungspareparametern nach links. Der Wert von \$1 fällt heraus und \$1 erhält den Wert von \$2, \$2 erhält den Wert von \$3, \$3 den Wert von \$4 usw.. \$0 wird durch das shift-Kommando nicht beeinflusst.

Das Kommando shift hat das Format:

```
shift
```

Beispiele für das Kommando shift:

Beispiel 1

Inhalt der Datei *testshift*:

```
while test "$1" != ""
do
    echo $1
    shift;
done
```

Aufruf: `testshift 1 2 3 4 5 6 7 8 9 10 11`

Ausgabe: 1
2
3
4
5
.
.
11

Beispiel 2Inhalt der Datei *testshift*:

```
while test "$1" != ""
do
    echo $1 $2 $3 $4 $5 $6 $7 $8 $9
    shift;
done
```

Aufruf: testshift 1 2 3 4 5 6 7 8 9 10 11

Ausgabe:

```
1 2 3 4 5 6 7 8 9
2 3 4 5 6 7 8 9 10
3 4 5 6 7 8 9 10 11
4 5 6 7 8 9 10 11
5 6 7 8 9 10 11
6 7 8 9 10 11
7 8 9 10 11
8 9 10 11
9 10 11
10 11
11
```

Beispiel 3Inhalt der Datei *testshift*:

```
while test "$1" != ""
do
    echo $*
    shift;
done
```

Aufruf: testshift 1 2 3 4 5 6 7 8 9 10 11

Ausgabe:

```
1 2 3 4 5 6 7 8 9 10 11
2 3 4 5 6 7 8 9 10 11
3 4 5 6 7 8 9 10 11
4 5 6 7 8 9 10 11
5 6 7 8 9 10 11
6 7 8 9 10 11
7 8 9 10 11
8 9 10 11
9 10 11
10 11
11
```

times

3.8.13 times Prozeßzeiten auflisten

Für die von der Shell aus gestarteten Prozesse werden die aufsummierten Benutzer- und Systemzeiten ausgegeben.

Das Kommando times hat das Format:

```
times
```

3.8.14 trap Shell-Prozedur unterbrechen

Eine Shell-Prozedur kann durch das Betriebssystem oder von der Datensichtstation aus durch den Benutzer unterbrochen werden. Dazu muß mit dem kill-Kommando ein Signal an die Shell-Prozedur gesendet werden. Mit dem trap-Kommando kann man Unterbrechungssignale empfangen und darauf reagieren.

Das Kommando trap hat das Format:

```
trap Kommando-Liste Signal-Liste
```

Kommando-Liste

An diese Stelle kann man ein oder mehrere Kommandos schreiben, die ausgeführt werden sollen, wenn ein in der Signal-Liste stehendes Signal eintrifft. Die Shell liest die angegebene Kommando-Liste, sobald sie das erste Mal das Kommando trap liest. Das passiert beim Start der Shell-Prozedur. Damit die Kommando-Liste aber erst beim Eintreffen eines Signals ausgeführt wird und nicht bereits dann, wenn die Shell die Prozedur startet, muß man sie in Apostrophe eingeschlossen schreiben. Apostrophe verbieten der Shell eine sofortige Ausführung der Kommandos (siehe auch Kapitel: Der Apostrophier-Mechanismus). Das wird z.B. wichtig, wenn man temporäre Dateien löschen will. Denn zum Zeitpunkt zu dem die Shell das Kommando trap erstmals liest, stehen die Namen temporärer Dateien meist noch garnicht fest. Ist keine Kommando-Liste angegeben, wird für jedes in der Signal-Liste angegebene und empfangene Signal eine Standardroutine durchlaufen.

Ist die Kommando-Liste eine Leermenge, z.B.: () oder " ", werden die Signale in der Signal-Liste ignoriert. Damit kann man eine Shell-Prozedur gegen bestimmte Signale immun machen.

Signal-Liste

In der Signal-Liste kann man eine oder mehrere Signalnummern angeben. Trifft eines der definierten Signale ein, wird die Kommando-Liste ausgeführt. Welche Signalnummern man angeben kann, ist beim Kommando kill beschrieben.

Beispiel für das Kommando trap:

Angenommen, es gibt folgende Prozedur:

Inhalt der Datei *traptest*:

```
trap 'echo Prozedurabbruch durch ein Signal;exit' 2 3
while true
do echo hallo
done
```

Ruft man diese Prozedur durch die Eingabe: *traptest* auf, wird am Bildschirm solange die Meldung: hallo ausgegeben, bis das Signal 2 oder 3 eintrifft. Die Signale kann man von der Datensichtstation aus durch folgende Tasten senden:

- Die Tasten **CTRL** und **I** gleichzeitig gedrückt, erzeugen das Signal 3
- Die Taste **DEL** erzeugt das Signal 2

Startet man die Prozedur *traptest* als Hintergrundprozeß (Eingabe: *traptest&*), kann man sie von der Datensichtstation aus nur noch durch das *kill*-Kommando abbrechen. Dazu muß man die Prozeßnummer mit angeben. Die Prozeßnummer eines Hintergrundprozesses wird sofort nach seinem Start am Bildschirm ausgegeben. Die Eingabe des *kill*-Kommandos kann in eine laufende Ausgabe eingetippt werden. In unserem Beispiel z.B.: *kill -2* (Prozeßnummer). Funktionstasten haben bei Hintergrundprozessen keine Wirkung.

3.8.15 umask Standardeinstellung der Schutzbits ändern

Mit dem Kommando `umask` kann man die Standardeinstellung der Schutzbits ändern. Die geänderte Einstellung der Schutzbits wird dann einer neu erstellten Datei oder einem neu erstellten Dateiverzeichnis mitgegeben. Die Wirkung des Kommandos `umask` gilt nur für den aufrufenden Benutzer und für die Dauer einer Sitzung.

Das Kommando `umask` hat das Format:

```
umask [nnn]
```

`nnn` Dreistelliger oktaler Wert (siehe `chmod`-Kommando). Jedes `n` gilt für eine der Benutzergruppen: Eigentümer, Gruppe oder Andere. Standardeinstellung: 066

Geben Sie das `umask`-Kommando ohne Parameter ein, wird die Standardeinstellung ausgegeben.

Wie funktioniert das `umask`-Kommando?

Das binäre Komplement der `umask`-Angabe wird der Standardeinstellung der Schutzbits logisch UND verknüpft und ergibt damit die neue Schutzeinstellung.

Beispiel für das Kommando `umask`:

Die Eingabe `umask 333`

setzt die Standardeinstellung der Schutzbits für eine Datei auf:

```
r--r--r--
```

Warum? Die oktale Eingabe 333 entspricht binär: 011011011.

Das Komplement davon ist: 100100100.

Wenn Sie jetzt eine neue Datei erstellen, wird dieses Komplement durch ein logisches UND zur alten Standardeinstellung addiert.

Die Standardeinstellung für eine Datei ist: `rw-rw-rw-`, das entspricht binär: 110110110. Die Addition ergibt:

```

110110110   (Standardeinstellung für eine Datei)
& 100100100 (Komplement der umask-Angabe)
-----
100100100

```

Binär 100100100 entspricht folgender Einstellung: `r--r--r--`.

3.8.16 wait Auf Prozeßabschluß warten

Das wait-Kommando wartet auf die Beendigung aller vor ihm laufenden Prozesse.

Das Kommando wait hat das Format.

```
wait [_prozeßnummer...]
```

prozeßnummer Nummer eines oder mehrerer Prozesse, auf deren Beendigung wait warten soll.

Beispiel für das Kommando wait::

Inhalt der Prozedurdatei *waittest*:

```
who&
date&
wait
echo Die Prozedur ist beendet
```

Aufruf: sh waittest

Ausgabe: (aktuelles Datum)
 (Liste aller angeschlossenen Benutzer)
 Die Prozedur ist beendet

Was wurde gemacht? In der Prozedur *waittest* wurden das *who*- und *date*-Kommando als Hintergrundprozesse gestartet. Durch das wait-Kommando wurde in der Prozedur solange gewartet, bis beide Hintergrundprozesse beendet waren. Anschließend wurde das Kommando *echo* ausgeführt.

Würde man die gleiche Prozedur ohne das wait-Kommando ablaufen lassen, würde am Bildschirm als erstes die Meldung des *echo*-Kommandos ausgegeben: Die Prozedur ist beendet. Warum? Weil *who* und *date* als Hintergrundprozesse ablaufen und das Kommando *echo* sofort ausgeführt wird.

3.9 Übersicht - Sonderzeichen der Shell

Zeichen	Bedeutung
>	<i>kdo</i> > <i>datei</i> : Umlenkung, <i>kdo</i> -Standardausgabe auf <i>datei</i>
>>	<i>kdo</i> >> <i>datei</i> : Anhängen, <i>kdo</i> -Standardausgabe an <i>datei</i>
<	<i>kdo</i> < <i>datei</i> : Umlenkung, <i>kdo</i> -Standardeingabe von <i>datei</i>
<<wort	<i>kdo</i> <<wort: Die Standardeingabe besteht aus den nächsten Zeilen bis zur ersten Zeile, die nur <i>wort</i> enthält.
*	Textmuster, beliebige Zeichenkette für Dateiname
?	Textmuster, ein beliebiges Zeichen
[...]	Textmuster, je ein Zeichen aus Angabe-Menge; [3c9] [a-z] etc.
[!...]	Textmuster, je nicht ein Zeichen aus der Menge [von-bis] etc.
;	Kommando-Begrenzer
&	<i>kdo</i> & startet <i>kdo</i> als Hintergrundprozeß
`kdo`	führt <i>kdo(s)</i> aus und ersetzt es durch seine Standardausgabe
(kdo)	führt Kommando(s) <i>kdo</i> in einer Subshell aus
{_kdo;}	führt <i>kdo(s)</i> in akt. Shell aus, faßt Std-Ausgaben zusammen
\$1,\$2,..	Stellungsparameter (Argumente) einer Shell-Prozedur
\$var	Wert einer Shellvariablen <i>var</i>
\${var}	Wie <i>\$var</i> ; wenn der Var-Name im Kontext nicht eindeutig ist
\	Hebt Sonderbedeutung des nachfolgenden Zeichens auf
'...'	Keine Sonderbedeutung aller eingeschlossenen Zeichen
"..."	Shell erkennt nur \$ `...' \ und interpretiert Zeichenkette
#	Leitet Kommentar ein, (Kommentar-Ende ist Zeilenende)
var=wert	Variable erhält Wert
&&	<i>kdo1</i> && <i>kdo2</i> <i>kdo1</i> erfolgreich: <i>kdo2</i> -Ausführung
	<i>kdo1</i> <i>kdo2</i> <i>kdo1</i> nicht erfolgreich: <i>kdo2</i> -Ausführung

—

—

—

—

4 Die SINIX-Kommandos

In diesem Kapitel sind die SINIX-Kommandos in Übersichten zusammengestellt, die Sie als "Benutzer" verwenden können. Sie können sich hier einen groben Überblick verschaffen über

- generelle Anwendungsmöglichkeiten der Kommandos
- die Namen der Kommandos
- die Verwandtschaft der Kommandos bezüglich ihrer Anwendung

Die detaillierten Beschreibungen von Kommandos finden Sie dort:

Kommandos für Shell-Prozeduren	Kapitel Shell in diesem Buch
Kommandos für Benutzer	Buch SINIX-Kommandos (Teil 1 und Teil 2)
Kommandos für Systemverwaltung	Buch SINIX Systemverwaltung
Menüs für Systemverwaltung	Buch SINIX Systemverwaltung
Dienstprogramme und Werkzeuge zur C-Programmierung	Buch SINIX Kommandos (Teil 1 und Teil 2)
Datei-Formate Include-Dateien Geräte-Dateien	CES C-Entwicklungssystem (Teil 1)
Systemaufrufe C-Funktionen und Makros	CES C-Entwicklungssystem (Teil 2)

Shell-Anweisungen (siehe Kapitel Shell) unterstützen Sie bei diesen Aufgaben:

- Generierung von Dateinamen
- Umlenkung der Ein- und Ausgabe
- Hintergrundprozesse
- Kommentare in Shell-Prozeduren
- Verwendung von Variablen
- Prozedur- bzw. Funktionsaufrufe
- Definition und Aufruf von Makros

Wenn Sie das entsprechende Buch eben nicht griffbereit haben, können Sie sich über die sog. *On-line-Dokumentation* zu den Kommandos, mit dieser Eingabe (kdo steht hier für den Kommandonamen)

`$ man kdo | pg`

eine in SINIX gespeicherte, englische Dokumentation am Bildschirm ausgeben lassen.

Aus traditionellen Gründen, sind z.B. in der On-Line-Dokumentation, wie auch in anderer Literatur, direkt nach dem Kommandonamen, Angaben in Klammern angefügt, z.B. `sh(1)`. Das sind Zuordnungsmerkmale, die auch im X/OPEN-Standard übernommen wurden.

(1) sind "Standard-Utilities.

(1D) sind z.B. "Development-Utilities",

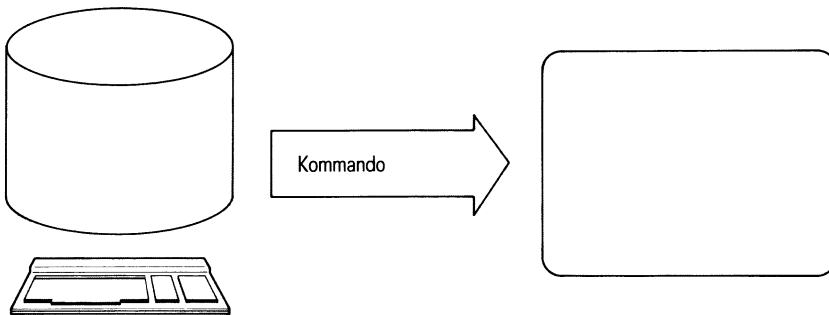
die man ursprünglich nur mit dem Lieferumfang des C-Entwicklungssystems bekam. Jetzt sind sie aber bereits in Ihrem System enthalten.

Im Anhang dieses Buches sind die Kommandos mit dem jeweiligen Suffix aufgeführt.

4.1 Was Sie zu jedem Kommando wissen sollten

Ein Kommando, das eine Eingabe erwartet, liest diese von der Standard-Eingabe oder von Dateien, deren Namen Sie so angeben müssen, wie es bei der jeweiligen Kommandobeschreibung angegeben ist. Ein Kommando gibt aus auf die Standard-Ausgabe.

Standard-Eingabe, Standard-Ausgabe



Standard-Eingabe ist die Tastatur

Standard-Ausgabe ist der Bildschirm

Bild 4-1 Standard-Eingabe, Standard-Ausgabe

Umlenken der Eingabe oder Ausgabe und Pipelines

Mit dem Umlenken der Eingabe oder Ausgabe und mit Pipelines (vgl. Erklärungen im Kapitel Shell), schaffen Sie sich mehr Möglichkeiten, z.B.:

- Ein Kommando, das normalerweise von der Tastatur liest, kann auch aus einer Datei lesen oder seine Eingabe von einem vorhergehenden Kommando bekommen.
- Ein Kommando, das normalerweise auf den Bildschirm ausgibt, kann auch in eine Datei ausgeben oder das Ergebnis an ein weiteres Kommando übergeben.

Gibt ein Kommando auf den Bildschirm aus und ist die Ausgabe länger als eine Bildschirmseite, dann können Sie:

- Kommando `pg` verwenden, z.B. `ls | pg`
- Ausgabe anhalten mit `CTRL S` und fortsetzen mit `CTRL Q`

Dateinamen

Wenn Sie Dateinamen oder Namen von Dateiverzeichnissen angeben, haben Sie immer mehrere Möglichkeiten:

- Sie geben einen einfachen Namen an. Damit beziehen Sie sich auf Namen, die im aktuelle Dateiverzeichnis stehen.
- Sie geben einen (absoluten oder relativen) Pfadnamen an. Damit können Sie beliebige Dateien und Dateiverzeichnisse im ganzen Dateisystem benutzen, vorausgesetzt, Sie haben die Zugriffsrechte.
- Sie können die abgekürzte Schreibweise von Dateinamen verwenden, besonders um mehrere Dateien gleichzeitig anzugeben (vgl. Kapitel Dateisystem).

Ende-Status:

Jedes Kommando liefert einen Ende-Status, der aussagt, wie das Kommando abgelaufen ist. Der Ende-Status steht als Zahlenwert in der Variablen: `?`. Sie können ihn z.B. mit `"echo $?"` abfragen.

Ist bei der Beschreibung eines Kommandos weiter nichts angegeben, dann ist der Ende-Status:

	0	bei fehlerfreiem Ablauf
ungleich	0	bei fehlerhaftem Ablauf
	3	bei Abbruch des Kommandos mit der Taste DEL

Fehlermeldungen

Zu jedem Kommando gibt es Fehlermeldungen. Sie sind weitgehend selbsterklärend. Fehlermeldungen gehen auf die *Standard-Fehlerausgabe*. Standard-Fehlerausgabe ist normalerweise der Bildschirm. Sie können sie aber *umlenken* mit `2 > & datei`.

Fehlermeldungen des Betriebssystems deuten meist auf einen hardwarebedingten fehlerhaften Ablauf. Als Maßnahme können Sie die Meldung ignorieren und es nochmals versuchen. Wiederholt sich der Fehler mehrmals, verständigen Sie den Systemkundendienst. Solche Fehler haben z.B. die Form:

```
ERR ON DEV 1/23
BN=.....
```


4.2 Kommandos eingeben, aber richtig

Wie eingeben ?

Am Bildschirm schreiben Sie Kommandos

- hinter die Eingabeaufforderung der Shell (\$)
- in eine Zeile
- und schließen die Eingabe mit  ab.

 ist für die Shell das Zeichen, alles Vorhergehende als Kommando zu interpretieren.

(Die Eingabeaufforderung der Shell ist standardmäßig das Zeichen \$. Es ist in der Standardvariablen PS1 als \$ hinterlegt.)

Mehrere Kommandos in einer Zeile

Sie können mehrere Kommandos, durch ; (Strichpunkt Leerzeichen) getrennt, in einer Zeile eingeben:

kommando1 ; kommando2 ; ... kommando 

Die Kommandos werden nacheinander ausgeführt, so als ob Sie sie einzeln eingegeben hätten. (Eine Verbindung von Ein- und Ausgaben wird nicht hergestellt.)

Wenn Sie mehrere Kommandos mit dem Zeichen | (senkrechter Strich) verbinden, wird die Ausgabe des ersten Kommandos als Eingabe für das folgende Kommando verwendet. Diese Verkettung, durch sog. Pipelines, ist im Kapitel Shell beschrieben.

Ist die Zeile zu kurz?

Schreiben Sie am Zeilenende einfach weiter, ohne  zu drücken.

Sie können aber auch in der nächsten Zeile weiterschreiben, wenn Sie mit \  auf die nächste Zeile schalten. Der Gegenschrägstrich (\) entwertet dabei die Kommandoabschlußfunktion der Taste . Ist die Eingabe vollständig, so schließen Sie diese mit der Taste  ab. Das Kommando wird dann ausgeführt.

Weitere Eingaben nach Kommandoaufruf

Manche Kommandos erwarten nach dem Abschluß mit  weitere Eingaben von der Tastatur. Das sehen Sie daran, daß die Schreibmarke auf den Beginn der nächsten Zeile gesetzt wird. Die Zeilen, die Sie nun eingeben, schließen Sie auch jeweils mit der Taste  ab. Beenden können Sie die Eingabe mit der Taste . Danach wird das Kommando fertig ausgeführt.

Eingabe von Schaltern

Schalter sind Wahlmöglichkeiten für die Wirkung des Kommandos. Für Schalter setzen Sie eine aktuelle Angabe ein. Meist sind verschiedene Buchstaben als Schalterangabe möglich. Welche, das finden Sie jeweils in der Beschreibung des Kommandos.

Schalter geben Sie in der Regel so an, daß Sie davor das Zeichen: (Bindestrich) angeben. Mehrere Schalter, z.B. -a, -b, -c, -d, können Sie auf zwei Arten angeben:

entweder *-a -b -c -d* oder *-abcd*, z.B.

ls -l ruft *ls* mit Schalter *l* auf,

ls -lig ruft *ls* mit Schaltern *l*, *i* und *g* auf.

In diesem Beispiel können Sie auch mehrere Namensangaben machen, z.B.

ls -l adam eva

Abgekürzte Dateinamen gelten auch als mehrere Einträge:

*ls -li c** (alle Dateien, die mit *c* beginnen).

(Schalter, die sich gegenseitig ausschließen, sollten Sie nicht gleichzeitig angeben. Solche Ausnahmen sind in den Kommandobeschreibungen im Buch SINIX-Kommandos angegeben.)

Darstellung der Kommandos (im Buch SINIX-Kommandos)

Alle Kommandos sind in dem hier beschriebenen Format dargestellt.

Beschrieben ist

unter *DEFINITION* jeweils die Syntax der Kommandos,

Unter *SCHALTER* die Syntax der Schalter und

unter *OPERANDEN* die Syntax der Operanden.

In der dort verwendeten Metasyntax bedeuten:

Zeichen (halbfett)

Diese Zeichen sind direkt so einzugeben, wie sie gedruckt sind.

Zeichen

Diese Zeichen stehen stellvertretend für andere Zeichen, die Sie auswählen und eingeben.

[] Alles, was zwischen den eckigen Klammern steht, können Sie weglassen. Welche Auswirkungen das hat, entnehmen Sie der Beschreibung für die Schalter und Operanden. Die eckigen Klammern selbst dürfen Sie nicht eingeben, es sei denn, es ist ausdrücklich verlangt.

_ Ein Leerzeichen, das Sie eingeben müssen.

... Die davorstehende Angabe können Sie mehrmals machen.

Beispiel für die Kommando-Darstellung

awk[_**F**][_**f**]programm[-parameter][_datei...]

Sie müssen eingeben:

- **awk**
- anstelle von *programm* ein Programm

Sie können zusätzlich eingeben:

- **-F** und anstelle von *c* irgendein Zeichen und/oder **-f**

Ob sich Schalter gegenseitig ausschließen, besagt ihre Beschreibung.

- anstelle von *parameter* die Parameter, die Sie an *programm* übergeben möchten
- anstelle von *datei...* einen oder mehrere Dateinamen, die als Eingabedatei verwendet werden soll(en).

Unter *Beispiel* sind die Eingaben unterstrichen angegeben, wenn sich Ein- und Ausgaben abwechseln. Eingaben, die alleine stehen und sofort als solche erkennbar sind, sind nicht eigens gekennzeichnet.

Beachten Sie:

- Bei manchen Kommandos gibt es Schalter mit vorangestelltem "-" und solche ohne "-" oder auch mit "+". Dann ist dieses Vorzeichen bei jedem Schalter beschrieben.
- Gehört zu einem Schalter ein Argument, dann ist die Syntax dafür jeweils eigens beschrieben, z.B. bedeutet `-f[_datei]`, daß Sie z.B. angeben können: `-f` oder `-f ausgabe`.
- Wenn Sie bei einem Kommando einen Schalter angeben der nicht beschrieben ist, bekommen Sie entweder eine Fehlermeldung oder es kann zu undefinierten Ereignissen kommen.

4.3 Anwendungsbereiche für SINIX-Kommandos

- Da SINIX für vielfältige Aufgabenlösungen auch viele Kommandos zur Verfügung stellt, sind hier vorerst die hauptsächlichen Funktionen zusammengestellt, für die Ihnen Kommandos zur Verfügung stehen.

Kommando-Interpreter Shell

Hilfskommandos für Shell-Prozeduren

Benutzer-Einträge abfragen und ändern

Benutzerumgebung abfragen und ändern

- Dateien bearbeiten, verwalten und ausgeben
 - Dateien drucken
 - Dateieigenschaften abfragen und ändern
 - sichern und archivieren
 - komprimiert speichern, Urzustand herstellen
 - Dateisystem verändern

Editoren

Zeichen einlesen, umwandeln und ausgeben

Prozesse steuern

Interprozeßkommunikation

Rechenfunktionen

- Kommunikation mit anderen Benutzern

Informationen abfragen

- Eigenschaften der Datensichtstation; auch ändern
- Prozeßdaten
- Systemdaten
- Speicherplatzbelegung
- Prozessortyp

On-Line-Dokumentation zu Kommandos

SCCS-Kommandos

- Kalenderfunktionen und Termine

C-Programme generieren, bearbeiten und verwalten

Makroprozessor für C-Sprache und andere

4.4 Alphabetische Liste der Kommandos

admin	SCCS-Dateien erstellen und verwalten	☾
ar	Bibliotheken erstellen und verwalten	
at	Kommando zu einer bestimmten Zeit ausführen (execute commands at a later time)	
att	Wechseln in das att-Universum	
awk	Dateien nach Mustern durchsuchen und bearbeiten	
banner	Argumente in großer Titelschrift ausgeben	
basename	Dateinamen vom Pfad trennen	☾
batch	Aufträge in Warteschlange; nacheinander abarbeiten	
bc	Arithmetische Sprache	
bdiff	Große Dateien zeilenweise vergleichen in <i>ed</i> -Skript	
bfs	Editor nur zum Durchsuchen von Dateien (Zeilen) (big file scanner)	
break	Schleifen steuern (siehe Kapitel Shell)	
bs	Übersetzer/Interpreter für Programme mittlerer Größe	
cal	Kalender ausgeben (print calendar)	
calendar	Erinnerungsdienst, Terminkalender	☾
case	Abfragen und Verzweigen (siehe Kapitel Shell)	
cat	Dateien aneinanderfügen und ausgeben (concatenate and print files)	
cb	C-Programme formatieren	
cc	C-Programme übersetzen	
cd	Dateiverzeichnis wechseln (change working directory)	☾
cdc	Kommentar bzw. MR-Nummern eines Deltas ändern	
ced	Bildschirmorientierter Editor	
cflow	Flußdiagramme für C-Programme generieren	

chgrp	Gruppennummer einer Datei ändern (change group)
chmod	Zugriffsrechte für eine Datei ändern (change mode)
chown	Eigentümer einer Datei ändern (change owner)
chroot	Root-Dateiverzeichnis für ein Kommando ändern (change root)
cmp	Dateien zeichenweise vergleichen (compare two files)
col	Filter für umgekehrte und halbe Zeilenvorschübe (filter reverse line-feeds)
comb	Deltas zusammenfassen
comm	Sortierte Dateien vergleichen (select lines common to two sorted files)
continue	Schleifen steuern (siehe Kapitel Shell)
cp	Datei kopieren (copy files)
cpio	Dateien und Dateiverzeichnisse ein- und auslagern (copy file archives in and out)
crontab	Kommandos automatisch wiederholt ausführen
csplit	Datei nach bestimmten Kriterien unterteilen (context split)
cut	Felder aus den Zeilen einer Datei herausschneiden (cut out selected fields of each line of a file)
cxref	Querverweisliste für C-Programme erstellen
date	Datum und Uhrzeit ausgeben
dc	Tischrechner (desk calculator)
dd	Dateien kopieren und konvertieren (copy device device)
delta	Delta erstellen
destroy	Datei physikalisch löschen
df	Dateisystem auf freien Platz prüfen (disk free)

Kommandos

diff	Dateien zeilenweise vergleichen in <i>ed</i> -Skript (differential file comparator)	
diff3	Drei Dateien zeilenweise vergleichen (three-way differential file comparison)	☾
dircmp	Dateiverzeichnisse vergleichen (directory comparison)	
dirname	Pfad vom Dateinamen trennen	
du	Belegten Speicherplatz ausgeben (display used blocks)	
echo	Zeichenfolgen ausgeben (echo arguments)	
ed	Zeilenorientierter Editor	
edit	Editor (Variante von ex für gelegentliche Benutzer)	☾
egrep	Erweiterte Muster suchen (extended grep)	
env	Umgebung bei Ausführung von Kommandos ändern (set environment for command execution)	
eval	Kommando-Übergabe an Shell (siehe Kapitel Shell)	
ex	Kommandoorientierter Editor	
exec	Kommando ausführen und Shell ersetzen (siehe Kapitel Shell)	
exit	Beenden einer Shell-Prozedur (siehe Kapitel Shell)	
export	Variablen weiterreichen (siehe Kapitel Shell)	
expr	Ausdrücke auswerten (evaluate expression)	☾
factor	Zahl in Primzahlen zerlegen (factor a number)	
false	Leeres Kommando mit Endestatus ungleich 0	
fgrep	Einfache Muster suchen (fast grep)	
file	Art einer Datei bestimmen (determine file type)	
find	Dateiverzeichnisse durchsuchen (find files)	
for	Liste in Schleifen abarbeiten (siehe Kapitel Shell)	☾
get	Version aus einer SCCS-Datei holen	
getopt	Parameter einer Prozedur nach Schaltern durchsuchen (get command options)	

grep	Muster in Dateien suchen (global regular expression print)
id	Benutzer- und Gruppenkennung ausgeben (print user and group IDs and names)
if	Abfragen und Kommando ausführen (siehe Kapitel Shell)
ipcrm	Kennung entfernen, Nachrichtenwarteschlange oder Semaphore (remove a message queue or semaphore id)
ipcs	Status von IPC-Einrichtungen (Interprozeßkommunikation ermitteln)
join	Dateien nach Vergleichsfeldern verbinden (join two files on identicalvalued field)
kill	Prozesse beenden, Signale senden (kill a processs, send signal to a process)
ld	Objektdateien zusammenbinden
lex	Programme zur lexikalischen Analyse generieren
line	Eine Zeile lesen (read one line)
lint	C-Programme überprüfen
ln	Verweis auf eine Datei eintragen (make a link)
login	Benutzerkennung wechseln
logname	Login-Name abfragen (get login name)
lorder	Objektmodule paarweise ordnen
lpr	Datei(en) drucken und Druckaufträge steuern
ls	Inf. über Dateiverzeichnisse und Dateien ausgeben (list contents of directory)
m4	Makroprozessor für C und andere Sprachen
machid	Wahrheitswerte über Art des Prozessors ausgeben
mail	Post senden und empfangen (send or receive mail)
mailx	Nachrichten interaktiv verarbeiten
make	Gruppen von Programmen verwalten
man	Engl. Einträge der Kommando-Dokumentation ausgeben

Kommandos

mesg	Ausgabe von Meldungen verhindern oder erlauben
mkdir	Dateiverzeichnis einrichten
mknod	FIFO-Datei erstellen (Pipe mit Name)
mv	Dateien umbenennen oder übertragen (move or rename files)
newform	Format einer Textdatei ändern
newgrp	Benutzergruppe wechseln (change to a new group)
nice	Priorität von Kommandos ändern
nl	Textzeilen numerieren (line numbering filter)
nm	Symboltabelle ausgeben
nohup	Kommando ausführen und dabei Signale ignorieren
ns32000	Prozessortyp bestätigen lassen (permit or deny messages)
newform	Format einer Textdatei ändern (new format of a text file)
od	Inhalt einer Datei oktal ausgeben (octal dump)
pack	Datei in komprimierter Form speichern
passwd	Kennwort für Benutzerkennung eintragen oder ändern (change login password)
paste	Zeilen zusammenfügen
pcat	Komprimiert gespeicherte Datei in Urform ausgeben
pg	Datei(en) seitenweise ausgeben
pr	Datei(en) zum Drucken aufbereiten (print files)
prof	Zeittabelle für ein C-Programm erstellen
prs	Informationen über eine SCCS-Datei ausgeben
ps	Prozeßdaten abfragen (process status)
punkt	Zeichen "Punkt" startet Datei (siehe Kapitel Shell)
pwd	Pfadnamen des aktuellen Dateiverzeichnisses ausgeben (path of working directory)

ranlib	Bibliothek mit einer Symboltabelle versehen
read	Prozedur anhalten und etwas einlesen (siehe Kapitel Shell)
readonly	Variablen-Schreibschutz (siehe Kapitel Shell)
red	Eingeschränkte Variante von Editor <i>ed</i>
regcmp	Reguläre Ausdrücke in die Sprache C übersetzen (regular expression compile)
rm	Dateien löschen (remove files)
rmdel	Delta löschen
rmdir	Dateiverzeichnisse löschen (remove directories)
sact	Ausgeben, welche Versionen anstehen
sccsdiff	Zwei Versionen vergleichen
sdiff	Zwei Dateien gleichzeitig ausgeben und vergleichen
sed	Editor im Prozedurbetrieb
set	Schalter für Shell setzen (siehe Kapitel Shell)
sh	Shell; Standardkommandointerpreter (siehe Kapitel Shell und Buch SINIX-Kommandos)
shift	Verschieben von Operanden (siehe Kapitel Shell)
sie	in das sie-Universum wechseln
size	Größe einer Objektdatetei ausgeben
sleep	Prozesse zeitweise stillegen
sort	Sortieren und Mischen von Dateien (sort and/or merge files)
spell	Rechtschreibfehler suchen (find spelling errors)
split	Datei auf mehrere Dateien aufteilen (split a file into pieces)
strip	Symboltabelle entfernen
stty	Eigenschaften einer Datensichtstation ändern
su	Benutzerkennung vorübergehend wechseln (become super-user or another user)

Kommandos

sum	Prüfsumme einer Datei berechnen (print checksum and block count of a file)
sync	Systempuffer zurückschreiben
tail	Endabschnitt einer Datei ausgeben
tar	Archivieren von Dateien auf Datenträger (file archiver)
tee	Ausgeben auf Standard-Ausgabe <i>und</i> in eine Datei
test	Bedingungen prüfen
tic	Informationen über Datensichtstation übersetzen
time	Laufzeit eines Kommandos messen (time a command)
times	Prozeß-Zeiten auflisten (siehe Kapitel Shell)
touch	Aktuelles Datum setzen für letzte Datei-Änderung
tput	Terminfo-Datenbank abfragen
tr	Zeichen durch andere ersetzen (translate characters)
trap	Shell-Prozedur unterbrechen (siehe Kapitel Shell)
true	Leeres Kommando mit Endestatus 0
tsort	Paarweise geordnete Objektmodule vollständig ordnen
tty	Namen der Datensichtstation ausgeben
umask	Standardeinstellung der Schutzbits ändern (siehe Kapitel Shell und Buch SINIX-Kommandos)
uname	Namen des aktuellen Systems ausgeben
unget	<i>get</i> -Kommando rückgängig machen
uniq	Mehrfache Zeilen suchen (unique lines)
units	Einheiten umrechnen
universe	das aktuelle Universum ausgeben
unpack	Komprimierte Dateien erweitern
until	Abfrage Kdo.-Ende-Status nicht null (siehe Kapitel Shell)
val	SCCS-Dateien auf Konsistenz prüfen

vc	Textdarstellung kontrollieren
vi	Bildschirmorientierter Editor
wait	Auf Prozeß-Ende warten (siehe Kapitel Shell und Buch SINIX-Kommandos)
wall	An alle Benutzer schreiben (write to all users)
wc	Zeilen, Wörter und Zeichen zählen (word count)
what	Versionsnummern ausgeben
while	Abfrage Kdo.-Ende-Status null (siehe Kapitel Shell)
who	Aktive Benutzerkennungen anzeigen (who is in the system)
write	Dialog mit anderem Benutzer (write to another user)
xargs	Argumentenliste(n) aufbauen und Kommando ausführen
yacc	Parser generieren

4.5 Welches Kommando für welche Aufgabe?

Die folgende Übersicht zeigt die Kommandos, gegliedert nach den hauptsächlichsten Funktionen. Manche Kommandos kommen mehrmals vor:

Kommando-Interpreter

sh(1) Shell

Shell-Anweisungen und Funktionen (siehe Kapitel Shell)

break	Schleifen steuern
case	Abfragen und Verzweigen
continue	Schleifen steuern
eval	Kommando-Übergabe an Shell
exec	Kommando ausführen und Shell ersetzen
exit	Beenden einer Shell-Prozedur
export	Variablen weiterreichen
for	Liste in Schleifen abarbeiten
if	Abfragen und Kommando ausführen
punkt	Zeichen "Punkt" startet Datei
read	Prozedur anhalten und etwas einlesen
readonly	Variablen-Schreibschutz
set	Schalter für Shell setzen
shift	Verschieben von Operanden
times	Prozeß-Zeiten auflisten
trap	Shell-Prozedur unterbrechen
until	Abfrage Kdo.-Ende-Status nicht null
while	Abfrage Kdo.-Ende-Status null

Benutzerumgebung abfragen bzw. ändern

att(1)	Ins att-Universum wechseln
cd(1)	Dateiverzeichnis wechseln
env(1)	Umgebung bei Ausführung eines Kommandos ändern
logname(1)	Login-Name abfragen
ls(1)	Inf. über Dateiverzeichnisse und Dateien ausgeben
pwd(1)	Pfadnamen des aktuellen Dateiverzeichnisses ausgeben
sie(1)	Ins sie-Universum wechseln
universe(1)	Aktuelle Arbeitsumgebung ausgeben
who(1)	Aktive Benutzerkennungen anzeigen

Dateien verwalten und bearbeiten

drucken

lpr(1) Datei(en) ausdrucken
pr(1) Datei(en) zum Drucken aufbereiten

ausgeben

cat(1) Datei(en) ausgeben
od(1) Dateiinhalt oktal ausgeben
pcat(1) Komprimiert gespeicherte Datei in Urform ausgeben
pg(1) Datei(en) seitenweise ausgeben
tail(1) Endabschnitt einer Datei ausgeben

bearbeiten

awk(1) Datei durchsuchen und bearbeiten
bdiff(1D) Große Dateien zeilenweise vergleichen in *ed*-Skript
cmp(1) Dateien zeichenweise vergleichen
comm(1) Sortierte Dateien vergleichen
csplit(1) Datei nach bestimmten Kriterien unterteilen
cut(1) Felder aus den Zeilen einer Datei herausschneiden
diff(1) Dateien zeilenweise vergleichen in *ed*-Skript
diff3(1) Drei Dateien zeilenweise vergleichen
dircmp(1) Dateiverzeichnisse vergleichen
egrep(1) Erweiterte Muster suchen
fgrep(1) Einfache Muster suchen
find(1) Dateiverzeichnisse durchsuchen
grep(1) Muster suchen
join(1) Dateien nach Vergleichsfeldern verbinden
nl(1) Dateizeilen numerieren
newform(1) Format einer Textdatei ändern
paste(1) Zeilen zusammenfügen
sdiff(1) Zwei Dateien gleichzeitig ausgeben und vergleichen
sort(1) Sortieren und Mischen von Dateien
spell(1) Rechtschreibfehler suchen
split(1) Datei aufteilen auf mehrere Dateien
sum(1) Prüfsumme einer Datei berechnen
tr(1) Zeichen durch andere ersetzen
uniq(1) Mehrfache Zeichen suchen
wc(1) Zeilen, Wörter und Zeichen zählen

Kommandos

verwalten

ar(1)	Bibliotheken erstellen und verwalten
cpio(1)	Dateien und Dateiverzeichnisse ein- und auslagern
make(1D)	Gruppen von Dateien verwalten
what(1D)	Versionen identifizieren

sichern und archivieren

ar(1)	Bibliotheken erstellen und verwalten
cpio(1)	Dateien und Dateiverzeichnisse ein- und auslagern
dd(1)	Dateien konvertieren und kopieren
tar(1)	Archivieren von Dateien

komprimiert speichern bzw. in Ursprungszustand versetzen

pack(1)	Datei in komprimierter Form abspeichern
pcat(1)	Komprimiert gespeicherte Datei in Urform ausgeben
unpack	Komprimiert gespeicherte Datei in Urform bringen

Dateieigenschaften abfragen und ändern

chgrp(1)	Gruppennummer für eine Datei ändern
chmod(1)	Zugriffsrechte für eine Datei ändern
chown(1)	Eigentümer einer Datei ändern
file(1)	Art einer Datei bestimmen
ls(1)	Informationen zu Dateiverz. und Dateien ausgeben
touch(	Aktuelles Datum setzen für letzte Datei-Änderung
umask(1)	Standardeinstellung der Schutzbits ändern

Dateisystem verändern

chroot(1)	Root-Dateiverzeichnis für ein Kommando ändern
cp(1)	Datei kopieren
cpio(1)	Dateien und Dateiverzeichnisse ein- und auslagern
destroy(1)	Datei physikalisch löschen
find(1)	Dateiverzeichnisse durchsuchen
ln(1)	Verweis auf eine Datei eintragen
mkdir(1)	Dateiverzeichnis einrichten
mv(1)	Dateien umbenennen oder übertragen
rm(1)	Dateien löschen
rmdir(1)	Dateiverzeichnisse löschen
split(1)	Datei aufteilen auf mehrere Dateien

Editoren

bfs(1)	Editor nur zum Durchsuchen von Dateien (Zeilen)
ced(1)	Bildschirmorientierter Editor
ed(1)	Zeilenorientierter Editor
edit(1)	Eingeschränkte Variante von <i>ex</i>
ex(1)	Kommandoorientierter Editor
sed(1)	Editor im Prozedurbetrieb
red(1)	Eingeschränkte Variante von <i>ed</i>
vi(1)	Bildschirmorientierter Editor

Hilfskommandos für Shell-Prozeduren

basename(1)	Dateinamen vom Pfad trennen
dirname(1)	Zugriffspfad vom Dateinamen trennen
expr(1)	Ausdrücke auswerten
false(1)	Leeres Kommando mit Endestatus ungleich 0
getopt(1)	Schalter suchen in Argumenten eines Prozeduraufrufs
sleep(1)	Prozesse zeitweise stilllegen
test(1)	Bedingungen prüfen
true(1)	Leeres Kommando mit Endestatus 0
xargs(1)	Argumentenliste aufbauen und Kommando ausführen

Zeichen einlesen und ausgeben

echo(1)	Zeichenfolgen ausgeben
line(1)	Zeilen einlesen und ausgeben
tee(1)	Ausgeben auf Standard-Ausgabe <i>und</i> in eine Datei

Zeichen einlesen, umwandeln und ausgeben

banner(1)	Zeichen in großer Titelschrift ausgeben
col(1)	Filter für umgekehrte und halbe Zeilenvorschübe
od(1)	Inhalt einer Datei oktal ausgeben

Kommandos

SCCS-Kommandos

admin(1D)	SCCS-Dateien erstellen und verwalten
bdiff(1D)	Große Dateien vergleichen
cdc(1D)	Kommentar bzw. MR-Nummern eines Deltas ändern
comb(1D)	Deltas zusammenfassen
delta(1D)	Delta erstellen
get(1D)	Version aus einer SCCS-Datei holen
prs(1D)	Informationen über eine SCCS-Datei ausgeben
rmDEL(1D)	Delta löschen
sact(1D)	Ausgeben, welche Versionen anstehen
scsdiff(1D)	Zwei Versionen vergleichen
unget(1D)	<i>get</i> -Kommando rückgängig machen
val(1D)	SCCS-Dateien auf Konsistenz prüfen
vc(1D)	Textdarstellung kontrollieren
what(1D)	Versionen identifizieren

Benutzereigenschaften abfragen und ändern

id(1)	Benutzer- und Gruppenkennung ausgeben
login(1)	Benutzerkennung wechseln
logname(1)	Login-Namen abfragen
mesg(1)	Ausgabe von Meldungen verhindern oder erlauben
newgrp(1)	Benutzergruppe wechseln
passwd(1)	Kennwort für Benutzerkennung eintragen oder ändern
su(1)	Benutzerkennung vorübergehend wechseln

Kommunikation mit anderen Benutzern

mail(1)	Post senden und empfangen
mailx(1)	Nachrichten interaktiv verarbeiten
mesg(1)	Ausgabe von Meldungen verhindern oder erlauben
write(1)	Dialog mit anderen Benutzern
wall(1)	An alle Benutzer schreiben

Kalenderfunktionen und Termine

at(1)	Kommando zu einer bestimmten Zeit ausführen
cal(1)	Kalender ausgeben
calendar(1)	Erinnerungsdienst
crontab(1)	Aufträge regelmäßig zu bestimmter Zeit ausführen
date(1)	Datum und Uhrzeit ausgeben

Rechenfunktionen

bc(1)	Arithmetische Sprache
dc(1)	Tischrechner
expr(1)	Ausdrücke auswerten
factor(1)	Zahl in Primzahlen zerlegen
units(1)	Einheiten umrechnen

Informationen über Prozesse

ps(1)	Prozeßdaten abfragen
time(1)	Laufzeit eines Kommandos messen

Prozesse steuern

batch(1)	Aufträge in Warteschlange; nacheinander abarbeiten
kill(1)	Prozesse beenden, Signale senden
nice(1)	Priorität von Kommandos ändern
nohup(1)	Signale ignorieren
sleep(1)	Prozesse zeitweise stilllegen
wait(1)	Auf Prozeß-Ende warten

Interprozeßkommunikation

ipcrm(1)	Kennung entfernen, Nachrichtenwarteschlange oder Semaphor
ipcs(1)	Status ermitteln, Interprozeßkommunikation-Einrichtungen
mknod(1)	FIFO-Datei erstellen (Pipe mit Name)

Kommandos

Eigenschaften der Datensichtstation abfragen und ändern

stty(1)	Eigenschaften der Datensichtstation abfragen und ändern
tic(1D)	Informationen über Datensichtstation übersetzen
tput(1)	Terminfo-Datenbank abfragen und der Shell Eigenschaften der Datensichtstation zur Verfügung stellen

Speicherplatzbelegung überprüfen

df(1)	Dateisystem auf freien Platz prüfen
du(1)	Belegten Speicherplatz ausgeben

Informationen über Systemdaten

ps(1)	Prozeßdaten abfragen
uname(1)	Namen des aktuellen Systems ausgeben
what(1D)	Versionsnummern ausgeben
who(1)	Aktive Benutzerkennungen anzeigen

Information über Prozessortyp

machid(1)	Wahrheitswerte über Art des Prozessors ausgeben

C-Programme bearbeiten und verwalten

ar(1)	Bibliotheken erstellen und verwalten
cb(1D)	C-Programme formatieren
cc(1D)	C-Programme übersetzen
cflow(1D)	Flußdiagramme für C-Programme generieren
cxref(1D)	Querverweisliste für C-Programme erstellen
ld(1D)	Objektdateien zusammenbinden
lint(1D)	C-Programme überprüfen
lorder(1D)	Objektmodule paarweise ordnen
make(1D)	Gruppen von Programmen verwalten
nm(1D)	Symboltabelle ausgeben
prof(1D)	Zeittabelle für ein C-Programm erstellen
ranlib(1D)	Bibliothek mit einer Symboltabelle versehen
size(1D)	Größe einer Objektdatei ausgeben
strip(1D)	Symboltabelle entfernen
tic(1D)	Informationen über Datensichtstation übersetzen
tsort(1D)	Paarweise geordnete Objektmodule vollständig ordnen

C-Programme generieren

lex(1D) Programme zur lexikalischen Analyse generieren
regcmp(1) Reguläre Ausdrücke in die Sprache C übersetzen
yacc(1D) Parser generieren

Makroprozessor

m4(1D) Makroprozessor für C und andere Sprachen

On-Line-Dokumentation

man(1) On-Line-Dokumentation auf Englisch ausgeben

Systempuffer leeren

sync(1) Systempuffer zurückschreiben

—

—

—

—

5 Editoren

Was können Sie mit einem Editor machen?

Wenn Sie selber Texte in den Rechner eingeben wollen, über Tastatur und Bildschirm, so können Sie das mit einem Editor machen. Solche Texte sind z.B.

- Briefe, eine Telefonliste, eine Dokumentation
- Quelltext eines Programms
- Shell-Prozedur u. ä.

Die Eingabe kleiner, einfacher Texte können Sie z.B. mit den SINIX-Kommandos `echo` oder `cat` in eine Datei umlenken, aber diese relativ starre Texterzeugung ist hier nicht gemeint.

Ein Editor ist ein Programm, mit dem Sie nicht nur Daten eingeben, sondern auch verändern können. Ein Editor bietet Ihnen viele Möglichkeiten, an Ihren Texten

- Korrekturen,
- Änderungen,
- Erweiterungen
- Textverschiebungen usw.

mit der Unterstützung von Kommandos vorzunehmen, bis der Text den Anforderungen entspricht.

Mit den meisten Editoren arbeiten Sie im Dialog und können dabei jeden Schritt sofort am Bildschirm sehen, kontrollieren und ggf. korrigieren.

Auch Texte in einer bereits bestehenden Datei können Sie mit einem Editor nach Ihren Wünschen verändern oder korrigieren. Voraussetzung dazu ist nur, daß Sie das Zugriffsrecht auf diese Datei haben.

Einen Editor rufen Sie mit einem Kommando auf. Dann bearbeiten Sie Ihre Datei mit Tastatur und Bildschirm. Zum Abschluß Ihrer Arbeiten speichern Sie die bearbeiteten Daten unter einem Namen, dem Namen einer Datei.

Mit einem Editor können Sie nur Daten bearbeiten, die aus Zeilen bestehen. Jede Zeile muß mit dem Neue-Zeile-Zeichen (oktal 012) abgeschlossen sein. Eine so aufgebaute Datei nennt man *Dokument*.

Ein Dokument ist z.B. der Quelltext eines Programms. Dieser kann mit einem Editor bearbeitet werden. Ist aber das Programm schon übersetzt, so ist es kein Dokument mehr und für eine Editorbearbeitung ungeeignet.

In diesem Kapitel ist das Wort *Datei* im Sinne von *Dokument* verwendet. Mit dem Wort *Text* sind beliebige Zeichenketten gemeint, die aus Buchstaben, Ziffern und Sondertasten bestehen, die auf der Tastatur vorhanden sind.

Es gibt mehrere Editoren

Im SINIX-System gibt es mehrere Editoren. Sie sind für verschiedene Zwecke und Editierumgebungen vorgesehen.

Folgend sind einige Editoren aufgeführt.

(Ihre Kommando-Beschreibungen finden Sie im Buch SINIX-Kommandos.)

ed	Zeilenorientierter Editor (Einfach und schnell)
sed	Stapelorientierter Editor (Prozedurbetrieb)
ex	Zeilenorientierter Editor (verbesserter ed)
edit	Zeilenorientierter Editor (einfache Variante von ex)
vi	Bildschirmorientierter Editor (erweiterter ex, visual)
ced	Bildschirmorientierter Editor (Kommandos per Menü)

Teilweise überschneiden sich die Anwendungsmöglichkeiten der Editoren. Einige Editoren sind Weiterentwicklungen von früher entwickelten Editoren, wobei ggf. neue Editor-Kommandos oder alte Editor-Kommandos gleichwertig benutzt werden können.

Neben den oben aufgeführten Editoren gibt es noch viele weitere Textbearbeitungs- und -verarbeitungs-Programme.

Was heißt bildschirmorientiert?

Der Gegensatz zu 'bildschirmorientiert' ist 'zeilenorientiert'. Bei letzteren Editoren kann jeweils innerhalb einzelner Zeilen geändert oder korrigiert werden, ggf. auch mehrere Zeilen in einer Art, mit einem Kommando. Solche Editoren sind für kleinere, schnell auszuführende Aktionen gut geeignet.

Moderne Editoren sind alle bildschirmorientiert. Dabei zeigt der Editor die Datei oder einen Dateiausschnitt automatisch über den ganzen Bildschirm.

Innerhalb des jeweils angezeigten Bereichs können Sie die Datei an beliebiger Stelle aufbauen oder ändern. Um diese Stelle zu erreichen, müssen Sie lediglich die Schreibmarke dorthin positionieren. Es gibt auch Kommandos, mit denen Sie über die ganze Datei ändern können.

Ist die Datei größer als auf einem Bildschirm darstellbar, so können Sie dem Editor angeben, ab welcher Zeile er Ihnen den Dateiausschnitt zeigen soll.

Bei einigen Editoren können Sie auch die Größe des anzuzeigenden Dateiausschnitts als *Fenster* festlegen und ggf. auch mehrere Dateiausschnitte in mehreren Fenstern auf einem Bildschirm anzeigen lassen.

Auch in einem Fenster können Sie die Schreibmarke an jede gewünschte Stelle bewegen und dann dort Ihre Änderungen sichtkontrolliert durchführen.

Wie arbeitet ein Editor?

Aufruf

Den gewünschten Editor rufen Sie mit seiner Kurzbezeichnung auf, also mit seinem Kommandonamen. Zumeist gibt man dabei auch gleich einen Dateinamen an. Das kann auch der Name einer neuen Datei sein.

Mit dem Aufruf des Editors richtet sich dieser selbst einen Arbeits-Speicherbereich als temporär angelegte Datei ein. Während der Editorsitzung arbeiten Sie mit diesem Speicherbereich, wie mit einem Notizzettel.

Wenn die angegebene Datei schon vorhanden war, so erzeugt sich der Editor selbst eine Kopie dieser Datei. Das ist dann wiederum Ihr Arbeits-Speicherbereich.

Warum macht er das? Damit haben Sie jederzeit die Möglichkeit, den verfaßten Text zu verwerfen oder ihn in einer Datei zu sichern. Sie können auch Zwischenergebnisse in Dateien mit anderen Namen zwischenspeichern. In jedem Falle können Sie Ihre Arbeiten so einrichten, daß Sie immer noch auf die ursprüngliche Datei zurückgreifen können.

Textbearbeitung

Im Editor-Arbeits-Speicherbereich können Sie Ihren Text

- Schreiben
- Hinzufügen
- Löschen
- Korrigieren, Ändern u. ä.

Editor-Kommandos

Einen Editor steuern Sie mit editoreigenen Kommandos, z.B. für

- Fenster positionieren
- Schreibmarke positionieren
- Text schreiben, ersetzen, einfügen, speichern, löschen, zurückholen ...
- Text suchen und anzeigen, ersetzen
- Textsegmente bearbeiten

Die Kommandos sind ein Zeichen oder wenige mehrere Zeichen. Im Editor *vi*, können Sie z. B. mit *w* die Schreibmarke zum nächsten Wortanfang bewegen. Auskunft über die festgelegten Kommando-Zeichen erhalten Sie jeweils durch die Editorbeschreibung oder über abrufbare Hilfetexte am Bildschirm, z.B. bei dem Editor *ced*.

Wie unterscheidet nun ein Editor die Eingaben für Text oder Kommandos? Üblicherweise arbeiten die Editoren in verschiedenen Modi. Die Umschaltung nimmt man mit festgelegten Tasten vor. So schaltet ein Editor z.B. vom Eingabemodus in den Kommandomodus um. Die eingegebenen Zeichen werden dann als Editor-Kommandos erkannt. Normalerweise schaltet der Editor, nachdem das Kommando abgeschlossen ist, selbst in den Eingabemodus zurück.

Editor-Speicherpuffer

Neben dem Arbeits-Speicherbereich, der meistens einfach als "Editor-Puffer" bezeichnet wird, stellt Ihnen ein Editor meistens viele Möglichkeiten zur Verfügung, um z.B. oft verwendete Wörter, schwierige Zeichenfolgen oder auch Kommandos zwischenzuspeichern. Diese *Puffer* können Sie bei Bedarf mit einem Namen oder einer bestimmten Taste abrufen. Ob diese Einträge mit dem Beenden des Editors gelöscht werden oder nicht, ist in seiner Beschreibung nachzulesen.

Während Ihrer Editierarbeiten ist es angebracht, gelegentlich den Editor-Puffer unter einem Dateinamen zu sichern und außerdem die während Ihrer Editorsitzung angelegten kleinen Puffer zu löschen, wenn Sie diese nicht mehr brauchen.

Editor beenden

Wenn Sie Ihre Arbeiten abschließen wollen, beenden Sie die Editorsitzung. Manche Editoren speichern dabei den Text des Editor-Puffers automatisch in die dem Editor, von vorherigen Eingaben schon bekannte Datei. Anderen Editoren können oder müssen Sie zum Speichern einen Dateinamen angeben. Nach dem Abschluß einer Editorsitzung haben Sie auf den Editor-Puffer keinen Zugriff mehr!

5.1 Der Editor *ced*. Eine Beispielsitzung

Welches Ziel hat diese Beispielsitzung?

Sie sollen ein Gefühl für die Arbeit mit dem *ced* bekommen. Anhand eines durchgehenden Beispiels werden die verschiedenen Modi erklärt, in denen der *ced* arbeitet.

Es sind bei weitem nicht alle Möglichkeiten des *ced* aufgeführt. Eine vollständige Kommandobeschreibung des *ced* finden Sie im Buch SINIX-Kommandos.

Der *ced* kann nur Dokumente bearbeiten, die aus Zeilen aufgebaut sind. (Jede Zeile muß mit dem Neue-Zeile-Zeichen abgeschlossen sein.)

Zum Arbeiten mit dem *ced* brauchen Sie keine komplizierte Kommando-syntax zu lernen, sondern wählen einfach aus einem Menü das nächste Kommando aus.

Was können Sie mit dem *ced* machen?

- Neuen Text schreiben, Dateien erzeugen
- Text einfügen, überschreiben, löschen
- Zeilen einfügen, löschen, merken, teilen, verbinden
- Mehrere Zeilen als Block kopieren, löschen, einfügen
- Rechtecke frei markieren, kopieren, löschen, einfügen
- Nach Zeichenketten suchen
- In andere Dateien umschalten
- SINIX-Kommandos aufrufen: Eingabe aus der Datei, Ausgabe in die Datei
- Für *ced* Tasten mit Text und Kommandos belegen
- Temporär in die Shell wechseln.

Ein kleines Beispiel

Damit Sie die prinzipielle Arbeitsweise des *ced* kennen lernen, soll eine kleine Telefonliste mit insgesamt nur 4 Zeilen erstellt und in eine neue Datei mit dem Namen *test* übernommen werden. Sie soll so aussehen:

Telefonliste

Adam 4999

Eva 3210

Sie rufen den *ced* auf:

\$ ced test

Der *ced* richtet Ihnen einen Arbeits-Speicherbereich ein.

Der Bildschirm zeigt Ihnen einen Editierrahmen mit 19 leeren Zeilen. Sie können gleich anfangen, den Text einzugeben: Telefonliste

Bei einem Tippfehler korrigieren Sie mit der Taste .

Dann schalten Sie in eine neue Zeile mit der Taste . Die Schreibmarke geht mit. Schalten Sie dann gleich in die nächste Zeile, damit eine Leerzeile entsteht. Sie können gleich weiter schreiben.

Adam 4999

Eva 3210

Der Text ist nun fertig. Er steht im Editor-Puffer. Die Datei *test* ist noch leer.

Bisher haben Sie im *ced* im Modus "neuen Text eingeben" gearbeitet. Der *ced* schaltet automatisch in diesen Modus, wenn er aufgerufen wird.

Jetzt müssen Sie Ihren Text in die Datei *test* schreiben, damit er erhalten bleibt. Dazu müssen Sie den *ced* in einen anderen Modus schalten. Im *ced* verwendet man dazu die Taste  und nachfolgend einen Buchstaben, in unserem Beispiel z.B. a für "abspeichern". Drücken Sie die Tasten:





für die Eingabe a



Wenn Sie unten am Bildschirm keinen Dateinamen eintragen, so speichert der *ced* den Inhalt des Editor-Puffers in die ihm bereits bekannte Datei *test*. So können Sie jederzeit, auch während Ihrer Arbeit, den erreichten Stand fixieren.

Hätten Sie vor einen anderen Dateinamen eingetragen, würde diese Datei verwendet werden.

Wenn Sie Ihre Arbeiten abschließen wollen und die dem *ced* bereits bekannte Datei zur Speicherung verwenden wollen, so schalten Sie den *ced* am einfachsten gleich in den Modus "*ced* verlassen":

für Eingabe v

Vor dem Verlassen des *ced* fragt er immer (unten am Bildschirm), ob Sie Ihre Arbeit speichern wollen oder nicht. Mit der Eingabe von j für ja oder n für nein, wird der Inhalt des Editor-Puffers in die dem *ced* bekannte Datei gesichert oder nicht. Wenn ja, wird die Datei *test* mit dem gleichen Inhalt, den der Editor-Puffer hat, eingerichtet. Wenn nein, wird die Datei *test* nicht eingerichtet.

Beachten Sie bitte, bevor Sie die Antwort n für nein eingeben, daß Sie nach dem Verlassen des *ced* in keinem Falle mehr Zugriff auf den vorherigen Inhalt des Editor-Puffers haben, d.h. er geht dann verloren!

Der *ced* wird dann beendet. Sie sehen wieder die Eingabeaufforderung \$.

Wenn Sie Ihre neue Datei am Bildschirm kontrollieren wollen, so geben Sie dieses SINIX-Kommando ein:

```
$ cat test
```

Am Bildschirm erhalten Sie dann die Ausgabe:

Telefonliste

Adam 4999

Eva 3210

Wenn Sie Ihre kleine Telefonliste am Drucker ausgeben wollen, so geben Sie das SINIX-Kommando:

```
$ lpr test
```

Folgend sind die weiteren Modi des *ced* und weitere Tastenfunktionen erklärt.

5.1.1 Welche Tasten können Sie im ced benutzen?

Zur Eingabe von Text stehen Ihnen alle alphanumerischen Tasten und alle Sonderzeichentasten zur Verfügung. Die im folgenden Bild abgebildeten Tasten haben für den *ced* besondere Bedeutung:

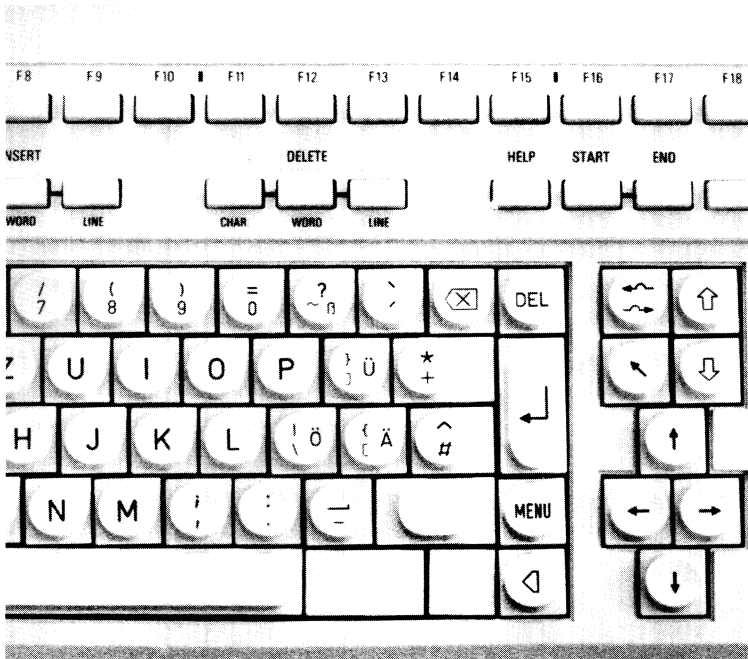


Bild 5-1 Tastatur-Teilansicht

Die Funktionstasten F9 bis F17

Die Funktionstasten **F9** bis **F17** dienen zum schnellen Aufruf einiger *ced*-Funktionen. Ihre Bedeutung steht auf dem mitgelieferten Tastenstreifen. Legen Sie ihn bitte über die obere Tastenreihe der Tastatur.

Löschen und Einfügen von Zeichen

Zum Korrigieren von einzelnen Zeichen können Sie diese Tasten verwenden

,  und .

Mit der Taste  setzen Sie die Schreibmarke um ein Zeichen nach links und löschen das Zeichen. Mit dem Betätigen der Taste  löschen Sie das Zeichen, auf dem die Schreibmarke steht. Drücken Sie die Taste , so fügen Sie an der Schreibmarkenposition ein Leerzeichen ein. Wenn Sie allerdings mehrere Zeichen einfügen wollen, benutzen Sie besser den *ced*-Modus 'Text einfügen'.

Löschen und Einfügen von Zeilen

Drücken Sie die Taste , so löschen Sie die Zeile, in der sich die Schreibmarke befindet. Um eine Zeile in Ihren Text einzufügen, drücken Sie die Taste . Die neue Zeile wird oberhalb der Zeile, in der sich Ihre Schreibmarke befindet, eingefügt.

Die Tasten  und  haben keine Wirkung.

Schreibmarke bewegen

Mit den Tasten , , , , ,  bewegen Sie die Schreibmarke auf dem Bildschirm hin und her. Die Taste  verstärkt deren Wirkung. Ihre genaue Funktion ist im Abschnitt 4.1.5 beschrieben.

Wie können Sie das Fenster über dem Dokument verschieben? oder Wie können Sie im Dokument blättern?

Mit den Tasten  und  verschieben Sie das Fenster jeweils um 10 Zeilen nach unten, bzw. oben. Mit den beiden Funktionstasten  und  verschieben Sie das Fenster jeweils um einen ganzen Bildschirm nach unten, bzw. nach oben. Wollen Sie das Fenster auf den Anfang des Dokuments positionieren, benutzen Sie dazu die Taste .

5.1.2 Der ced arbeitet in verschiedenen Modi

Je nachdem, in welchem Modus Sie sich gerade befinden, haben Ihre Eingaben eine andere Bedeutung für den *ced*. Wollen Sie in einen anderen Modus umschalten oder ein anderes *ced*-Kommando ausführen, müssen Sie zuerst in das *ced*-Hauptmenü umschalten. Drücken Sie dazu die Taste **MENU**. In der untersten Zeile des Bildschirms bietet Ihnen nun der *ced* die möglichen Eingaben an.

Die Modi des *ced* und die dazugehörigen Kommandobuchstaben, die Sie benutzen müssen, um in den entsprechenden Modus umzuschalten, sind:

• Zeilenbereich bearbeiten	Taste B
• SINIX-Kommando ausführen	Taste K
• <i>ced</i> verlassen	Taste V
• Text einfügen	Taste E
• suchen	Taste S
• eine Zeile bearbeiten	Taste Z
• ein Rechteck bearbeiten	Taste R
• neuen Text eingeben	Taste N
• abspeichern	Taste A
• Dokument wechseln	Taste D
• Fenster positionieren	Taste F
• Textmarkierung suchen	Taste T
• Protokoll, Tasten belegen	Taste P

Außerdem bietet Ihnen der *ced* noch zwei Kommandos an, die nicht in einen anderen Modus schalten, sondern direkt ausgeführt werden:

• Tastenbelegung anzeigen	Taste X
• temporär in die Shell wechseln	Taste I

Die HELP-Taste

Wenn Sie während einer *ced*-Sitzung nicht mehr weiter wissen, drücken Sie im *ced*-Hauptmenü, d.h. nach der Taste **[MENU]** die Taste **[HELP]**. Der *ced* schaltet dann in ein Dokument um, das eine Kurzbeschreibung des *ced* enthält. Mit den Tasten **[↓]** und **[↑]** können Sie dort vor- und zurück blättern.

Wichtig: Um in das ursprüngliche Dokument zurückzugelangen, drücken Sie einfach die Taste **[MENU]** oder die Funktionstaste **[F10]**.

Ein Tastendruck - ein Kommando

Der *ced* verarbeitet Ihre Eingaben sofort. Deshalb brauchen Sie Ihre Kommandos nicht mit der Taste **[↵]** abzuschließen.

Ausnahmen: In vier Modi, in denen Zusatzinformationen notwendig sind, müssen Sie diese mit der Taste **[↵]** abschließen. Das sind:

- im Modus 'abspeichern' der Name einer Datei
- im Modus 'Dokument wechseln' der Name des anderen Dokuments
- im Modus 'suchen' die Zeichenkette, die gesucht werden soll
- im Modus 'Fenster positionieren' die Schrittweite oder die Zeilennummer

5.1.3 Beginnen Sie !

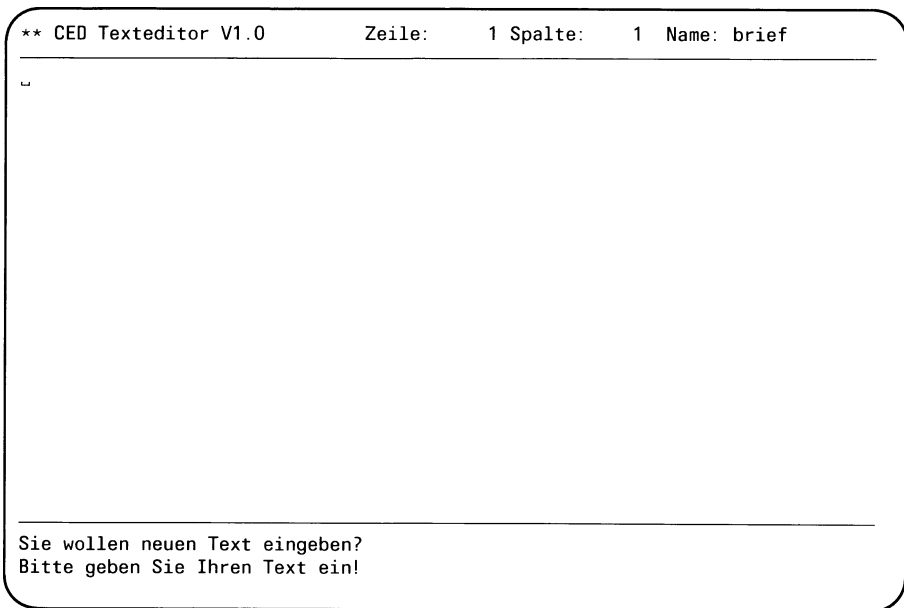
Tippen Sie ein:

ced, dann ein Leerzeichen und den Namen der zu editierenden Datei.

Für unsere Beispielsitzung wählen wir den Dateinamen *brief*. Diese Datei soll in unserem aktuellen Dateiverzeichnis noch nicht existieren. Ihre Eingabezeile sollte jetzt so aussehen.

\$ ced brief

Drücken Sie die nun Taste . Wie sieht Ihr Bildschirm jetzt aus?



Hinweis

Die Position der Schreibmarke ist auf dem Bildschirm durch das Zeichen  dargestellt.

Der *ced* hat Ihnen einen Editierraum auf den Bildschirm geschrieben. Dieser Rahmen besteht aus drei Teilen:

- einer Kopfzeile
- dem Sichtbereich
- dem Bedienbereich.

Die Kopfzeile informiert Sie über

- den Namen und die Versionsnummer des Editors,
- die Position der Schreibmarke im Dokument,
- den Namen der Datei, die Sie gerade bearbeiten.

Die Angabe der Zeile und Spalte - hier 1 und 1, d.h. die Schreibmarke steht am Anfang der Datei - bezieht sich immer auf die absolute Position der Schreibmarke im Dokument und nicht auf ihre Position am Bildschirm.

Der Sichtbereich, der von der Kopfzeile und dem Bedienbereich durch je eine durchgezogene Linie getrennt ist, ist leer, wenn Sie den Namen einer nicht existierenden Datei angegeben haben.

Der zweizeilige Bedienbereich am unteren Bildschirmrand zeigt Ihnen:

- in der ersten Zeile den Namen des eben gültigen Modus
- in der zweiten Zeile alle in diesem Modus möglichen Kommandooptionen

Zu Beginn einer *ced*-Sitzung befinden Sie sich immer im Modus: 'neuen Text eingeben'.

5.1.4 Wie geben Sie Text ein?

Tippen Sie ihn wie auf einer Schreibmaschine ein. Zum Korrigieren benutzen Sie diese Tasten:

 schiebt die Schreibmarke um eine Position nach links und löscht das dortige Zeichen,

 löscht das Zeichen an der Schreibmarkenposition,

 fügt an der Schreibmarkenposition ein Leerzeichen ein.

Wenn Sie eine neue Zeile beginnen wollen, drücken Sie die Taste . Die Schreibmarke befindet sich jetzt in der ersten Spalte der nächsten Zeile.

Hinweis

Auf diese Weise können Sie sich neue Zeilen am Ende Ihres Dokuments schaffen.

Versuchen Sie, über den rechten Rand hinaus, d.h. über die 80. Spalte hinauszuschreiben. Der *ced* verschiebt den Sichtbereich um 41 Spalten nach rechts. Nun können Sie wieder weiterschreiben, bis Sie wieder am rechten Rand anstoßen. Dieses Mal verschiebt der *ced* das Fenster nur um 6 Spalten, denn die maximale Länge einer Zeile ist 127 Zeichen. Wenn Sie versuchen, darüber hinauszuschreiben, gibt der *ced* eine Fehlermeldung aus.

5.1.5 Wie blättern Sie im Dokument?

Mit diesem 'an-den-Rand-Stoßen' haben Sie eine interessante Eigenschaft des *ced* entdeckt. Sobald Sie auf dem Bildschirm an einen Rand stoßen, entweder mit der Schreibmarke oder beim Eintippen von Text, blättert der *ced* automatisch um einen halben Bildschirm in die entsprechende Richtung - falls möglich.

Eine andere Möglichkeit, im Dokument zu blättern, haben Sie durch die Tasten  und . Drücken Sie eine dieser Tasten, so bewegen Sie den Sichtbereich jeweils eine halbe Seite, d.h. 10 Zeilen nach unten bzw. oben.

Wollen Sie noch schneller blättern, benutzen Sie die beiden Funktionstasten  oder .  verschiebt das Fenster um eine Seite, d.h. um 20 Zeilen weiter nach unten,  um 20 Zeilen nach oben.

'Schnelle Schreibmarkenbewegungen'

Um die Schreibmarke in Sprüngen nach rechts oder links in einer Zeile zu bewegen, können Sie die Tasten  und  benutzen. Der *ced* benutzt intern Tabulatormarken, die alle 8 Spalten gesetzt sind. Betätigen Sie eine der Tasten  oder , so springt die Schreibmarke zuerst auf die nächste Tabulatormarke rechts oder links; bei jedem weiteren Betätigen dieser Tasten auf die nächste Tabulatormarke.

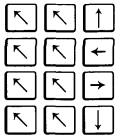
Sie können die Tabulatormarken nicht verändern.

Wie springen Sie an den Rand des Sichtbereichs?

Haben Sie schon einmal die Taste  benutzt? Sie haben sich sicherlich gewundert, daß sie nicht so wirkt, wie Sie es erwartet haben. Drücken Sie anschließend noch eine der vier Schreibmarkentasten    . Jetzt können Sie ihre Funktion ahnen. Die Taste  verstärkt die Wirkung der nachfolgend gedrückten Taste. Die Schreibmarke wird auf die oberste Zeile, den linken oder den rechten Rand oder die unterste Zeile des Sichtbereichs positioniert. Damit haben Sie nun die Möglichkeit, mit zwei Tastenanschlägen alle Ränder des Sichtbereiches zu erreichen.

Wie springen Sie ans Ende oder den Anfang Ihrer Datei?

Auch das ermöglicht die Taste . Drücken Sie sie zweimal und dann eine der Tasten    oder . Das Fenster wird auf den entsprechenden Rand des Dokuments positioniert. Die Schreibmarke steht bei der Tastenfolge:



auf der ersten Zeile,
auf der ersten Spalte,
auf der letzten Spalte,
auf der letzten Zeile des Dokuments.

An den rechten oder linken Rand des Dokuments können Sie auch durch die Tastenfolge   bzw.   gelangen. Mit der Taste  kommen Sie auch an den Anfang des Dokuments.

Der Beispieltext

Als Grundlage für die nächsten Schritte in unserer Beispielsitzung soll der folgende Text dienen. Die zweispaltige Tabelle können Sie mit Hilfe der Tabulatortasten leicht eingeben:

```

** CED Texteditor V1.0           Zeile:   11 Spalte:  33  Name: brief
-----
Betr.: SINIX-Dokumentation.
Die folgende Tabelle enthaelt Begriffe, wie sie aus der
englischen Originaldokumentation fuer die SINIX-Dokumen-
tation uebersetzt wurden.

process      Prozess
shell        Shell
special file  Geraetedatei
cursor       Schreibmarke
directory    Dateiverzeichnis
file         Datei

-----
Sie wollen neuen Text eingeben?
Bitte geben Sie Ihren Text ein!

```

5.1.6 Wie verschieben Sie einen Zeilenbereich?

Nehmen Sie an, Sie wollen Ihre Tabelle alphabetisch sortieren. In unserem Beispiel ist das dadurch möglich, daß Sie die drei unteren Zeilen der Tabelle an den Tabellenanfang nach oben verschieben - und Ihre Tabelle ist sortiert. Aber wie? Drücken Sie folgende Tasten:

MENU

B

Schreibmarke auf die Zeile 9 positionieren

M

Schreibmarke auf die Zeile 11 positionieren

M

Mit Menü **b** schalten Sie in den Modus 'Zeilenbereich bearbeiten'. In diesem Modus können Sie zusammenhängende Bereiche von ganzen Zeilen bearbeiten. Einen Bereich legen Sie durch Markieren der ersten und letzten Zeile fest. Dabei spielt es keine Rolle, in welcher Spalte Sie die Zeile markieren.

Die untere Zeile des Bedienbereichs zeigt Ihnen dabei die möglichen Kommandos an.

Mit der Taste **M** markieren Sie die erste Zeile des Bereichs, nachdem Sie die Schreibmarke auf die Zeile 9 positioniert haben:

cursor	Schreibmarke

Drücken Sie die Taste **M**. Der *ced* meldet:

***** MARKE GESETZT

Genauso markieren Sie die Zeile 11, als letzte Zeile des Bereichs:

file	Datei

Drücken Sie dann die Taste **L**. Der markierte Bereich wird gelöscht. Aber keine Angst: Der *ced* hat die gelöschten Zeilen in einem Puffer zwischengespeichert.

Wenn Sie die Schreibmarke auf die oberste Zeile der Tabelle positioniert haben, drücken Sie die Taste **Z**, für zurückholen:
Der gelöschte Bereich wird dadurch *oberhalb* der Zeile, in der die Schreibmarke steht, eingefügt. Unser Ziel ist erreicht.

Jetzt kennen Sie eine der Möglichkeiten, wie Sie einen markierten Zeilen-Block verschieben können.

Wie wurde das gemacht? Sie markieren im Modus 'Zeilenbereich bearbeiten' den entsprechenden Zeilenblock und löschen ihn. Dann positionieren Sie die Schreibmarke auf die neue Position und holen den gelöschten Bereich zurück.

Die Bearbeitungsfolge war:

MENU

B

Schreibmarke positionieren

M

Schreibmarke positionieren

M

L

Schreibmarke positionieren

Z

Aber Achtung: Der Inhalt des Puffers wird bei der nächsten Löschoption überschrieben!

Den Inhalt des Zwischenspeichers können Sie auch mit der Funktionstaste **F17** zurückholen.

Die gesetzten Markierungen werden nach der abgeschlossenen Aktion gelöscht.

Falls Sie der gezeigten Methode mißtrauen, wählen Sie eine andere, zweite Möglichkeit:

Nachdem Sie wie oben den Bereich markiert haben, positionieren Sie die Schreibmarke auf die erste Zeile der Tabelle und *kopieren* den markierten Bereich vor diese Zeile. Erst dann löschen Sie den Bereich am Ende der Tabelle.

Drücken Sie die Tasten:

.

. markieren

.

Schreibmarke positionieren **K L**

↑

In beiden Fällen muß Ihr Bildschirm so aussehen:

```
** CED Texteditor V1.0           Zeile:      6 Spalte:  39  Name: brief
-----
      Betr.: SINIX-Dokumentation.
      Die folgende Tabelle enthaelt Begriffe, wie sie aus der
      englischen Originaldokumentation fuer die SINIX-Dokumen-
      tation uebersetzt wurden.

      cursor      Schreibmarke
      directory   Dateiverzeichnis
      file        Datei
      process     Prozess
      shell       Shell
      special file Geraetedatei

-----
Sie wollen einen Zeilenbereich bearbeiten?
Bitte waehlen Sie eine der folgenden Funktionen: (e,k,l,m,z,<MENU>)!

```

Stellen Sie sich vor, Sie entschließen sich, die beiden Spalten der Tabelle zu vertauschen, d.h. die deutschen Begriffe vor die englischen zu stellen.

Können Sie mit dem ced Rechtecke verschieben?

Drücken Sie die Tasten:

MENU

R

Sie schalten damit in den Modus 'Rechtecke bearbeiten' um. Jetzt können Sie einen beliebigen rechteckigen Textblock verschieben. Die untere Zeile des Bedienbereichs zeigt Ihnen wieder alle möglichen Unterkommandos an. Mit der Taste **M** markieren Sie einen Eckpunkt eines Rechtecks. Um ein ganzes Rechteck festzulegen, müssen Sie *zwei* diagonal gegenüberliegende *Ecken* markieren. Bewegen Sie also die Schreibmarke zuerst auf den ersten Buchstaben des ersten Words in der englischsprachigen Spalte und drücken Sie die Taste **M**. *ced* quittiert Ihre Markierung durch:

***** MARKE GESETZT.

Genauso markieren Sie nun das letzte Leerzeichen vor dem Wort Geraetedei.

```

** CED Texteditor V1.0           Zeile:   11 Spalte:  33 Name: brief
-----
    Betr.: SINIX-Dokumentation.
           Die folgende Tabelle enthaelt Begriffe, wie sie aus der
           englischen Originaldokumentation fuer die SINIX-Dokumen-
           tation uebersetzt wurden.

           cursor           Schreibmarke
           directory        Dateiverzeichnis
           file             Datei
           process          Prozess
           shell            Shell
           special file     Geraetedei

-----
Sie wollen ein Rechte ***** MARKE GESETZT
Bitte waehlen Sie eine der folgenden Funktionen: (h,l,m,u,v,<MENU>)!

```

Im obigen Bild ist das nun markierte Rechteck gekennzeichnet.

Wie machen Sie weiter?

Der grundsätzliche Ablauf ist wie folgt:

- Schreibmarke auf die Position der oberen linken Ecke des "neuen Rechtecks" positionieren
- Je nach Art des Kopierens, ☐H oder ☐V oder ☐U drücken. (horizontal, vertikal, überschreiben)

Bewegen Sie jetzt die Schreibmarke auf eine Position hinter dem Wort Schreibmarke in der ersten Zeile der Tabelle. Die Position der Schreibmarke wird die linke obere Ecke des einzufügenden Rechtecks. Um die markierte Spalte hier einzufügen, drücken Sie die Taste ☐H (horizontal einfügen). Ihr Bildschirm müßte jetzt so aussehen:

```
** CED Texteditor V1.0           Zeile:      6 Spalte:  55 Name: brief
-----
Betr.: SINIX-Dokumentation.
      Die folgende Tabelle enthaelt Begriffe, wie sie aus der
      englischen Originaldokumentation fuer die SINIX-Dokumen-
      tation uebersetzt wurden.

      cursor      Schreibmarke      cursor
      directory   Dateiverzeichnis   dirctory
      file        Datei              file
      process     Prozess            process
      shell       Shell              shell
      special file Geraetedatei      special file

-----
Sie wollen ein Rechteck bearbeiten?
Bitte waehlen Sie eine der folgenden Funktionen: (h,l,m,u,v,<MENU>)!

```

Haben Sie Ihre deutsche Spalte zerstört? Dann haben Sie die Schreibmarke vor dem Kopieren nicht weit genug nach rechts bewegt. Wenn Sie horizontal einfügen, und das markierte Rechteck

- s Spalten und
- z Zeilen umfaßt

reserviert der *ced*

- ab der Schreibmarkenposition
- s Spalten nach rechts und
- z Zeilen - nach unten gezählt

Er schafft also genau den Platz, den das einzufügende Rechteck benötigt. Nun kopiert er das markierte Rechteck in den freien Bereich und positioniert die Schreibmarke auf die obere linke Ecke des eingefügten Rechtecks.

Jetzt haben Sie drei Spalten auf dem Bildschirm. Drücken Sie die Taste um die erste Spalte zu löschen.

Sie wundern sich? Die von Ihnen vorher gesetzten zwei Eck-Marken der ersten Spalte waren noch immer gültig. Die Taste bezog sich also auf diese Spalte. Ihr Bildschirm hat jetzt wieder nur noch zwei Spalten:

```

** CED Texteditor V1.0           Zeile:      6 Spalte:  19  Name:  brief
-----
    Betr.: SINIX-Dokumentation.
           Die folgende Tabelle enthaelt Begriffe, wie sie aus der
           englischen Originaldokumentation fuer die SINIX-Dokumen-
           tation uebersetzt wurden.

           Schreibmarke      cursor
           Dateiverzeichnis  directory
           Datei             file
           Prozess           process
           Shell             shell
           Geraetedatei      special file

-----
Sie wollen ein Rechteck bearbeiten?
Bitte waehlen Sie eine der folgenden Funktionen: (h,l,m,u,v,<MENU>)!

```

Warum horizontal einfügen? Kann man auch vertikal einfügen?

Im Modus 'Rechteck bearbeiten' stehen Ihnen zum Kopieren noch zwei weitere Tasten zur Verfügung: ☐ (vertikal einfügen) und ☐ (überschreiben):

Fügen Sie vertikal ein, so geschieht folgendes:

- Oberhalb der Zeile, in der die Schreibmarke steht, werden so viele Leerzeilen eingefügt, wie das markierte Rechteck umfaßt,
- Das markierte Rechteck wird in diese Leerzeilen kopiert und steht rechts neben der aktuellen Position der Schreibmarke,
- Die Schreibmarke springt an die obere linke Ecke des eingefügten Rechtecks.

Fügen Sie überschreibend ein (Taste ☐), so überschreibt das kopierte Rechteck den an der Einfügestelle stehenden Text.

Hinweis

Das kleinste markierbare Rechteck ist ein Zeichen.

Die Tabelle ist jetzt natürlich wieder in einem unsortierten Zustand (siehe letzter Bildschirm). Eine Sortier-Möglichkeit wäre, per Hand die einzelnen Zeilen entsprechend zu verschieben. Das könnten Sie im Modus 'eine Zeile bearbeiten' durchführen.

Beispiel

-10 bzw. 101 kennzeichnet die nächsten 10 Zeilen als Eingabebereich, ab der Zeile, in der sich die Schreibmarke befindet.

Geben Sie nur eine Dezimalzahl *n* an, so werden damit die nächsten *n* Abschnitte als Eingabebereich bezeichnet.

Machen Sie keine Angaben, nimmt *ced* standardmäßig den aktuellen Abschnitt, ab der Zeile, in der sich die Schreibmarke befindet, als Eingabebereich an.

Um die Tabelle alphabetisch zu sortieren müssen Sie folgende Tasten drücken:

Schreibmarke positionieren auf die erste Zeile der Tabelle

1 sort

Also:

Nachdem Sie die Schreibmarke positioniert haben, schalten Sie in den Modus 'Kommando ausführen' um. Geben Sie jetzt Eingabebereich und Kommando in der ersten Zeile des Bedienbereichs ein: 1 sort

Drücken Sie die Taste . Der *ced* meldet:

***** KOMMANDO WIRD AUSGEFUEHRT. BITTE WARTEN

und bearbeitet das Kommando. Ist die Kommandoverarbeitung beendet, löscht der *ced* den Eingabebereich, d.h. die Tabelle, und fügt an seine Stelle die Ausgabe des Kommandos, d.h. die alphabetisch geordnete Tabelle ein.

Anschließend schaltet er in den Modus 'neuen Text eingeben' um. Ihr Bildschirm muß mit der jetzt sortierten Tabelle so aussehen.

```

** CED Texteditor V1.0           Zeile:      6 Spalte:  19 Name: brief
-----
    Betr.: SINIX-Dokumentation.
           Die folgende Tabelle enthaelt Begriffe, wie sie aus der
           englischen Originaldokumentation fuer die SINIX-Dokumen-
           tation uebersetzt wurden.

           Datei                file
           Dateiverzeichnis     directory
           Geraetedatei         special file
           Prozess              process
           Schreibmarke         cursor
           Shell                shell

-----
Sie wollen neuen Text eingeben?
Bitte geben Sie Ihren Text ein!

```

Achtung

Es sind vor allem solche SINIX-Kommandos sinnvoll,

- die von der Standardeingabe lesen und auf die Standardausgabe schreiben, wie z.B. grep, sort, usw.
- die nur auf die Standardausgabe schreiben, z.B. who, date

SINIX-Kommandos, die eine Veränderung der Umgebung zur Folge haben, können zu seltsamen Ergebnissen führen.

Nachdem Sie Ihre Tabelle sortiert haben, möchten Sie vielleicht noch einen Schlußsatz unter diesen Brief setzen, z.B.:

Diese Liste wurde am Di 8.Nov 1987, 14:05:00 MEZ erstellt und soll laufend ergaenzt werden.

Wie bekommen Sie das aktuelle Datum in den Text? Natürlich wieder durch Angabe eines SINIX-Kommandos wie oben beschrieben.

Tippen Sie Ihren Text zunächst ohne das Datum in die oberste Leerzeile.
Ihr Brief könnte jetzt so aussehen:

** CED Texteditor V1.0

Zeile: 6 Spalte: 19 Name: brief

Betr.: SINIX-Dokumentation.

Die folgende Tabelle enthaelt Begriffe, wie sie aus der
englischen Originaldokumentation fuer die SINIX-Dokumen-
tation uebersetzt wurden.

Datei	file
Dateiverzeichnis	directory
Geraetedatei	special file
Prozess	process
Schreibmarke	cursor
Shell	shell

Diese Liste wurde am erstellt und soll laufend ergaenzt werden..

Sie wollen neuen Text eingeben?
Bitte geben Sie Ihren Text ein!

5.1.8 Wie können Sie einzelne Zeilen bearbeiten?

Um das Datum an der richtigen Stelle einzufügen, müssen Sie die gerade eingegebene Zeile an der richtigen Stelle teilen.

Im Modus 'eine Zeile bearbeiten' können Sie eine Zeile so teilen:

MENU

Z

Schreibmarke dorthin positionieren, wo die neue Zeile beginnen soll

T

Also:

Schalten Sie in den Modus 'eine Zeile bearbeiten' um. Positionieren Sie die Schreibmarke auf den ersten Buchstaben des Wortes 'erstellt', und teilen Sie die Zeile an dieser Stelle durch drücken der Taste **T**.

Der Rest der Zeile ab der Schreibmarkenposition, ist in eine neue, eingefügte Zeile geschoben worden.

Ihr Brief hat jetzt folgende Form:

** CED Texteditor V1.0
Zeile: 13 Spalte: 23 Name: brief

Betr.: SINIX-Dokumentation.

Die folgende Tabelle enthaelt Begriffe, wie sie aus der englischen Originaldokumentation fuer die SINIX-Dokumentation uebersetzt wurden.

Datei	file
Dateiverzeichnis	directory
Geraetedatei	special file
Prozess	process
Schreibmarke	cursor
Shell	shell

Diese Liste wurde am 12.11.84 erstellt und soll laufend ergaenzt werden.

Sie wollen neuen Text eingeben?
Bitte geben Sie Ihren Text ein!

Das Datum fügen Sie im Modus 'Kommando ausführen' mit Hilfe des SINIX-Kommandos *sie datum* in den Text ein.

Beachten Sie bitte, daß Sie explizit *keinen* Eingabebereich festlegen, sonst löscht der *ced* Teile des Textes. Ihre Kommandoeingabe muß dazu so sein:

0 sie datum 

Drücken Sie folgende Tasten:

Schreibmarke positionieren auf die zweite Zeile des geteilten Satzes





0 sie datum 

Das Datum wird über der aktuellen Zeile eingefügt.

```

** CED Texteditor V1.0           Zeile:   14 Spalte:   1  Name: brief
-----
    Betr.: SINIX-Dokumentation.
           Die folgende Tabelle enthaelt Begriffe, wie sie aus der
           englischen Originaldokumentation fuer die SINIX-Dokumen-
           tation uebersetzt wurden.

           Datei                file
           Dateiverzeichnis     directory
           Geraetedatei         special file
           Prozess              process
           Schreibmarke        cursor
           Shell                shell

Diese Liste wurde am
Mi 2.Dez.1987, 08:46:50 MEZ
erstellt und soll laufend ergaenzt werden.

-----
Sie wollen neuen Text eingeben?
Bitte geben Sie Ihren Text ein!

```

Um das Datum richtig in Ihren Schlußsatz einzufügen, müssen Sie die Zeile, die den ersten Teil des Satzes enthält und die Zeile, die das Datum enthält, zusammenfassen.

Drücken Sie folgende Tasten:

MENU

Z

Schreibmarke positionieren auf die erste Zeile des Nachsatzes

V

Nachdem Sie im Modus 'eine Zeile bearbeiten' die Schreibmarke auf der oberen Zeile des Nachsatzes positioniert haben, verbinden Sie die beiden Zeilen zu einer mit der Taste V. Eventuell überflüssige Leerzeichen können Sie jetzt entfernen mit der Taste CHAR DELETE bzw. fehlende einfügen im Modus 'Text einfügen', so daß Ihr Brief dann so aussieht:

```
** CED Texteditor V1.0           Zeile:   13 Spalte:  23  Name: brief
-----
    Betr.: SINIX-Dokumentation.
           Die folgende Tabelle enthaelt Begriffe, wie sie aus der
           englischen Originaldokumentation fuer die SINIX-Dokumen-
           tation uebersetzt wurden.

           Datei                file
           Dateiverzeichnis     directory
           Geraetodatei         special file
           Prozess              process
           Schreibmarke         cursor
           Shell                shell
Diese Liste wurde am Mi 2.Dez.1987, 09:04:11 MEZ
erstellt und soll laufend ergaenzt werden.

-----
Sie wollen neuen Text eingeben?
Bitte geben Sie Ihren Text ein!
```

Hinweis

Zeilen können Sie in diesem Modus nicht kopieren.

Wie Sie sicherlich schon gesehen haben, bietet der Modus 'eine Zeile bearbeiten' nicht die Möglichkeit, eine Zeile zu kopieren. Wollen Sie doch einmal einzelne Zeilen kopieren, schalten Sie in den Modus 'Zeilenbereich bearbeiten' um. Dann können Sie eine Zeile durch doppeltes Markieren als Bereich kennzeichnen und die gewohnten Operationen auf ihn anwenden.

Oder einfacher:

Benutzen Sie die Funktionstaste **F11**, um eine Zeile auszuwählen. Positionieren Sie jetzt die Schreibmarke an die neue Position und drücken Sie die Funktionstaste **F12**. Die mit **F11** ausgewählte Zeile wird oberhalb der aktuellen Zeile eingefügt.

Nehmen Sie an, Sie benötigen diese Tabelle auch noch in einer anderen Datei. Natürlich können Sie in der Shell den gesamten Inhalt der Datei *brief* in eine andere kopieren. Es geht aber auch viel einfacher. Der *ced* bietet Ihnen die Möglichkeit, in andere Dateien umzuschalten und Daten hin und her zu bewegen.

5.1.9 Daten bewegen von einer Datei in eine andere

Markieren Sie im Modus 'Zeilenbereich bearbeiten' die erste und letzte Zeile der Tabelle. Schalten Sie nun in den Modus 'Dokument wechseln'. Tragen Sie jetzt den Namen des anderen Dokuments, z.B. *brief.kopie* in die obere Zeile des Bedienbereichs ein und drücken Sie die Taste ☐. Existiert die Datei und können Sie auf sie zugreifen, zeigt der *ced* im Sichtbereich die ausgewählte Datei an. Die Titelzeile enthält den Namen der Datei; die Schreibmarke befindet sich am Anfang dieser Datei. Der *ced* ist im Modus 'neuen Text eingeben'.

Meldet der *ced*

***** DATEI ANLEGEN? (j,n):

konnte der *ced* die angegebene Datei nicht öffnen. Entweder ist sie noch nicht vorhanden oder Sie dürfen auf Sie nicht zugreifen.

Antworten Sie mit n,
so schaltet der *ced* in den Modus 'neuen Text eingeben' zurück.

Antworten Sie mit j,
so versucht der *ced*, die Datei anzulegen und schaltet bei Erfolg in die neue Datei um.

Kann der *ced* die Datei nicht anlegen, z.B. auf Grund fehlender Schreibberechtigung im angegebenen Dateiverzeichnis, meldet er:

***** DOKUMENT KANN NICHT GEOEFFNET/ERZEUGT WERDEN.

und schaltet zurück in den Modus 'neuen Text eingeben'.

Was heißt umschalten? Sie können jetzt in dieser Datei ohne Einschränkungen arbeiten. Sind die von Ihnen gesetzten Marken in der Datei *brief* noch vorhanden? Ja! Den markierten Bereich können Sie so bearbeiten, als hätten Sie die Datei *brief* nie verlassen. Was müssen Sie nun tun, um die Tabelle zu kopieren?

Die Tastenfolge nach dem Markieren der Tabelle in der Datei *brief* ist:

MENU

D

brief.kopie ↵

Auf die Frage: DATEI ANLEGEN? (j,n): antworten:

J

MENU

B

K

Im Modus 'Zeilenbereich bearbeiten' im Dokument *brief.kopie* drücken Sie einfach die Taste **K** und der im Dokument *brief* markierte Bereich - die Tabelle - wird in das Dokument *brief.kopie* kopiert.

Wie können Sie in das alte Dokument zurückschalten?

Am schnellsten geht es mit der Funktionstaste **T10**. Der Name des zuletzt aktiven Dokuments, d.h. des Dokuments, das Sie vor dem letzten Umschalten bearbeitet haben, ist intern abgelegt. Die Funktionstaste **T10** bewirkt den Aufruf der richtigen *ced*-Kommandos.

Wenn Sie auf die ausführliche Weise zurückschalten wollen, wählen Sie wieder den Modus 'Dokument wechseln' und drücken die Taste ↵. Sie befinden sich wieder im Dokument *brief* im Modus 'neuen Text eingeben'.

Auch die Funktionen, die den internen Zwischenspeicher benutzen, arbeiten zwischen Dateien.

Ein Beispiel. Um eine Zeile aus der Datei *brief* in die Datei *brief.kopie* zu verschieben, müssen Sie:

in der Datei *brief* in den

Modus 'eine Zeile bearbeiten' schalten,

Schreibmarke auf die zu löschende Zeile positionieren;

die Zeile löschen; sie befindet sich jetzt im Zwischenspeicher;

in den Modus 'Dokument wechseln' umschalten;

brief.kopie

in die Datei *brief.kopie* umschalten;

in den Modus 'eine Zeile bearbeiten' schalten,

Schreibmarke positionieren,

die Zeile aus dem Zwischenspeicher zurückholen.

Jetzt ist es an der Zeit, den so mühsam erstellten Brief zu sichern. Denn bis jetzt haben Sie nur auf einer temporären Datei gearbeitet.

Wie können Sie Ihre Datei sichern?

Wollen Sie Ihre Datei unter einem anderen Namen als den, den Sie beim Aufruf des *ced* angegeben haben, sichern oder einmal während der *ced*-Sitzung zwischensichern, so können Sie dies im Modus 'abspeichern' tun. Drücken Sie die Tasten:

Nach dem Drücken der Taste  , wird die Datei unter ihrem aktuellen Namen abgelegt - das ist der in der Kopfzeile angegebene. Wollen Sie sie unter einem Namen sichern, geben Sie bevor Sie die Taste  drücken, diesen Namen ein. Drücken Sie nun die Taste . Der *ced* rettet Ihre Datei unter diesem Namen. Nach erfolgreichem Abspeichern -Sicherung- ist der *ced* wieder im Modus 'neuen Text eingeben'.

Wollen Sie nur unter dem aktuellen Namen abspeichern, genügt auch schon ein Druck auf die Funktionstaste  . Ihr Dokument wird dann automatisch gespeichert.

5.1.10 Beenden Sie Ihre ced-Sitzung

Drücken Sie die Tasten:

jetzt meldet sich der *ced* mit der Frage:

Wollen Sie Ihren Text sichern? Geben Sie j = Ja oder n = Nein ein!

Achtung

Antworten Sie jetzt unvorsichtigerweise mit n, war Ihre ganze Arbeit umsonst. Alle Änderungen, die Sie während dieser *ced*-Sitzung gemacht haben, sind gelöscht, falls Sie diese nicht zwischendurch abgespeichert haben. Geben Sie dagegen ein j ein, so schreibt der *ced* Ihre Änderungen in die Datei: *brief*.

Haben Sie während der *ced*-Sitzung im Modus 'Dokument wechseln' andere Dateien geöffnet und diese noch nicht geschlossen, schaltet der *ced* jetzt zu der zuletzt aktiven um; in unserem Fall also zu *brief.kopie*. Diese können Sie jetzt auf dieselbe Weise sichern. Haben Sie während der *ced*-Sitzung noch weitere Dokumente geöffnet *und* verändert, so wechselt der *ced* automatisch zu all diesen Dateien. Sie können dann von Fall zu Fall entscheiden, ob Sie die Veränderungen in der jeweiligen Datei retten wollen oder nicht. Damit der *ced* weiterschaltet, müssen Sie die einzelnen Dateien im Modus 'verlassen' abspeichern oder nicht.

5.1.11 Sonstige Kommandos

Was gibt es noch für Modi und Kommandos?

Unser Beispiel ist zu kurz, um alle Modi und Kommandos des *ced* zu besprechen. Deshalb suchen Sie sich am besten eine Übungsdatei, die etwas größer ist als unsere Briefdatei. Oder, tippen Sie einfach ein etwas längeres Dokument in eine Datei.

Wie können Sie Texte in einem Dokument suchen?

Natürlich gibt es im *ced* auch die Möglichkeit, nach Zeichenketten zu suchen. Schalten Sie in den Modus 'suchen' um. Drücken Sie die Tasten:

In der oberen Zeile des Bedienbereichs können Sie die zu suchende Zeichenfolge eingeben. Alle Zeichen, auch führende und nachfolgende Leerzeichen, sind signifikant.

Drücken Sie jetzt die Taste oder die Taste , beginnt der *ced*, ab der Schreibmarkenposition nach der eingegebenen Zeichenfolge zu suchen. Hat der *ced* diese nicht bis zum Erreichen des Dateiendes gefunden, gibt er die Meldung aus:

***** TEXT NICHT GEFUNDEN

Wird die Zeichenkette gefunden, positioniert der *ced* die Schreibmarke auf deren ersten Buchstaben. Wollen Sie nach weiterem Auftreten der angegebenen Zeichenkette suchen, drücken Sie noch einmal die Taste oder .

Wie können Sie in der Datei vor der Schreibmarkenposition suchen?

Entweder Sie positionieren zuerst die Schreibmarke an den Anfang der Datei und beginnen dann zu suchen. Oder Sie suchen rückwärts, indem Sie den Suchvorgang durch Drücken der Taste **[↑]** auslösen.

Hinweis

Haben Sie eine Zeichenkette im Modus 'suchen' als Suchzeichenkette eingetragen, bleibt diese während der gesamten *ced*-Sitzung als solche erhalten. Wollen Sie sie löschen, drücken Sie im Modus 'suchen' die Tasten **[CTRL]** und **[X]** gleichzeitig.

Ist die Suchzeichenkette noch eingetragen, können Sie auch, ohne in den Modus 'suchen' zu schalten, nach dieser Zeichenkette suchen. Durch Drücken der Funktionstasten **[F13]** bzw. **[F14]**, schalten Sie automatisch in diesen Modus und starten das Suchen vorwärts - **[F13]** -, bzw. rückwärts - **[F14]**. Anschließend befinden Sie sich wieder in Ihrem Ausgangsmodus.

Achtung

Der *ced* versteht im Gegensatz zu vielen anderen Kommandos *keine* regulären Ausdrücke (vgl. Kommando *ed* im Buch *SINIX-Kommandos*).

Wie können Sie das Fenster über das Dokument verschieben?

Schalten Sie durch Drücken der Tasten **MENU** **F** in den Modus 'Fenster positionieren'.

Wollen Sie das Fenster auf eine bestimmte Zeile positionieren?

Tragen Sie die Zeilennummer - ohne Vorzeichen - im Bedienbereich ein und drücken Sie die Taste **↵** oder eine der Tasten **↑** oder **↓**. Das Fenster wird über den gewählten Bereich geschoben; die gewählte Zeile wird oberste Zeile im Fenster. Die Schreibmarke befindet sich in dieser Zeile. Ist die eingegebene Zahl größer als die Anzahl der Zeilen, springt der *ced* auf die letzte Zeile der Datei.

Wollen Sie eine bestimmte Spalte zur ersten, bzw. letzten im Fenster machen, benutzen Sie in diesem Modus die Taste **←** bzw. die Taste **→**.

Oder wollen Sie das Fenster in festen Schrittweiten zeilen- oder spaltenweise über dem Dokument verschieben? Dann tragen Sie die Schrittweite - mit einem Vorzeichen, die Art des Vorzeichens spielt keine Rolle - in das freie Feld im Bedienbereich ein. Mit den Tasten **←** bzw. **→** verschieben Sie das Fenster nach links bzw. rechts in der angegebenen Schrittweite; mit den Tasten **↑** bzw. **↓** nach oben bzw. unten.

Zur Erinnerung:

Auch mit den Tasten **↕** und **↘** - halbseitenweise -, und den Funktionstasten **F15** und **F16** können Sie das Fenster über das Dokument verschieben.

Wechseln in die Shell

Haben Sie sich auch schon einmal geärgert, daß Sie erst eine Editorsitzung beenden mußten, um ein einfaches Shell-Kommando auszuführen? Den *ced* können Sie vorübergehend unterbrechen, ohne daß das gerade bearbeitete Dokument erst gerettet werden muß. Drücken Sie die Tasten:

MENU

!

Der *ced* verabschiedet sich jetzt mit der Meldung:

>>> Aufruf des *cedsh* Kommandos, Rueckkehr zum *CED* mit Taste: **END**

und die Shell meldet sich mit dem Empfangs-Bereitschaftszeichen \$. In der Shell können Sie nun beliebige *SINIX*-Kommandos ausführen. Allerdings hat diese sog. temporäre Shell eine niedrigere Priorität, so daß Sie, wenn Sie umfangreichere Arbeiten in der Shell durchführen wollen, doch besser den *ced* ganz verlassen.

Wollen Sie die temporäre Shell wieder verlassen, drücken die Taste END anstatt einer Kommandoeingabe. Sie befinden sich dann wieder im *ced* und können Ihr Dokument weiter bearbeiten.

Haben Sie sich auch schon oft gewünscht, mit einem einzigen Tastendruck eine ganze Folge von Befehlen auslösen zu können? Der *ced* bietet Ihnen diese Möglichkeit.

Tasten mit beliebiger Zeichenfolge belegen

Hinweis

Folgenden Tasten kann man keinen Text zuweisen:

MENU , **P** , **↵** , **SHIFT** **F11** bis **SHIFT** **F20** , **PRINT**

Tastenbelegungen bleiben auch über die *ced*-Sitzung hinaus erhalten.

Beim Verlassen des *ced* werden Ihre Tastenbelegungen automatisch für Sie *gespeichert*. Wenn Sie den *ced* wieder aufrufen, gelten dann wieder Ihre Tastenbelegungen.

Wie können Sie einer Taste einen Text zuweisen?

Schalten Sie in den Modus 'Tasten belegen' um: **MENU** **P** , tippen Sie Ihre gesamte Zeichen- und Befehlsfolge ein, schalten ein zweites Mal in den Modus 'Tasten belegen' und drücken Sie die Taste, der Sie den gespeicherten Text zuweisen wollen.

Das klingt einfacher, als es meistens ist. Deshalb soll an zwei Beispielen die Belegung von Tasten erklärt werden.

Nehmen Sie an, Sie wollen einen Bericht über den *Mississippi* schreiben. In diesem Bericht wird der Name Mississippi oft vorkommen. Anstatt ihn nun jedes Mal neu einzutippen, können Sie auch die Zeichenkette Mississippi einer sonst nicht genutzten Taste zuordnen. Anschließend reicht dann *ein* Tastendruck, um das Wort Mississippi in den Text einzufügen.

Um eine Taste mit der Zeichenkette Mississippi zu belegen, schalten Sie zuerst in den Tastenbelegungsmodus um: **MENU** **P** . Der *ced* schaltet sofort wieder in den Modus 'neuen Text eingeben' oder 'einfügen' zurück.

Aber - Sie sind noch im Modus 'Tasten belegen', was Sie nur an dem Wort **PROTO**, erkennen können, das in der Kopfzeile des Editierraumens, gleich rechts neben der Versionsnummer des *ced* steht. Ab jetzt wird jedes von Ihnen eingegebene Zeichen vom *ced* gespeichert. Tippen Sie also Mississippi ein. Der Text Mississippi erscheint so im Dokument, als hätten Sie ihn im Texteingabemodus eingegeben. Jetzt wollen Sie den Text beenden und einer Taste zuweisen.

Schalten Sie ein zweites Mal in den Modus 'Tasten belegen': MENU P

Der *ced* wird Sie auffordern:

***** GEWUENSCHTE TASTE DRUECKEN

Die Taste, die Sie als nächste drücken, erhält die Zuordnung der Zeichenkette, die Sie soeben eingegeben haben. Üblicherweise benutzt man entweder selten gebrauchte Sonderzeichen, eine der vom *ced* nicht verwendeten Funktionstasten, außer den oben ausgeschlossenen, oder eine CTRL-Tastenkombination.

Drücken Sie z.B. gleichzeitig die Tasten CTRL A, so wird dann jedes Mal, wenn Sie von nun an diese Tastenkombination drücken, der Text Mississippi in Ihr Dokument eingetragen.

Achtung

Wie Sie gemerkt haben, führt der *ced* alle Ihre Eingaben sofort aus. D.h. im Modus 'Tasten belegen' können Sie sofort Ihre eingegebene Zeichenkette oder Kommandofolge überprüfen.

Nun ein zweites Beispiel, etwas komplexer.

Hier sollen zu der Zeichenfolge auch einige *ced*-Kommandos gehören.

Stellen Sie sich vor, Sie wollten das Wort *ced* an einigen Stellen im Text durch das Wort Editor ersetzen. Ideal wäre, wenn Sie für die beiden Tätigkeiten - suchen und ersetzen - nur je einen Tastenanschlag benötigen.

Mit den bisher bekannten Kommandos ist das ziemlich umständlich: Sie müssen jedes Mal in das Menü Suchfunktionen umschalten, um das nächste Auftreten von *ced* zu finden. Wenn Sie dann entschieden haben, daß Sie es durch Editor ersetzen wollen, tippen Sie im Modus 'neuen Text eingeben' die ersten drei Buchstaben: Edi, schalten dann in den Modus 'einfügen' und tippen die restlichen drei Buchstaben: tor ein. Jetzt schalten Sie wieder in den Modus 'suchen' usw.

Durch das Belegen einer Taste, können Sie sich die Arbeit erheblich vereinfachen. Legen Sie zuerst die Suchfunktion auf die Tastenkombination CTRL B. Dabei müssen Sie folgendes beachten: Das Eintragen der Suchzeichenkette *ced* sollte nicht mit zur Tastenbelegung gehören! Da der *ced* den Sucheintrag nach dem Suchen nicht löscht, würde jedes weitere Eintragen die Zeichenkette verlängern. Stattdessen können Sie sich dieses Gedächtnis zunutze machen.

Schalten Sie zuerst in den Modus 'suchen' und tragen Sie ced als Suchzeichenkette ein. Ohne die Taste zu drücken und damit den Suchvorgang auszulösen, schalten Sie dann in den Modus 'Tasten belegen' um.

In der Kopfzeile muß jetzt PROTO erscheinen, d.h. alles, was Sie von nun an eingeben, ordnen Sie nachher einer Taste zu:

Schalten Sie zuerst in den Modus 'suchen' um. (Der Sucheintrag ced ist noch vorhanden.) Drücken Sie die Taste . Der ced beginnt, nach dem nächsten Auftreten von ced zu suchen. Ungeachtet, ob Sie dabei ced im Text gefunden haben oder nicht, schalten Sie wieder in den Modus 'neuen Text eingeben' und beenden jetzt die Befehlsfolge, indem Sie wieder (jetzt das 2. Mal) in den Modus 'Tasten belegen' schalten. Sie drücken die Tasten: .

Auf die Aufforderung: ***** GEWUENSCHTE TASTE DRUECKEN

drücken Sie jetzt gleichzeitig die Tasten und .

Sie können jetzt ausprobieren, ob Sie alles richtig gemacht haben, indem Sie mehrere Male drücken. Der ced sollte dann nach dem jeweils nächsten Auftreten von ced suchen.

Die zweite Funktion, das Ersetzen von ced durch Editor legen Sie jetzt auf die Tasten . Positionieren Sie die Schreibmarke auf das c von ced, so wie es die ced-Suchfunktion beim Auffinden der Zeichenkette ced machen würde. Schalten Sie wieder in den Modus 'Tasten belegen'. Überschreiben Sie jetzt das Wort ced mit Edi. Schalten Sie um in den Modus 'einfügen' und tippen Sie die restlichen drei Buchstaben: tor ein. Schalten Sie wieder in den Texteingabemodus zurück. Beenden Sie jetzt den Modus 'Tasten belegen' und weisen Sie Ihre Eingabe der Tastenkombination zu.

Jetzt haben Sie die Möglichkeit, mit zwei Tastenanschlügen das Wort ced durch Editor zu ersetzen. Positionieren Sie zuerst die Schreibmarke auf den Anfang Ihres Dokuments. Mit der Tastenkombination suchen Sie jetzt das nächste Auftreten des Wortes ced; wollen Sie es durch Editor ersetzen, drücken Sie .

Tastenbelegung anzeigen

Wenn das nicht so funktioniert wie beschrieben, lassen Sie sich Ihre Tastenbelegungen anzeigen. Drücken Sie die Tasten:

MENU X

Die Ausgabe müßte so aussehen:

⌞B: MENU s RETURN MENU n
⌞C: E d i MENU e t o r MENU n

Zur Ausgabe:

RETURN bezieht sich auf die Taste ↵,

⌞B bedeutet z.B. die Tastenkombination CTRL und B,

Schreibmarkenbewegungen werden durch einfache Pfeile oder

UP,	Taste ↑
DOWN,	Taste ↓
HOME,	Taste ↶
PJUMP,	Taste ⏮
MJUMP	Taste ⏭

angezeigt. Funktionstasten werden als $P_n - 1 < n < 20$ - bzw. F_n , wenn gleichzeitig mit der Funktionstaste die Taste SHIFT gedrückt wurde, bezeichnet.

Tastenbelegung ändern

Schon belegte Tasten können Sie jederzeit neu belegen. Wollen Sie eine Tastenbelegung löschen, schalten Sie direkt nach dem ersten Umschalten in den Modus 'Tasten belegen' ein zweites Mal in ihn um. Die so entstandene leere Zeichenfolge weisen Sie der Taste zu, deren Belegung Sie löschen wollen.

Bildschirm neu aufbauen

Schickt Ihnen während einer *ced*-Sitzung an einem Mehrplatzsystem ein anderer Benutzer eine Meldung mit dem write-Kommando, wird durch die auf dem Bildschirm erscheinende Meldung der Bildschirmaufbau des *ced* zerstört. Auch durch Übertragungsfehler in der Leitung kann der Bildschirmaufbau verändert werden. Um den alten Bildschirminhalt wieder

abgebildet zu bekommen, drücken Sie die beiden Tasten **CTRL** und **R** gleichzeitig. Der Bildschirm wird gelöscht und so wieder aufgebaut, wie er ausgesehen hatte, bevor er zerstört wurde.



5.2 Der Editor vi . Eine Beispielsitzung

Welches Ziel hat diese Beispielsitzung?

Sie wollen einen Text schreiben z.B. den Quelltext für ein Programm, einen Brief oder anderen Text.

Diese Beschreibung soll Ihnen helfen, solche Arbeiten mit dem Editor *vi* zu machen. Hier finden Sie die Schritte, die Sie im *vi* gehen müssen, um Ihre Vorstellungen zu verwirklichen.

Mit der Kommandobeschreibung alleine (vgl. Buch "SINIX-Kommandos") ist das nur mühsam zu erreichen, da der *vi* viele Kommandos und Kommandovarianten kennt und seine gezielte Nutzung etwas Übung erfordert.

Deshalb sind hier einige Einführungsbeispiele gegeben, die Ihnen den Einstieg sehr einfach machen. Die Anfangsschritte sind relativ ausführlich behandelt, damit Sie die prinzipielle Arbeitsweise des *vi* kennenlernen. Die Erklärung der *vi*-Kommandos ist kürzer gefaßt und mehr auf Problembeispiele und Lösungsbeispiele ausgerichtet.

Was können Sie mit dem *vi* machen?

Hier sind einige Möglichkeiten aufgeführt:

- Datei einrichten
- Text eingeben, korrigieren
- Textteile anfügen, einfügen, löschen, ersetzen, kopieren
- Zeilen einfügen, löschen, teilen, miteinander verbinden
- Zeichen anfügen, einfügen, löschen, ersetzen
- Zeichenketten suchen, löschen, ändern
- In andere Dateien wechseln
- Temporär in die Shell wechseln
- Shell-Kommandos ausführen
- Puffer mit Text oder Kommando laden
- Voreinstellungen vornehmen

Beginnen Sie mit einem kleinen Beispiel

Sie sind am SINIX-System angeschlossen und die Shell fordert Sie mit \$ auf, ein Kommando einzugeben. Sie rufen den Editor *vi* auf:

\$ vi dateiname

Bereits im *vi*-Aufruf können Sie einen Dateinamen angeben - oder nicht. Existiert die Datei nicht, so wird sie vorläufig angelegt.

Existiert die Datei, so wird eine Arbeitskopie im Editor-Puffer angelegt. Ist *dateiname* nicht angegeben, so arbeiten Sie mit dem *vi*, für die Dauer Ihrer Editorsitzung, mit dem Editor-Puffer. Sie arbeiten also in jedem Falle erst einmal mit dem Editor-Puffer.

Wenn Sie nach Ihrer Arbeit die Editorsitzung beenden, fragt Sie der *vi*, ob Sie den Editor-Puffer speichern, d.h. sichern wollen oder nicht. Wenn ja, so muß dem *vi* ein Dateiname bekannt sein, vom *vi*-Aufruf her oder Sie müssen ihm dann dazu einen Dateinamen angeben. Existierte bereits eine Datei mit dem gleichen Namen, so wird sie überschrieben.

Von Dateien, mit für Sie wichtigem Inhalt, sollten Sie zumindest in Ihrer Lern-Anfangszeit, vor dem Editieren eine *Kopie* anlegen.

Für unser Beispiel wählen wir den Dateinamen *tel-liste*.

Diese Datei soll im aktuellen Dateiverzeichnis noch nicht existieren.

Es soll nun eine kleine Telefonliste mit insgesamt nur vier Zeilen erstellt und gespeichert werden. Sie soll so aussehen:

Telefonliste

Adam 4999

Eva 3210

Der *vi* wird aufgerufen:

\$ vi tel-liste

Nach dem Aufruf sehen Sie am Bildschirm am linken Rand die Zeichen ~. So sind alle 23 Zeilen gekennzeichnet, die noch leer sind. In der letzten Bildschirmzeile, der 24. Zeile, gibt Ihnen der *vi* eine Meldung aus:

"tel-liste" [New file]

(Beim *vi* ist die 24. Zeile für die Ausgabe von Editor-Meldungen oder Systemausgaben oder für die Eingabe von *ex*-Editor-Kommandos reserviert.)

Der *vi* hat nun einen Arbeits-Speicherbereich für Ihren Text eingerichtet.

Ohne Kommando geht gar nichts

Wenn Sie jetzt beginnen wollen Text einzugeben, zeigt der *vi* keine Reaktion oder er tut etwas Unerwartetes! Das ist richtig.

Der *vi* befindet sich nach dem Aufruf in einem Modus, in dem er ein Kommando erwartet.

Drücken Sie jetzt die Taste für a.

A

Am Bildschirm ändert sich noch nichts. Mit dem a haben Sie den *vi* in den "Eingabemodus" umgeschaltet. Er versteht a als Kommando (append = anfügen; alleine den Buchstaben a in Kleinschreibung.) Das nächste eingegebene Zeichen wird als Textzeichen erkannt.

Viele *vi*-Kommandos haben nur ein Zeichen oder wenige mehr. Sie wirken sofort (ohne die Taste ↵). Das eingegebene Kommando wird nicht angezeigt!

Schreiben Sie jetzt den Text:

Telefonliste

Der Text wird im Editor-Puffer eingetragen und am Bildschirm angezeigt
Sie können jetzt weiterschreiben, wie mit einer Schreibmaschine.

Eine neue Zeile erhalten Sie mit Taste .

Zum Korrigieren können Sie die Taste  benutzen.

Drücken Sie die Taste .

Sie erhalten eine Leerzeile.

Drücken Sie nochmals die Taste .

Schreiben Sie nun die beiden Zeilen

Adam 4999

Eva 3210

Dieses kleine Telefonverzeichnis soll nun fest abgespeichert werden, d.h.
mit dem Verlassen des *vi* nicht verloren gehen.

Dazu benutzen Sie wieder ein Kommando. Vorher müssen Sie aber erst
wieder in den "Kommandomodus" zurückschalten. Dazu drücken Sie die
Taste ESC.



Geben Sie jetzt ZZ ein.

(ZZ als direkt aufeinanderfolgende Großbuchstaben ohne )

Damit wird der Text gespeichert und außerdem die *vi*-Sitzung beendet:

Der *vi* gibt in der untersten Bildschirmzeile die Meldung aus:

"tel-liste" [New file] 4 lines, 32 characters

Sie erhalten wieder das Zeichen \$ am Bildschirm.

Der *vi* ist jetzt beendet. Die Datei tel-liste ist erzeugt. Auf den Editor-Puffer
haben Sie nun keinen Zugriff mehr.

Schauen Sie noch einmal zurück. Das war die *Folge der Eingaben*:

\$ vi tel-liste	Aufruf des <i>vi</i> (mit Dateiname)
a	<i>vi</i> -Kommando: Text anfügen (append)
text	Text für 1. Zeile
	neue Zeile 2. Zeile, bleibt leer
	neue Zeile
text	Text für 3. Zeile
	neue Zeile
text	Text für 4. Zeile
	In Kommandomodus zurückschalten
ZZ	Editor-Puffer sichern und <i>vi</i> beenden
\$	

Wenn Sie Ihre neue Datei am Bildschirm kontrollieren wollen, so geben Sie z.B. das SINIX-Kommando:

\$ cat tel-liste

Am Bildschirm erhalten Sie dann die Ausgabe:

Telefonliste

Adam 4999

Eva 3210

Wenn Sie Ihre kleine Telefonliste am Drucker ausgeben wollen, so geben Sie das SINIX-Kommando:

\$ lpr tel-liste

5.2.1 Der vi arbeitet in verschiedenen Modi

Sie geben Zeichen von der Tastatur her ein. Der *vi* reagiert darauf verschieden, je nachdem in welchem Modus er sich befindet.

Prinzipiell unterscheidet der *vi* zwei Modi

- **Kommandomodus**
- **Eingabemodus**

Nach dem Start befindet sich der *vi* immer im Kommandomodus und erwartet ein *vi*-Kommando vom Benutzer.

Wenn Sie ein Kommando geben, das auf eine Texteingabe oder Texterweiterung abzielt, so schaltet der *vi* in den Eingabemodus, z.B. nach einem der Kommandos

- a (append) Text anfügen
- i (insert) Text einfügen

Das Nächste eingegebene Zeichen wird gleich als Text-Zeichen verwendet!

Zum *Eingabemodus* ist noch zu erwähnen, daß man speziell bei der Anwendung von *vi*-Funktionen zum Text-Einfügen, wie z.B. bei Verwendung eines der beiden oben genannten *vi*-Kommandos, von diesem Modus spricht:

- **Einfügemodus**

Beachten Sie bitte, daß der *vi* eine Kommandoeingabe nicht extra bestätigt. Sie können aber eine Aktion schnell erkennen, wenn Sie z.B. mit dem Kommando x ein Zeichen löschen und es am Bildschirm sofort verschwindet.

Zum Rückschalten in den Kommandomodus ist für den *vi* grundsätzlich die Taste `[ESC]` festgelegt. Nach dem Betätigen dieser Taste wird das nächste Zeichen immer als Kommando-Information verwendet. Nach einigen Kommandos schaltet der *vi* auch selbsttätig wieder in den Kommandomodus um.

Sollten Sie einmal nicht mehr sicher sein, ob sich der *vi* im Kommanomodus oder im Eingabemodus befindet, so betätigen Sie einfach die Taste `ESC`. Damit haben Sie sich wieder eine sichere Ausgangsposition geschaffen, den Kommandomodus.

Drücken Sie die Taste `ESC` lieber einmal zuviel, als einmal zu wenig.

Bei notwendigen Text-Korrekturen macht man sonst anfangs noch Bedienungsfehler. Aber etwas Geduld und Übung bringen Sie bald zur Meisterschaft.

5.2.2 Kommandoformen

Im *vi* gibt es mehrere Kommandoformen.

Nehmen Sie bitte zu diesen Erklärungen auch das Buch *SINIX Kommandos* zur Hand und vergleichen Sie bitte bei Kommando *vi* immer gleich die hier angegebenen Kommandos. Auch die Kommandos *ed* und *ex* sind in diesem Zusammenhang zum Lernen interessant. Da in dieser Beschreibung nicht alle Kommandos behandelt sind und es auch viele Kombinationsmöglichkeiten gibt, sehen Sie dabei gleich weitere Kommando-Möglichkeiten für Aufgabenlösungen.

Die einfachsten Kommandos bestehen aus einem *Buchstaben in Kleinschreibung*:

a	(append)	Text anfügen
i	(insert)	Text einfügen
x		aktuelles Zeichen löschen
u	(undo)	letztes Änderungs-Kommando rückgängig machen

Ebenso gibt es Kommandos mit einem *Buchstaben in Großschreibung*; sie entsprechen oft den Funktionen der Kommandos in Kleinschreibung, aber mit verstärkter oder besonderer Wirkung:

A	(append)	Text anfügen am Zeilenende
I	(insert)	Text einfügen am Zeilenanfang
X		Zeichen löschen vor der Schreibmarke
U	(undo)	ganze aktuelle Zeile wieder herstellen

Viele Kommandos können durch Angabe einer Zahl direkt vor dem Kommando *mehrfach ausgeführt* werden:

7x löscht 7 Zeichen ab der Position der Schreibmarke, rechts

4X löscht 4 Zeichen vor der Position der Schreibmarke, links

Nachdem der *vi* eine Weiterentwicklung von früheren Editoren ist und deren Funktionen möglichst beibehalten wurden, wird beim *vi* auch relativ oft die Taste CTRL in Verbindung mit einem Buchstaben verwendet:

CTRL d	(down) Fenster um halben Bildschirm vorwärts schieben
CTRL u	(up) Fenster um halben Bildschirm rückwärts schieben

Eine weitere Kommandovariante sind die Kommandos, die von früheren Editorentwicklungen übernommen wurden (von *ed* und *ex*). Diese *ex-Kommandos* beginnen im *vi* immer mit einem *Doppelpunkt* und werden immer in der untersten Bildschirmzeile (Statuszeile) eingegeben. Im Unterschied zu den anderen Kommandos *müssen* diese abgeschlossen werden, mit der Taste `[↵]` oder mit `[ESC]`.

Wenn in den folgenden Erklärungen Kommandos mit vorne einem `:` angegeben sind, so ist aus Vereinfachungsgründen der dabei immer erforderliche Kommando-Abschluß mit `[↵]` oder `[ESC]` nicht immer eigens angegeben!

`:w [ESC]` (write) Text aus Editor-Puffer abspeichern, in bekannte Datei

`:q [ESC]` (quit) *vi*-Editor beenden

(Diese beiden Kommandos haben die gleiche Wirkung wie das *vi*-Kommando ZZ.)

In diesen speziellen Kommandomodus, den Ex-Modus, schalten Sie durch Eingabe von `[ESC]` und anschließender Eingabe eines dieser Zeichen:

`:` Doppelpunkt

`[Q]`

Q Buchstabe in Großschreibung

Die Schreibmarke springt dann automatisch in die Statuszeile. Dort wird am Zeilenanfang ein Doppelpunkt angezeigt. Die Schreibmarke steht rechts daneben. Wenn Sie von dem `:` nicht mehr loskommen, geben Sie *vi* ein.

Wenn Sie z.B. ein *w* eingeben, sehen Sie in der letzten Bildschirmzeile `:w`. Es geschieht noch nichts. Sie könnten die Kommandoangabe z.B. noch ändern, indem Sie die Schreibmarke mit der Taste `[←]` nach links bewegen und das Kommando überschreiben.

Mit der Eingabe von `[ESC]` oder `[↵]` wird das Kommando ausgeführt. Nach der Kommandoausführung ist der *vi* wieder im Kommandomodus. Nach dem Kommando `:w` erhalten Sie in der Statuszeile die Meldung:

"datei" xxx lines, yyyy characters

Die Meldung gibt Ihnen den Namen der Datei an, die Anzahl der Zeilen und die Anzahl der Zeichen. Solche Meldungen bleiben dort stehen, bis Ihr Text am Bildschirm auf beliebige Art bewegt wird.

Die Schreibmarke springt normalerweise wieder an die aktuelle Textstelle zurück, wenn das Kommando abgeschlossen ist. Sie können dann Ihren Text weiter bearbeiten. Bei Situationen, die die Schreibmarke nicht zurückführen, benutzen Sie eine dieser Tasten zum Unterbrechen oder Abschluß des Vorganges.

 oder  oder 

Bei den Ex-Kommandos können Sie bei vielen Kommandos auch einen *Wirkungsbereich* angeben, den Sie vor das Kommando schreiben.

:20,\$w datei

Bei diesem Beispiel wird der Editor-Puffer ab Zeile 20 bis zum Dateiende in die angegebene Datei gespeichert.

(Bei *ex*-Kommandos gilt \$ als Teil der Bereichsangabe symbolisch für "Dateiende". Als *vi*-Kommandos setzt \$ die Schreibmarke an das Zeilenende, sie kennzeichnet das letzte Zeichen der aktuellen Zeile.)

Wenn Sie das Kommando so eingeben

:20w datei

wird nur alleine die Textzeile mit der laufenden Nummer 20 gesichert. (Die erste Zeile des Editor-Puffers hat die Nummer 1.)

Den ganzen Editor-Puffer können Sie so sichern, mit

:1,\$w datei

Wenn Sie den ganzen Editor-Puffer sichern wollen, in die dem *vi* schon bekannte Datei, so ist das ein spezieller Fall. Dann reicht alleine :w.

Wenn Sie :wq eingeben, wird der Editor-Puffer gesichert und der *vi* beendet.

Man kann auch die *aktuelle Zeile* in der Bereichsangabe angeben mit "Punkt".

:.w datei schreibt *nur* die aktuelle Zeile in datei.

:. , + 20w datei schreibt die aktuelle Zeile und 20 weitere in datei.

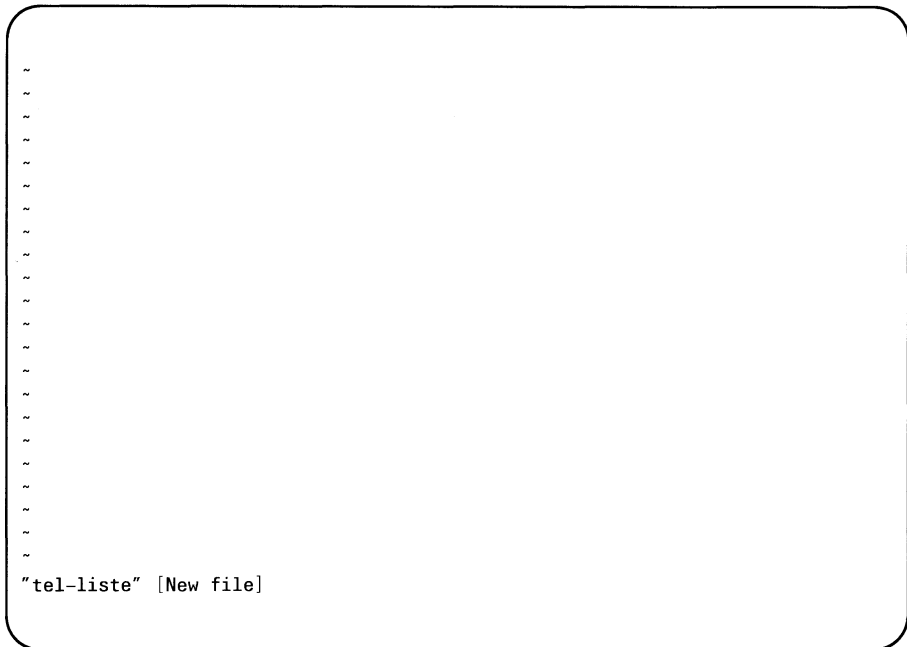
:. , , 20 datei schreibt auch die aktuelle Zeile und 20 weitere in datei.

Manche *ex*-Kommandos überschneiden sich in der Wirkung mit *vi*-Kommandos, so positioniert z.B. 20G das Fenster auf die 20. Zeile; ebenso aber auch :20 . Welche Kommandoart Sie verwenden wollen, ist Ihnen freigestellt. Die Praxis wird dann zeigen, welche Kommandoart Ihnen besser zusagt.

5.2.3 Der Bildschirm als Arbeitsfeld

Nach dem Aufruf des *vi* erhalten Sie am Bildschirm dieses Bild, wenn als Dateiname z.B. *tel-liste* angegeben war und die Datei noch leer ist.

Der Bildschirm hat 24 Zeilen und 80 Spalten.



Das ist der Anfangs-Bildschirm des *vi* .

Die Zeichen ~ kennzeichnen 23 Zeilen, die noch leer sind. Die Zeichen ~ sind nur Platzhalter und verschwinden bei einer Texteingabe.

Enthält die Datei bereits Text, z.B. 10 Zeilen, so wird der Text angezeigt und die verbleibenden Zeilen mit ~ gekennzeichnet.

Diese 23 Zeilen sind Ihr Text-Bereich am Bildschirm.

Die 24. Zeile, also die letzte Zeile, ist reserviert und dient *ausschließlich* zur Eingabe von :-Kommandos oder zur Ausgabe, z.B. von Meldungen.

Wenn Sie Text eingeben über den rechten Bildschirmrand hinaus, so wird er auf der nächsten Bildschirmzeile weitergeschrieben. Sie können bis zu 1022 Zeichen als Zeile eingeben. (Beim *vi* gibt es daher keine Kommandos für Bild-Rechts-Links-Verschiebung.)

Das Zeilenende können Sie normalerweise nicht erkennen. Wenn es für Sie aber erforderlich ist, geben Sie dieses Kommando ein

```
:set list
```

Es werden dann alle Zeichen sichtbar, z.B.:

```
Zeilenende    als    $
Escape        als    ^[
Tabulator     als    ^I
```

Das Kommando können Sie rücksetzen mit

```
:set nolist
```

Dann ist wieder alles wie vorher.

Erreichen Sie mit Ihren Texteingaben die 23. Zeile, so wird das Bild automatisch mit dem Schalten auf die neue Zeile (z.B. mit ) um eine Zeile nach oben verschoben.

Sie können auch  f eingeben, dann springt die vorletzte Textzeile an den Bildschirmanfang. Die Möglichkeiten zum Verschieben des Fensters sind einige Seiten später zusammengestellt.

Wenn Sie eine größere Datei bearbeiten und sich orientieren wollen, welche Zeile Sie erreicht haben, so können Sie sich alle Textzeilen mit *Zeilennummern* ausgeben lassen.

```
:set number    oder :se nu (abgekürzte Eingabe)
```

Die Zeilennummern werden nur angezeigt, Ihr Editor-Puffer bleibt aber unverändert!

Das Kommando können Sie rücksetzen mit

```
:set nonumber  oder :se nonu
```

Interessiert Sie nur die *Nummer der aktuellen Zeile* (in der die Schreibmarke steht), geben Sie eines dieser Kommandos an:

CTRL g	gibt Meldung aus: "datei" line xxx of xxx --yy%--
:. =	gibt Zeilennummer aus
:nu	gibt Zeilennummer und Textzeile aus
: =	gibt Zeilennummer der letzten Zeile des Editor-Puffers
:f	gibt Meldung aus: "datei" line xxx of xxx --yy%--
line xxx	ist die Nummer der aktuellen Zeile
of xxx	ist die Anzahl aller Zeilen
%	ist Prozentwert der akt. Zeile (100% = Dateiende)

Wollen Sie den Bildschirm neu aufbauen, z.B. nach einer Meldung, die Ihr Textbild verändert hat, so verwenden Sie dieses Kommando:

CTRL l

Wenn Sie in einem umfangreicheren Dokument blättern und z.B. mit dem Kommando G das Ende der Datei sehen wollen, so ist der *vi* immer bemüht, die aktuelle Zeile in der *Bildschirmmitte* zu zeigen. Bei G zeigt *vi* die letzte Zeile der Datei auf der 11. Bildschirmzeile.

5.2.4 Aufrufen des vi

Den *vi* rufen Sie mit dem SINIX-Kommando *vi* auf und schließen, wie üblich, mit  ab.

Sie können das Kommando alleine angeben oder nach dem Kommandonamen weitere Angaben machen, durch Leerzeichen getrennt:

<i>vi</i>	<i>vi</i> legt temporäre Datei an (Editor-Puffer)
<i>vi</i> [<i>␣</i> datei]	<i>vi</i> zeigt Datei ab ihrem Anfang
<i>vi</i> [<i>␣</i> -R][<i>␣</i> datei]	<i>vi</i> zeigt Datei, (read only = nicht zum Schreiben) Ändern ist möglich, sichern in Datei mit anderem Namen ist möglich
<i>vi</i> [<i>␣</i> -r][<i>␣</i> datei(en)]	<i>vi</i> versucht Datei wieder herzustellen (recover), (falls der Inhalt des Editor-Puffers zerstört wurde).

Hier sind noch weitere Angaben zum Aufruf:

vi Der *vi* wird gestartet. Der Editor-Puffer ist leer und keiner Datei zugeordnet. (Dateiname wird bei *vi*-Abschuß angefordert.)

vi datei Ist die Datei nicht leer, so wird sie in den Editor-Puffer geladen. Der Bildschirm zeigt sie ab ihrem Anfang. Die ursprüngliche Datei bleibt unberührt erhalten.

Es können auch mehrere Dateinamen angegeben werden, durch Leerzeichen getrennt.

(Sie bearbeiten dann die erste Datei, speichern Sie mit :w und schalten dann mit :n in die nächste Datei. Mit :e# können Sie dann jeweils auf die vorherige Datei zurückschalten, wenn Sie wollen abwechselnd. Mit :n datei können sie die Namensliste im *vi* ändern.)

vi + datei Am oberen halben Bildschirm wird das Ende der Datei gezeigt. Sie können ab der Bildschirmmitte sofort weiterarbeiten.

vi + n datei Zeigt die Datei ab der Zeile n (n ist eine Zahl). Zeile n steht in Bildschirmmitte.

vi + /pat datei

Zeigt die Datei ab der Zeile, die das angegebene Zeichenmuster pat (pattern = Muster) enthält.

Beispiel

vi + /Bei datei

Die Datei wird in den Editor-Puffer geladen und nach der Zeichenkette "Bei" durchsucht. Am Bildschirm wird die erste Trefferstelle angezeigt, z.B. bei dem Wort Beispiel. Die Zeile mit dem gefundenen Muster steht in Bildschirmmitte.

(Mit dem *vi*-Kommando *n* könnten Sie dann gleich die nächste Stelle mit diesem Muster suchen - *vi* merkt sich das Muster.)

5.2.5 Beenden des vi und Sichern des Editor-Puffers

Auch zum Beenden des *vi* haben Sie mehrere Möglichkeiten. Vor dem Beenden des *vi* muß man überlegen, was mit dem Text im Editor-Puffer geschehen soll. Zu diesem Zeitpunkt existiert nämlich dieser Text nur dort, wenn er neu oder geändert ist.

- :w Der Editor-Puffer wird in die bekannte Datei gespeichert. Die Datei wird neu angelegt oder überschrieben! (Auch während Ihrer Arbeit können Sie dieses Kommando geben und damit den jeweils aktuellen Stand sichern. Der Bildschirm bleibt unverändert. Sie können ungestört weiterarbeiten.)
- :q *vi* wird beendet, wenn vorher der Editor-Puffer gesichert wurde. Bei nicht gesicherten Änderungen wird das Kommando nicht ausgeführt und eine Meldung ausgegeben.
- :wq Editor-Puffer sichern und *vi* beenden.
- :q! Editor-Puffer **geht verloren!** und *vi* wird beendet.
- :x Wie :wq
- ZZ Wie :wq
- :w datei ganzen Editor-Puffer sichern in Datei (mit anderem Namen)
- :wq datei ganzen Editor-Puffer sichern in Datei und *vi* beenden

Beispiel

:w neudatei 

Sie speichern Ihren Editor-Puffer in eine Datei mit anderem Namen.

Hinweis

Ist die eigene Datei mit "read only" (nur lesen) belegt oder Sie wollen eine andere bereits existierende Datei überschreiben, so benutzen Sie das Zeichen ! nach dem Kommando.

:w! datei

Die Datei wird dann überschrieben, ob Sie es wollten oder nicht.

5.2.6 SINIX-Kommandos ausführen

Es kann durchaus vorkommen, daß Sie während der *vi*-Sitzung schnell einmal ein SINIX-Kommando ausführen wollen, z.B. um die Namen Ihrer Dateien zu überprüfen.

Der *vi* stellt Ihnen diese Möglichkeiten zur Verfügung:

:sh Der *vi* wird unterbrochen. Ihr Editor-Puffer bleibt unverändert. Die Shell meldet sich mit \$. In der Shell können Sie nun beliebige SINIX-Kommandos ausführen, z.B. mit *ls* Ihre Dateinamen ausgeben. (Für größere Aktionen sollten Sie aber den *vi* verlassen.)

In den *vi* kommen Sie wieder zurück mit der Taste **END**.

:r!cmd Das SINIX-Kommando wird ausgeführt und das *Ergebnis eingefügt*, hinter der aktuellen Zeile (read).

!:cmd Der *vi* wird nicht verlassen und Ihr Text *nicht verändert*! Das SINIX-Kommando *cmd* wird ausgeführt und ggf. eine Ausgabe ab der Statuszeile ausgegeben. Sie werden dann aufgefordert mit **↵** wieder zurückzuspringen, mit der Meldung: [Hit return to continue].

!!cmd Das SINIX-Kommando wird ausgeführt und das *Ergebnis eingefügt*, ab der aktuellen Zeile (sie wird ggf. überschreiben).

!! Das letzte gegebene SINIX-Kommando wird wiederholt. Ein Ergebnis wird immer ab der Statuszeile ausgegeben. Der Editor-Puffer wird *nicht* verändert.

Beispiel

Wenn Sie das Tagesdatum in Ihren Text einfügen wollen, gehen Sie so vor:

Schreibmarke positionieren. (Das Datum wird in die folgende Zeile gesetzt.)

:r!date oder **:r!sie datum**

Die Datum-Ausgabe erfolgt am Zeilenanfang. (Das zweitgenannten Kommando bringt das Datum in deutscher Schreibweise.) Den Text müssen Sie anschließend zurechtrücken.

5.2.7 Wie blättern Sie in einem Dokument?

Angenommen Sie wollen eine bestehende größere Datei anschauen oder ändern, so sehen Sie nach dem Aufruf des *vi* auf 23 Bildschirmzeilen den Anfang der Datei. Interessiert Sie eine Textstelle, die noch nicht sichtbar ist, so müssen Sie das Fenster verschieben.

Ein Links- oder Rechts-Verschieben gibt es im *vi* nicht, weil längere Datensätze immer vollständig angezeigt werden, falls erforderlich auf mehreren Bildschirmzeilen.

Es gibt also nur ein auf die Datei bezogenes nach unten oder nach oben Verschieben des Fensters.

Um (fast) allen Wünschen gerecht zu werden, kennt der *vi* dazu etwa 20 Möglichkeiten. Lassen Sie sich davon vorerst nicht beeindrucken, sondern benutzen Sie anfangs nur einige einfache Positionierkommandos.

Fenster positionieren

Der *vi* gestattet Ihnen in großen oder kleinen Schritten in Ihrem Dokument zu blättern, um

ganze Fenster

halbe Fenster

Zeilen

Bei einer Textbearbeitung will man sich den zu bearbeitenden Textteil zurechtrücken können. Interessant sind dabei

angezeigter Ausschnitt der Datei

was steht am Bildanfang

was steht am Bildende

was steht in Bildmitte

Dazu sind hier einige *vi*-Kommando aufgelistet:

CTRL d	(down)	halben Bildschirm vorwärts
CTRL u	(up)	halben Bildschirm zurück
CTRL f	(forward)	einen Bildschirm vorwärts
CTRL b	(backward)	einen Bildschirm zurück
CTRL e		eine Zeile vorwärts

<code>CTRL y</code>	eine Zeile zurück	
<code>ESC z</code> <code>↵</code>	aktuelle Zeile an Bildanfang	☾
<code>ESC z.</code>	aktuelle Zeile in Bildmitte	
<code>ESC z-</code>	aktuelle Zeile an Bildende	
<code>ESC 1G</code> (go)	Dateianfang (ab Zeile 1)	
<code>ESC nG</code>	Zeile Nr. n auf Bildmitte	
<code>ESC G</code>	Dateiende	
<code>%</code>	Schreibmarke springt zu einer Klammer. Enthält die aktuelle Zeile eine Klammer, so springt die Schreibmarke zu der vorhergehenden oder folgenden zugehörigen Klammer eines Klammerpaars ({ () }). Gut geeignet für Programmierung!	☾
<code>``</code>	Schreibmarke springt zur vorherigen Position zurück bei der etwas gemacht wurde.	
<code>mc</code>	aktuelle Zeile markieren z.B. mit Buchstabe c	
<code>`c</code>	Rücksprung auf mit c markierte Zeile	☾

Beispiel

Zum Vorwärtsblättern ist `CTRL d` sehr praktisch (halber Bildschirm). Sollte sich das Fenster nicht verschieben, so drücken Sie vorher `ESC`.

Sie können auch die Taste `↵` verwenden und gedrückt halten, dann *durchlaufen* die Zeilen den Bildschirm.

Ihre aktuelle Zeile wird in Bildmitte gerückt mit `ESC z.` (z und Punkt).

Dateianfang: `ESC 1G`

Dateiende: `ESC G`

Textstelle suchen

Zum Suchen einer bestimmten Textstelle im Editor-Puffer, können Sie durch Angeben des ganzen Musters oder auch nur eines Teils davon, die Schreibmarke positionieren. *vi* sucht das Muster im ganzen Editor-Puffer, nicht nur in dem am Bildschirm sichtbaren Bereich.

Suchen im Editor-Puffer:

/muster 	vorwärts
?muster 	rückwärts
/	Wiederholung, gleicher Suchbegriff, vorwärts
?	Wiederholung, gleicher Suchbegriff, rückwärts
n	Wiederholung, gleicher Suchbegriff, Richtung wie vorher
N	Wiederholung, gleicher Suchbegriff, Gegenrichtung

Suchen in einer Zeile

fc (find)	Sucht vorwärts in Zeile, Zeichen c (beliebiges Zeichen)
Fc	Sucht Zeichen rückwärts
tc	Sucht Zeichen vorwärts und setzt Schreibmarke links daneben
Tc	Sucht Zeichen rückwärts und setzt Schreibmarke rechts daneben
;	Wiederholung vorwärts
,	Wiederholung rückwärts

Der *vi* merkt sich sowohl den allgemeinen Suchbegriff als auch das Suchzeichen. Nach dem Anwenden anderer Kommandos kann man z.B. mit *n* gleich die nächste Textstelle suchen, die den Suchbegriff enthält. Unabhängig davon funktioniert auch weiterhin z.B. mit *;* das Suchen innerhalb einer Zeile.

Beispiel

Angenommen, Sie suchen das Wort Mississippi.

Sie geben ein:  /issis

Mit hoher Wahrscheinlichkeit finden Sie damit das gesuchte Wort. Die Abkürzung erspart Ihnen Arbeit.

Die Zeile mit dem gefundenen Zeichenmuster wird in Bildschirmmitte angezeigt. Die Schreibmarke steht unter dem ersten i des Musters.

Wollen Sie in einer anderen Zeile z.B. die Schreibmarke zum nächsten M bewegen, so geben Sie ein:

`[ESC] fM`

Die Schreibmarke springt zum nächsten M (falls vorhanden). Sie können auch 3fM eingeben, dann suchen Sie das dritte M in der Zeile!

3fM ist gleich mit fM fM fM oder fM ; ; . Das muß man einmal ausprobieren.

Sie können auch Klammern suchen, Abschnitte, Satzende (zwei Leerzeichen) usw.

Wo bin ich?

Wenn Sie das Fenster positioniert haben oder aus anderen Gründen wissen wollen, auf welcher Zeile die Schreibmarke genau steht, können Sie das abfragen.

<code>. =</code>	gibt Zeilennummer aus
<code>:nu</code>	gibt Zeilennummer aus und Textzeile
<code>:f</code>	gibt Meldung aus: "datei" line xxx of xxx --yy%--
<code>[CTRL] g</code>	gibt Meldung aus: "datei" line xxx of xxx --yy%--

Nebenbei sei bemerkt, daß Sie auch die Version des installierten *vi* erfahren können mit

<code>:ve</code>	Gibt in Statuszeile Version und Freigabedatum des <i>vi</i> aus

Nicht erfahren können Sie den aktuellen Modus des *vi*, aber Sie können `[ESC]` drücken und wissen dann sicher, daß Sie im "Kommandomodus" sind.

Beispiel

Sie wollen z.B. wissen, wieviele Zeilen Ihr Dokument bereits hat. Drücken Sie dazu gleichzeitig die Tasten `[CTRL]` und `g`. In der untersten Bildschirmzeile erhalten Sie dann eine Meldung, z.B.

"test" [Modified] line 98 of 98 --100%--

So sieht die Meldung aus, wenn Sie am Dateiende sind.

5.2.8 Wie geben Sie neuen Text ein?

Der *vi* nimmt neuen Text erst an, wenn er auf "Eingabemodus" geschaltet ist.

Geben Sie das *vi*-Kommando ein: a (append = anfügen)

Jetzt können Sie Ihren Text schreiben.

(Tippen Sie im Kommandomodus z.B. `[ESC]` a a ein, so wird das erste a als Kommando gewertet und das zweite a als Texteingabe, die als a am Bildschirm angezeigt wird.)

Im Eingabemodus können Sie Ihren Text schreiben wie mit einer Schreibmaschine:

- Textzeichen schreiben (die Schreibmarke zeigt nächsten Zeichenplatz)
- Zurückpositionieren der Schreibmarke mit `[X]`
- Text überschreiben
- neue Zeile mit `[J]`

Das sind die einfachsten Möglichkeiten des *vi*, wobei der Vergleich mit der Schreibmaschine weit untertrieben ist, wenn man die fast unzählbaren Funktionen des *vi* in Betracht zieht, wie:

- Verschiedene, schnelle Schreibmarkenpositionierungen
- Zwischenspeicherungen
- Text-Ersetzungen
- Voreinstellungen (mit `:set ...` Kommando)

Korrigieren bei Eingabefehlern

Einen Tippfehler, den Sie gleich erkennen, korrigieren Sie mit der Taste `[X]` (Eingabe ungültig). Ein Korrigieren mit z.B. `[←]` und anschließendem Kommando r oder R oder s oder C usw. ist weniger empfehlenswert, weil man vorher mit `[ESC]` in den Kommandomodus umschalten muß - das kann man leicht vergessen und man muß dann noch mehr korrigieren. Außerdem springt mit `[ESC]` die Schreibmarke um ein Zeichen zurück!

Bei Korrekturen muß man genau beobachten, wo die Schreibmarke steht!

(Bei anderen Arbeiten sind diese Eigenarten aber wieder hilfreich.)

Wenn Sie sich beim Korrigieren vertippt haben und Fehlbedienungen anwachsen, so geben Sie das Kommando u (undo) und die letzte Änderung wird rückgängig gemacht. Sie setzen dann neu an.

Zum Korrigieren von vorhandenem Text gibt es viele verschiedene Möglichkeiten. Prinzipiell muß zuerst die Schreibmarke richtig positioniert werden. Wie Sie schnell und gezielt die notwendige Position erreichen, zeigt Ihnen der nächste Abschnitt.

5.2.9 Schreibmarke positionieren

Zum Ändern von vorhandenem Text müssen Sie die Schreibmarke an die entsprechende Textstelle positionieren, zeilengenau oder zeichengenau. Die Grobpositionierung des Bildschirmausschnitts wurde vorher beschrieben, hier wird erklärt wie Sie innerhalb des Fensters die Schreibmarke setzen.

Mit den Positionierungs-Kommandos können Sie die Schreibmarke generell nicht über den vorhandenen Text hinaus bewegen. Wenn Sie das wollen, müssen Sie das mit einem Kommando anfügen, einfügen usw. machen.

Pfeiltasten

Mit den Pfeiltasten, rechts neben dem Buchstabenfeld der Tastatur, können Sie die Schreibmarke in entsprechender Richtung bewegen.

- ein Zeichen links
- ein Zeichen rechts
- eine Zeile nach oben
- eine Zeile nach unten

Wenn Sie vorher die Taste **[ESC]** betätigen, können Sie auch einen Wiederholungsfaktor angeben, z.B. 5 **[→]** rückt die Schreibmarke 5 Zeichen weiter nach rechts. Mit dieser Methode können Sie auch spaltengenau +1 positionieren, wenn die Schreibmarke vorher ganz links steht. - Kommando 0

Nicht verwenden können Sie im *vi* die Tasten **[↑]** und **[↓]**.

Die Taste **[↩]** hat eine ganz andere Funktion. Mit ihr springt die Schreibmarke in die letzte Bildschirmzeile, in der dann : erscheint und ein Kommando erwartet wird.

Nehmen Sie an, die Schreibmarke steht etwa in der Mitte des Bildschirms.

Nun einige Kommandos zum Positionieren der Schreibmarke.

h	ein Zeichen nach links
j	eine Zeile vor
+	eine Zeile vor, an Zeilenanfang
[↩]	eine Zeile vor, an Zeilenanfang

k	eine Zeile zurück
—	eine Zeile zurück, an Zeilenangfang
l	ein Zeichen nach rechts
Leertaste	ein Zeichen nach rechts

Bevorzugen Sie bitte die Buchstaben-Kommandos (h, j, k, l) gegenüber den Pfeiltasten. Sie müssen dann die Hand nicht so weit bewegen und laufen nicht Gefahr, ggf. ein *ex*-Kommando (:...) ungewollt mit einer Pfeiltaste auszulösen. (Könnte sofort mit Kommando u repariert werden.)

Noch einige Positionierungs-Kommandos:

H (home)	Bildschirmangfang
M (middle)	Bildschirmmitte
L (last)	Bildschirmende (Bildschirmzeile 23)
0	Bildschirmrand links
	Bildschirmrand links
^	erstes Zeichen der Zeile (ungleich Leerzeichen)
_	erstes Zeichen der Zeile (ungleich Leerzeichen)
\$	letztes Zeichen der Zeile rechts
w (word)	Wortanfang rechts (nächstes Wort)
e (end)	Wortende rechts (nächstes Wortende)
b (back)	Wortbeginn links
W (word)	Wortanfang rechts (Langwort)
E (end)	Wortende rechts (Langwort)
B (back)	Wortbeginn links (Langwort)

„Langwort“ bedeutet, daß Sonderzeichen in Zeichenketten nicht als Wortgrenzen gewertet werden, sondern nur Leerzeichen oder Zeilenende.

Beispiel

In einen vorhandenen Text soll ein Wort eingefügt werden. Wie positioniert man dann die Schreibmarke schnell an die richtige Stelle? Steht die Schreibmarke links, so können Sie mit der Pfeiltaste  zeichenweise nach rechts positionieren.

Schneller:

Sie gehen in größeren Schritten, z.B. mit 5  oder 5 und Leertaste oder wortweise mit w oder W, von Wortanfang zu Wortanfang oder um mehrere Worte mit z.B. 3W - haben Sie die gewünschte Position überlaufen, so gehen Sie mit b oder B an den Wort-Beginnt zurück. Dann

springen Sie mit e oder E an das Wort-Ende.

Geben Sie dann das Kommando a und Ihr neues Wort ein, dann ESC oder

wenn Sie mit der Schreibmarke den Wortanfang des nächsten Wortes erreicht haben, können Sie mit dem Kommando i arbeiten und Ihr einzufügendes Wort genau an der Stelle der Schreibmarke absetzen.

Sie können auch in der Zeile suchen mit fc, Fc, tc, Tc (c ist Zeichen).

Ist Ihnen der Weg vom Zeilenanfang zu weit, um an die gewünschte Stelle zu gelangen, so geben Sie ESC ein und \$, dann springt die Schreibmarke an das Zeilenende und Sie können zurückspringen mit z.B. B oder 3B.

Wollen Sie eine Zeile *trennen*, positionieren Sie die Schreibmarke und geben das Kommando a. Mit dem Drücken der Taste ↵ wird der Zeilenrest, ab der Schreibmarkenposition, in eine neue Zeile übertragen.

Mit dem Kommando J können Sie die nächste Zeile an die aktuelle Zeile anhängen, also *Zeilen verbinden*. Die Schreibmarke kann in der aktuellen Zeile irgendwo stehen. Ein Kommando 3J verbindet die aktuelle Zeile mit den *zwei* folgenden.

5.2.10 Wie bearbeiten Sie vorhandenen Text?

Zuerst suchen Sie die entsprechende Textstelle und positionieren die Schreibmarke.

Textteile löschen und zurückholen

Zum Löschen verwendet man vorwiegend die Buchstaben x und d.

Hier sind einige Kommandos zum Löschen zusammengestellt:

x	Löscht ein Zeichen an der Position der Schreibmarke
3x	Löscht drei Zeichen ab Schreibmarke
X	Löscht ein Zeichen links vor Schreibmarke
3X	Löscht drei Zeichen links vor Schreibmarke
dd	Löscht aktuelle Zeile
3dd	Löscht drei Zeilen, aktuelle und zwei weitere
dG	Löscht aktuelle Zeile und bis Dateiende
dH	Löscht aktuelle Zeile und rückwärts bis Bildschirmanfang
d2H	Löscht aktuelle Zeile ..., oben bleiben 2 Zeilen erhalten
D	Löscht Zeilenrest, ab Schreibmarke

Jede Löschung wird automatisch in einem unbenannten Puffer gespeichert. Wenn Sie eine Löschung rückgängig machen wollen, so können Sie aus diesem Puffer den gelöschten Text wieder zurückholen.

Wie? Die einfachste Methode ist das Kommando u (undo). Geben Sie dieses Kommando, so wird die letzte Änderung rückgängig gemacht. Geben Sie das Kommando U, so wird die ganze aktuelle Zeile wieder hergestellt.

Auch das Kommando p holt Ihnen den Inhalt dieses Puffers zurück. - Man kann damit auch schnell eine Zeile z.B. verdoppeln; erst löschen, dann eine Zeile zurück, p geben und nochmals p.

Haben Sie schon nach dem Löschen ein Kommandos gegeben, das den Puffer verändert hat, so ist trotzdem noch nicht alles verloren - und Sie können sich die Daten zurückholen:

Der *vi* hat 10 Puffer, in denen er immer automatisch die letzten Löschungen speichert.

Der erste Puffer wird mit jedem neuen Kommando gelöscht. Sein Inhalt wird an den zweiten Puffer weitergegeben und dessen Inhalt an den dritten usw. Wenn Sie z.B. eine Zeile gelöscht haben und bereits neuen Text geschrieben haben, ist der erste Puffer bereits leer - das Kommando `u` bringt nichts. Geben Sie aber das Kommando `"1p` (put 1. Puffer), so erhalten Sie die Daten wieder zurück. Wenn dieser Puffer nicht die erhoffte Information enthält, so geben Sie gleich anschließend das Kommando `.` (Punkt für Kommandowiederholung). *vi* gibt Ihnen dann pro Kommando Punkt den Inhalt der weiteren Puffer am Bildschirm aus. Ggf. müssen Sie dann Ihren Bildschirm wieder aufräumen. Das sollten Sie einmal ausprobieren.

Diese mit Ziffern (1 bis 9) adressierbaren Puffer können Sie selbst nicht laden, aber lesen. Zum Schreiben und Lesen stehen Ihnen im *vi* 27 Puffer zur Verfügung, die mit den Kleinbuchstaben vom Benutzer selbst adressierbar sind (vgl. Beispiele am Ende dieser *vi*-Beschreibung).

Textteile ändern und erweitern

Gleich wieder ein paar Kommandos "zum Aufwärmen".

r (replace) ersetzt aktuelles Zeichen (eines) durch nächste Eingabe

R (replace) ab aktuellem Zeichen Text überschreiben

Diese beiden Kommandos helfen Ihnen auch, wenn Sie bei Textkorrekturen mit dem Positionieren der Schreibmarke Schwierigkeiten haben sollten.

Eine weitere radikale Methode: Löschen Sie einfach die Zeile mit **dd**, wenn Sie mehrfach unbeabsichtigt Kommandozeichen in Ihren Text eingefügt haben und schreiben Sie die Zeile neu. Bei ein paar Zeichen am Zeilenanfang ist das oft einfacher als weitere hektische Korrekturversuche.

Merke: **[ESC]** nicht vergessen und auf Position der Schreibmarke achten!.

Leerzeilen einfügen

o Leerzeile einfügen nach aktueller Zeile

O Leerzeile einfügen vor aktueller Zeile

Beim Zeileneinfügen können Sie keinen Wiederholungsfaktor angeben. Mit z.B. **5o** erhalten Sie nur eine neue Leerzeile. Für mehrere Leerzeilen müssen Sie mehrfach **[ESC]** und **o** oder **[ESC]** und **O** eingeben!

Scheuen Sie sich nicht, ein paar Leerzeilen zu viel in Ihren Text einzufügen, wenn Sie auch nur wenige Textzeilen einfügen wollen. Sie haben dann genügend Platz und auch Abstand vom übrigen Text. Anschließend löschen Sie die Zeilen mit z.B. **4dd** und wenn Sie nicht genau getroffen haben, mit einzelner Zeilenlöschung.

Es ist natürlich kein Muß, Leerzeilen einzufügen. Sie können auch mit **A** oder mit **[↵]** im laufenden Text arbeiten.

Textbearbeitung

a	(append)	Text <i>nach</i> aktuellem Zeichen anfügen bzw. einfügen
A	(append)	Text am Zeilenende anfügen
i	(insert)	Text <i>genau an</i> aktuellem Zeichenplatz einfügen
I	(insert)	Text ab Zeilen- <i>Anfang</i> einfügen

Beispiel

Texteingabe:

Der vi kann viel, er hat aber auch viele Kommandos.

Textkorrektur: Am Satzende stehen zwei Punkte. Einer muß weg. Sie sind schon in der nächsten Zeile, am Zeilenanfang.

Erst Schreimarke positionieren: Vorher immer ESC drücken! Dann

k\$*x* (Zeile zurück, Satzende, Zeichen löschen) oder

-\$*x* (Zeile zurück, Satzende, Zeichen löschen) oder

bx (einen Wortbereich zurück, Zeichen löschen)

Noch ein praktisches Beispiel:

Angenommen Sie mühen sich, eine Tabelle mit Strichen aufzubauen. Helfen Sie sich dabei selbst. Sie schreiben am Zeilenanfang einige Striche. Positionieren zurück, z.B. mit ^, Löschen mit D und holen *mehrfach* mit p die Striche zurück; die Zeile ist dann gleich voll. Geben Sie dann in der nächsten Zeile spaltengenau die erforderlichen senkrechten Striche ein. Löschen Sie die Zeile, positionieren Sie eine Zeile nach oben (k) und holen Sie die gespeicherte Zeile mehrfach wieder (p). Das geht schnell.

Die Zeile mit den senkrechten Strichen können Sie auch ohne sie zu löschen in den unbenannten Puffer speichern, mit Y.

Wenn Sie also Y als vi-Kommando geben, wird die aktuelle Zeile zwischengespeichert.

Geben Sie anschließend das Kommando p, so wird die eine Zeile einkopiert.

Wenn Sie sich mehrere Zeilen merken wollen, setzen Sie vor Y eine Zahl. Z.B. 4Y speichert 4 Zeilen. Kommando p, gleich anschließend gegeben, ggf. nach einer Schreibmarkenpositionierung, bringt Ihnen dann die 4 Zeilen als Kopie.

5.2.11 Kommando-Mehrfachausführung

Bei vielen *vi*-Kommandos können Sie einen Zahlenwert als Vervielfacher vor dem Kommando angeben.

Beispiel

- 3- Schreibmarke 3 Zeilen zurückpositionieren
- 4dd Löscht aktuelle Zeile und 3 weitere Zeilen
- 4dw Löscht rechts von Schreibmarke Zeichen bis zur vierten Wortgrenze
- 5| Positioniert Schreibmarke genau auf Spalte 5 der aktuellen Zeile (oder auf Zeilenende)
- 5z. Positioniert Fenster zentriert auf fünfte Zeile

Nicht alle Kommandos des *vi* reagieren auf Vervielfachungsangaben, so z.B. *o* und *O* oder *p*.

5.2.12 Arbeiten mit mehreren Dateien

Im *vi*-Aufruf können Sie mehrere Dateien angeben. Sie arbeiten erst mit der einen, sichern sie und schalten zur nächsten.

Zur nächsten Datei schalten Sie mit: `[ESC] :n [↵]`

Zur vorherigen Datei schalten Sie mit: `[ESC] :e # [↵]`

In eine andere Datei kann man immer nur wechseln, wenn die aktuelle z.B. mit `:w ...` gesichert wurde. Wenn Sie den Inhalt des Editor-Puffers nicht mehr verwenden wollen, können Sie mit `!` arbeiten, d.h. Kommando-Funktion erzwingen.

Mit `:e!` wird der Editor-Puffer mit dem Inhalt der aktuellen Datei überschrieben, d.h. mit der Datei, deren Name für den *vi* aktuell ist. Sie können aber auch selbst dabei einen Namen angeben, dann gilt dieser.

Andere Datei einkopieren

Mit `:r datei` können Sie den Inhalt von *datei* in den Editor-Puffer einfügen. Die Einfügung erfolgt hinter der aktuellen Zeile des Editor-Puffers. Der Rest des Editorpuffers wird weitergeschoben.

5.2.13 Mehrere Beispiele zu vi

Folgend sind einige Anwendungsbeispiele für *vi* zusammengestellt. Teilweise sind darunter auch Anwendungen gezeigt, die im vorherigen Text noch nicht angesprochen wurden. Die Praxis zeigt aber oft, daß eine Beispielsammlung ohne Text-Ballast auch gerne zum Nachschlagen verwendet wird, wenn die Prinzipien klar sind. Einige Beispiele sind nicht nur für *vi* verwendbar, sondern auch für andere, verwandte Editoren interessant. (Siehe auch Buch SINIX Kommandos bei *ed* und *ex* und *vi*.)

Sichern

:w	Editor-Puffer sichern
:w datei	Editor-Puffer sichern in Datei
:w > > datei	Editor-Puffer anhängen an Datei
:4,17w datei	Zeilen 4 bis 17 sichern in Datei (14 Zeilen)
:. ,17w datei	akt. Zeile bis Zeile 17 sichern in Datei
:. , +17w datei	akt. Zeile und 17 weitere sichern in Datei
:. ,.17w datei	akt. Zeile und 17 weitere sichern in Datei (+ kann hierbei entfallen)
:. , +17w datei	akt. Zeile und 17 weitere sichern in Datei (. kann hierbei entfallen)
<i>Bereichsangaben, siehe bei: Suchen - Ersetzen</i>	

Löschen

dd	ganze akt. Zeile löschen
d0	in akt. Zeile alle Zeichen links vor Schreibmarke löschen
D	in akt. Zeile alle Zeichen rechts ab Schreibmarke löschen
dW	akt. Zeichen bis Lang-Wort-Ende (z.B. Leerz.) löschen
db	Wortbeginn bis eine Stelle vor dem akt. Zeichen löschen

5dd	akt. Zeile und vier weitere löschen (5 Zeilen)
-4dd	eine Zeile zurück!, dann 4 Zeilen löschen
:-4d	vier Zeilen zurück, dann 1 Zeile löschen
:%d	löscht alles (ganzen Editor-Puffer)
:d^	löscht ganze akt. Zeile (wie dd)
d^	löscht links neben Schreibmarke bis Zeilenanfang (wie d0)
mc	Markierung setzen, z.B. mit Buchstabe c (alle Buchstaben möglich Keine Überschneidung mit "-Puffer a bis z)
d'c	löscht ganze Zeilen, akt. Zeile bis inkl. markierte Zeile (' bezeichnet Anfang der Zeile mit Markierung)
d`c	löscht genau Zeichenkette > ab Schreibmarke bis < Marke !! auch über mehrere Zeilen (` bezeichnet Zeichenplatz, wo die Markierung steht)
ma	Markierung setzen, z.B. mit Buchstabe a
mb	Markierung setzen, z.B. mit Buchstabe b
:`a,'bd	löscht ganze Zeilen, a-markierte Zeile bis b-markierte

Kopieren

Y	akt. Zeile merken
yy	akt. Zeile merken
5Y	akt. Zeile und 4 weitere merken
Yp	akt. Zeile verdoppeln
yyp	akt. Zeile verdoppeln
:10,13t 20	Zeilen 10 bis 13 ab neue Zeile 21 einfügen
:.t.	akt. Zeile verdoppeln
:t.	akt. Zeile verdoppeln
:-2t.	zweite Zeile vor akt. Zeile, hinter akt. Zeile kopieren
:r datei	Inhalt von Datei hinter akt. Zeile einkopieren
:4r datei	Inhalt von Datei hinter Zeile 4 einkopieren

Übertragen

3dd p	3 Zeilen löschen, auf Zeile positionieren gespeicherte Zeilen einfügen
"a3dd oder 3"add	3 Zeilen löschen und in Puffer a ablegen (steht auch im namenlosen Puffer bis zu nächster Änderung, Puffer a bleibt geladen) auf Zeile positionieren
"ap "aP	gespeicherte Zeilen <i>nach akt. Zeile</i> einfügen gespeicherte Zeilen <i>vor akt. Zeile</i> einfügen
"aD "ap "aP	Zeilenrest löschen und in Puffer a ablegen Puffer a <i>nach Schreibmarke in akt. Zeile</i> einfügen Puffer a <i>vor Schreibmarke in akt. Zeile</i> einfügen
"cyw	speichert Wort ab Schreibmarke in Puffer c
:4,18 m 21	Zeilen 4 bis 18 hinter Zeile 21 setzen (move)
:m. +	Vertauscht zwei Zeilen
:m. +	Vertauscht zwei Zeilen

Suchen

/xyz	im ganzen Editor-Puffer xyz suchen, vorwärts
/xyz /	im ganzen Editor-Puffer xyz suchen, mit 2 Leerzeichen
/^\$/	Leerzeilen suchen
/^\./	Punkt am Zeilenanfang suchen <i>Sonderzeichen</i> sind zu entwerten mit \
	^ . \$ * \ /
/\.\$/	Punkt am Zeilenende suchen
/\(...,\)	suche Zeichenkette: 2 Zeichen, auf die ein Komma folgt
/^\(...,\)	suche Komma in Spalte 5
/a.b	suche axc abc a = c usw. (. ersetzt ein Zeichen)
/(%	suche nächstes Zeichen: öffnende Klammer suche nächste <i>zugehörige</i> schließende Klammer

Suchen - Ersetzen

<code>:s/muster/muster/</code>	in akt. Zeile ersetzen 1. muster durch 2. muster
<code>:s/alt/neu/</code>	in akt. Zeile ersetzen, erstes alt durch neu
<code>:/alt/s//neu/</code>	erst alt suchen, dann ersetzen alt durch neu
<code>:/alt/s/alt/neu/</code>	erst alt suchen, dann ersetzen alt durch neu
<code>:s/alt/neu/g</code>	in akt. Zeile ersetzen, alle alt durch neu (generell)
<code>:1,\$s/alt/neu/g</code>	im ganzen Editor-Puffer ersetzen, alle alt durch neu
<code>:%s/alt/neu/g</code>	im ganzen Editor-Puffer ersetzen, alle alt durch neu
<code>:%s/^ / /g</code>	im ganzen Editor-Puffer vor jede Zeile 2 neue Leerzeichen
<code>:%s/^ //g</code>	im ganzen Editor-Puffer vorne 2 Zeichen löschen
<code>&</code>	Letztes :s-Kommando wiederholen ! -aber:
<code>:s/alt/(&)/</code>	in akt. Zeile ersetzen, erst-alt durch (alt)
<code>:s/alt/&neu/</code>	in akt. Zeile ersetzen, erst-alt durch altneu
<code>:%s/alt/&neu/g</code>	im ganzen Editor-Puffer ersetzen, alle alt durch altneu
<code>:s/\$./.</code>	in akt. Zeile, am Zeilenende einen Punkt setzen

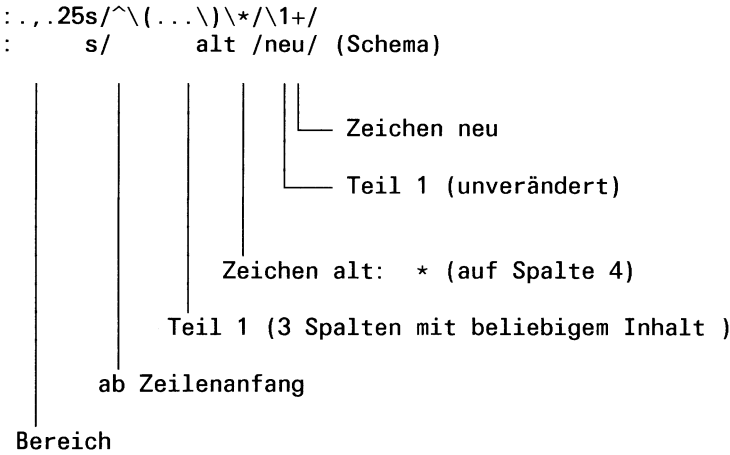
Bereich vor Kommando:

%	ganzer Editor-Puffer
.	aktuelle Zeile
n	Nummer der Zeile
.n	n-te Zeile nach akt. Zeile
.-n	n-te Zeile vor akt. Zeile
1	Dateianfang
\$	Dateiende
/muster/	Zeile, die Muster enthält (siehe auch Reguläre Ausdrücke im Anhang)
von,bis	Komma als Trennzeichen

Sonderzeichen für Muster

^	Zeilenanfang (nicht Leerzeichen) /^.../
\$	Zeilenende /...\$/
.	beliebiges Zeichen, eines
..	beliebige Zeichen, zwei ...usw.
*	beliebig häufige Wiederholung des Zeichens davor
.*	Zeichenkette
xx*	ein oder mehrere Zeichen x
\	Entwerten des Zeichens danach
	erforderlich für ^ . \$ * / \
\<wort\>	

Im Bereich akt. Zeile und 25 weitere Zeilen, soll, wenn ein * in Spalte 4 steht, dieser in + umgewandelt werden:



Folgende Vorgabe geändert werden:

```
1-3 abc      in:  
1-x3 abc
```

Begriff teilen und ändern:

```
:s/^\\(\\.\\.\\.\\)\\([1-9]\\)/\\1x\\2/      Schema: /  \\(\\.\\.\\.\\)  \\(\\.\\.\\.\\)  /  \\1  \\2  /  
      1      2      1  2
```

Nach dem ersten Teil (ein beliebiges Zeichen gefolgt von -) steht eine Ziffer (hier 3). Der erste Teil (Teil 1) wird erweitert um das Zeichen x, das dann nach dem Bindestrich steht. Der zweite Teil wird wieder angehängt. Die Zeichen abc werden einfach automatisch weitergeschoben.

Mit dieser Methode können Sie auch Teile umstellen, z.B. /\2\1/ .

Kommando speichern

Kommando alleine in eine Zeile schreiben:

Wenn Sie speichern wollen: erst v eingeben dann

Wenn Sie speichern wollen: erst v eingeben dann

zeigt sich als [
 zeigt sich als M

[1G4dd kann z.B. vier Zeilen am Dateianfang löschen

Speichern Sie diese Zeichenfolge mit "ayy und löschen dann mit dd.

Kommando starten

Mit @a wird die gespeichert Zeichenfolge als Kommando *gestartet*.

Mit @@ wird die gespeichert Zeichenfolge als Kommando *wiederholt*.

Damit haben Sie viele, viele Möglichkeiten, bei häufigen, gleichartigen Hantierungen, sich die Sache zu automatisieren. Fangen Sie einfach an und probieren Sie eigene Konstruktionen aus.

SINIX-Kommando

!Gsort sortiert ganzen Editor-Puffer

:1,5!sort sortiert Zeilen 1 bis 5

:.5!sort sortiert akt. sechs Zeilen

5.2.14 Voreinstellungen durch vi-Optionen

Auch im *vi* lassen sich die *ex*-Variablen verwenden, mit denen Sie Editor-Aktionen steuern können. Mit `:set all` oder `:se all` können Sie sich alle am Bildschirm auflisten lassen.

Für die Variablen gibt es Standard-Einstellungen. Diese Voreinstellungen lassen sich während der *vi*-Sitzung mit dem Kommando `:set` verändern, z.B. `:se nu`. Die Variablen-Einstellungen können auch beim Aufruf des Editors wirksam werden, indem man den entsprechenden `set`-Kommando-Wert in diese Datei schreibt: *.exrc*. Die Einstellung bleibt dann über die Sitzung hinaus erhalten:

```
cat > .exrc
```

```
se nu
```

```
END
```

Man kann auch mit der Umgebungsvariablen `EXINIT` arbeiten:

```
EXINIT="se nu" ; export EXINIT
```

Hier sind einige Werte für `EX`-Variable angegeben.

(Weitere finden Sie im Buch *SINIX Kommandos* bei `ex`-Kommando.)

autoindent, ai (Voreinstellung *noai*)

Im Eingabemodus wird jede Zeile entsprechend der vorhergehende Zeile eingerückt (es werden Leer- und Tabulatorzeichen eingefügt). Der Beginn des eingerückten Textes hängt davon ab, welche Zeile danach angehängt, davor eingefügt oder zuerst geändert wird. Sind Tabulatorzeichen eingefügt, so werden die darauffolgenden Zeilen ebenfalls entsprechend eingerückt. Soll der Text weniger eingerückt sein, so ist `CTRL` d ein- oder mehrmals zu geben. Jedesmal wird der Zeilenanfang um *shiftwidth* Positionen nach links verschoben. Das erste Mal wird die Einrückung für die aktuelle Zeile aufgehoben; mit 0, gefolgt von `CTRL` d, werden alle Einrückungen aufgehoben.

autowrite, aw (*noaw*)

Der Editor-Puffer wird automatisch in die bekannte Datei gesichert bei Dateiwechsel oder dem Verlassen des *vi*.

ignorecase, ic (*noic*)

Beim Suchen soll kein Unterschied zwischen Groß- und Kleinbuchstaben gemacht werden.

list (*nolist*)

Tabulatorzeichen werden als `^I` und Zeilenende mit `$` angezeigt.

number, nu (nonu)

Die Zeilen werden mit vorangestellten Zeilennummern ausgegeben.

shiftwidth, sw (=8)

Der bei dieser Variablen angegebene Wert steht für die Anzahl von Positionen, um die der Text bei aktivem *autoindent* und den Kommandos < und > verschoben werden soll.

tabstop, ts (=8)

Der bei dieser Variablen angegebene Wert steht für die Position der Tabulatorstopps in der Eingabedatei.

window (=23)

Die Anzahl der Zeilen in einem Textfenster.

wrapscan, ws (ws)

Ist diese Variable gesetzt, so wird das Suchen (mit / / oder ? ?), wenn das Ende bzw. der Anfang des Puffers erreicht ist, am Anfang bzw. Ende fortgesetzt; andernfalls wird das Suchen am Anfang bzw. am Ende der Datei beendet.

writeany, wa (nowa)

Macht das Ändern von beliebigen Dateien möglich (sofern die Zugriffsrechte entsprechend geregelt sind). Dabei wird nicht überprüft, ob die bei einem *write* Kommando angegebene Datei bereits vorhanden ist.

Hinweis

Bei dem Kommando *undo* gehen alle Markierungen für Zeilen, die geändert und dann wieder zurückgeholt wurden, verloren.

Ist in den Eingabedateien das ASCII-Zeichen NUL enthalten, so wird es gelöscht; in den Ausgabedateien kann es daher nicht enthalten sein.

Bei einigen Variablen ist die Wiederaufnahme einer Editorsitzung mit dem Schalter -r nur möglich, wenn beim erneuten Starten des Systems eine bestimmte, systemabhängige Vorgehensweise eingehalten wird.

—

—

—

—

6 Programme

6.1 Das Programm SCCS

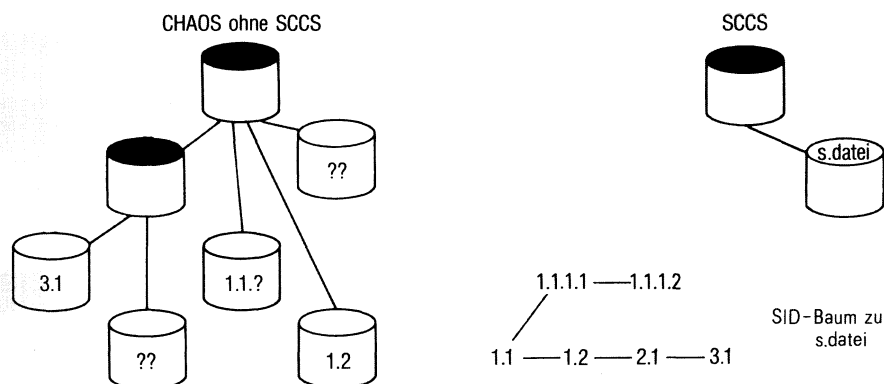


Bild 6-1 SCCS-Schema

Was ist SCCS?

SCCS (Source Code Control System) ist ein System, mit dem Sie verschiedene Versionen von Quellcode- oder Textdateien verwalten können. SCCS besteht aus einer Reihe von Shell-Kommandos; diese Kommandos helfen Ihnen, Änderungen, die Sie an Versionen vornehmen, zu kontrollieren und zu protokollieren. Das ist wichtig, wenn Sie Programme oder Dokumentation häufig ändern müssen, die ursprünglichen Versionen aber behalten wollen.

SCCS speichert nur die erste Version vollständig. Wenn Sie eine Version verändern, speichert SCCS nur die Änderungen. Dadurch sparen Sie Speicherplatz. Andererseits muß SCCS immer alle Vorgängerversionen durchlaufen, wenn Sie auf eine bestimmte Version zugreifen wollen. Das kostet Zeit.

Alle Versionen sind in einer einzigen Datei, der sogenannten s-Datei, enthalten. Jede Version bzw. jede Änderung erhält eine Nummer. Mit dieser Nummer können Sie leicht auf eine bestimmte Version zugreifen. Bei jedem Zugriff überprüft SCCS die Zugriffsrechte. Die Zugriffsrechte können Sie nach Belieben erteilen oder entziehen.

Wie arbeiten Sie mit SCCS?

An einem Beispiel soll gezeigt werden, wie Sie

- die erste Version erstellen,
- eine weitere Version erstellen,
- eine Version nur zum Lesen ausgeben lassen,
- Informationen zu einer Version abfragen.

Außerdem werden die Begriffe "Identifikations-Schlüsselwort" und "MR-Nummer" erläutert.

Die Datei *zoo* habe den Inhalt

```
vogel
penguin
hängebauchschwein
```

Mit dem Kommando

```
$ admin -izoo s.tiere
```

übergeben Sie die Datei *zoo* dem SCCS. *admin* legt eine sogenannte s-Datei an, die alle späteren Versionen der übergebenen Datei enthalten wird. Die Namen von s-Dateien müssen immer mit *s.* beginnen. In unserem Fall wird die s-Datei *s.tiere* erstellt.

admin meldet:

```
No id keywords (cm7)
```

Die Meldung besagt, daß in der Datei *zoo*, die ins SCCS eingegliedert wird, keine "id keywords" stehen. "id keywords" sind **Identifikations-Schlüsselwörter**, die Sie irgendwo in den Text einer Version schreiben können. Sie erleichtern das Identifizieren einer Version, z.B. welche Nummer sie hat und wann sie erstellt worden ist. Identifikations-Schlüsselwörter haben das Format

```
%großbuchstabe%
```

Je nachdem, welcher Buchstabe *großbuchstabe* ist, wird ein Text (z.B. das Datum) an die Stelle von *großbuchstabe* geschrieben, wenn *get* die Version zum Lesen aus dem SCCS holt (siehe unten).

Beim Kommando *get* (siehe SINIX Kommandos) finden Sie eine Tabelle der Identifikations-Schlüsselwörter.

Wenn Sie eine Meldung, die SCCS ausgibt, nicht verstehen, dann erhalten Sie mit *help* weitere (englischsprachige) Informationen, Dazu müssen Sie das Codewort, das hinter der Meldung in runden Klammern steht, mit angeben, z.B.

\$ help cm7

Hinweis

Nachdem Sie die s-Datei *s.tiere* erstellt haben, müssen Sie die Originaldatei *zoo* entfernen. Alle Änderungen dürfen nur noch über SCCS-Kommandos stattfinden.

Wenn Sie den Text der s-Datei ändern, d.h. eine zweite Version erstellen wollen, müssen Sie eingeben:

```
$ get -e s.tiere
1.1
new delta 1.2
3 lines
$
```

get holt die jüngste Version, in unserem Fall 1.1, aus dem SCCS, erstellt im aktuellen Dateiverzeichnis die sog. g-Datei *tiere* und schreibt den Text der Version 1.1 in die g-Datei. Der Name der g-Datei ist derselbe wie der Name der s-Datei ohne *s.* am Anfang.

Da Sie bei *get* den Schalter *-e* angegeben haben, können Sie die g-Datei nicht nur lesen, sondern den Text mit einem beliebigen Editor auch verändern oder löschen.

Die neue Version ("new delta") wird die SID-Nummer 1.2 haben. "SID-Nummer" ist die Abkürzung für "SCCS Identification Number". Die erste Komponente der SID-Nummer ist die Releasenummer (hier: 1), die zweite die Levelnummer (hier: 2). Wenn Sie bei *get -e* nichts weiter angeben, dann bleibt die Releasenummer gleich, und die Levelnummer erhöht sich um 1. Dementsprechend entsteht in unserem Beispiel aus Version 1.1 die Version 1.2.

Zusätzlich errichtet *get* eine sogenannte p-Datei, die Informationen über die SCCS-Datei enthält. Die Informationen betreffen den Zugriffsschutz, die spätere Wiedereingliederung der Version und sonstige Angaben zur Verwaltung der Version. Der Name der p-Datei ist derselbe wie der Name der s-Datei, nur mit *p.* statt *s.* am Anfang.

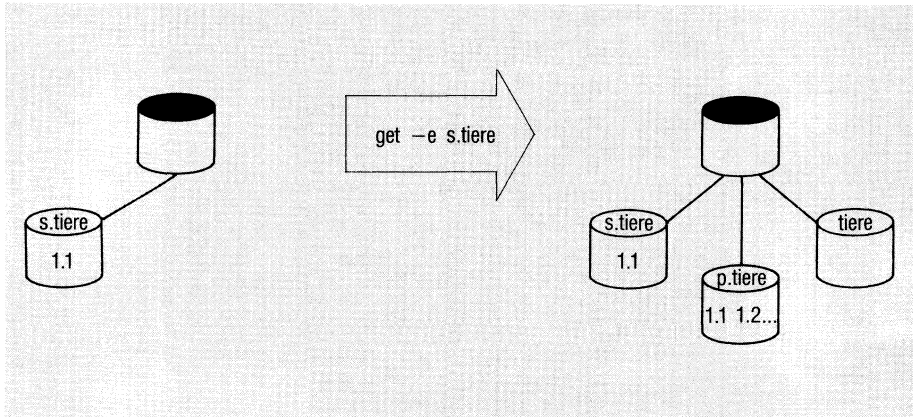


Bild 6-2 Kommando *get*

Fügen Sie nun am Anfang der g-Datei *tiere* noch zwei Zeilen ein. Verwenden Sie dazu einen Editor Ihrer Wahl.

Diese Version hat die SID-Nummer %I%
rhinozeros

Mit *delta* gliedern Sie die neue Version ins SCCS ein:

```
$ delta s.tiere
comments?      noch ein tiername, von gudrun, 27.9.87
```

Zuerst fragt *delta* nach einem Kommentar. Der Kommentar kann ein beliebiger Text oder leer sein. Sie geben den Kommentar interaktiv ein. Dann informiert sich *delta* in der p-Datei, welche Version der Datei *s.tiere* Sie geändert haben. *delta* holt sich diese Version und vergleicht sie mit der neuen Version.

Aus der p-Datei erfährt *delta* auch, welche SID-Nummer die neue Version haben soll. Die neue Versionsnummer und die Unterschiede, die die neue Version gegenüber der Vorgängerversion hat, speichert *delta* in der s-Datei ab.

delta gibt auf der Standard-Ausgabe die neue Versionsnummer und die Anzahl der Zeilen aus, die Sie eingefügt, gelöscht bzw. nicht verändert haben:

```
1.2
2 inserted
0 deleted
3 unchanged
$
```

delta löscht die g-Datei *tiere*. Außerdem löscht *delta* in der p-Datei den Eintrag über die Version, die Sie gerade erstellt haben. Wenn (wie in unserem Fall) keine Einträge mehr in der p-Datei stehen, wird auch die p-Datei gelöscht.

Holen Sie jetzt die neue Version zum Lesen aus dem SCCS:

```
$ get s.tiere
1.2
5 lines
$
```

get holt die jüngste Version, in unserem Fall 1.2, aus dem SCCS und erstellt die g-Datei *tiere*. Anders als vorher, wo Sie den Schalter *-e* angegeben haben, steht der Text nur zum Lesen in der g-Datei; er kann nicht verändert werden. Die g-Datei *tiere* hat folgenden Inhalt:

```
Diese Version hat die SID-Nummer 1.2
rhinoceros
vogel
penguin
hängebauchschwein
```

Das Identifikations-Schlüsselwort *%I%* wurde durch seinen Wert, nämlich 1.2, ersetzt.

Bevor Sie mit *get* weitere Versionen holen, müssen Sie die g-Datei aus dem aktuellen Dateiverzeichnis entfernen.

Wollen Sie Release 2 einführen, d.h. eine Version mit Releasenummer 2 erstellen, dann geben Sie statt *get -e s.tiere* die folgende Kommandozeile ein:

```
$ get -e -r2 s.tiere
1.2
new delta 2.1
5 lines
$
```

get holt die Version 1.2 zum Ändern. Die neue Version wird die SID-Nummer 2.1 haben.

Außer einem Kommentar können Sie einer Version eine **MR-Liste**, d.h. eine Liste von **MR-Nummern** mitgeben (Modification Request). Eine MR-Nummer ist eine Zahl (oder auch eine beliebige ASCII-Zeichenreihe ohne Leer- und Tabulatorzeichen), die stellvertretend steht für

- einen Grund, der angibt, warum Sie die s-Datei einrichten (bei MR-Nummern für die erste Version)
- einen Änderungswunsch oder/und
- einen Änderungsgrund

Welche Bedeutung die einzelnen MR-Nummern haben sollen, legen Sie selbst fest. Die MR-Nummern trennen Sie mit einem Leerzeichen.

Wenn Sie der neuen Version 2.1 MR-Nummern mitgeben wollen, dann geben Sie z.B. ein:

```
$ admin -fv s.tiere
$ delta s.tiere
MRs? 1 3
comments? nichts geändert!
2.1
0 inserted
0 deleted
5 unchanged
$
```

Haben Sie das Kommando *admin -fv s.tiere* eingegeben, dann fragt *delta* von nun an immer nach MR-Nummern; diese können Sie dann interaktiv eingeben. Die MR-Nummern 1 und 3 stehen z.B. für zwei Gründe, warum Sie die Version 1.2 erstellt haben.

Das Kommando *prs* liefert Ihnen Informationen über eine oder mehrere Versionen. Geben Sie z.B.

`$ prs -r1.2 s.tiere`

ein, dann erhalten Sie Informationen über die Version 1.2. *prs* gibt unter anderem aus:

- die SID-Nummer der Version,
- den Zeitpunkt, zu dem die Version erstellt worden ist,
- die Kennung des Benutzers, der die Version erstellt hat,
- die MR-Liste,
- den Kommentar.

In der folgenden Tabelle sind die Arbeitsschritte, die in diesem Beispiel behandelt wurden, noch einmal zusammengestellt.

Tätigkeit	Kommando	Dateityp	Dateien im Beispiel
Erste Version erstellen	admin	s-Datei .	s.tiere
Weitere Version erstellen	get -e delta	s-Datei g-Datei	s.tiere tiere
Version nur lesen	get	s-Datei g-Datei	s.tiere tiere
Informationen zu einer Version abfragen	prs	s-Datei	s.tiere

6.1.1 Die Dateien des SCCS

Die s-Datei

Wenn Sie eine Datei dem SCCS übergeben, wird eine **s-Datei** angelegt. Der Name einer s-Datei muß immer mit *s.* beginnen. Wenn von einer **SCCS-Datei** gesprochen wird, ist im allgemeinen immer eine s-Datei gemeint.

Die s-Datei hat die Schutzbiteinstellung `-r---r---r---`. Sie gehört dem effektiven Benutzer.

Die Originalversion wird in ihrer Gesamtheit in der s-Datei abgespeichert. Für die nachfolgenden Versionen wird jeweils nur der Unterschied zur Vorgängerversion abgespeichert. In der s-Datei stehen sämtliche Versionen - die Originalversion und alle Änderungen.

Mit **Delta** bezeichnet man den Unterschied zwischen einer Version und ihrer Vorgängerversion; meistens aber ist "Delta" nur eine andere Bezeichnung für "Version".

Jede Version hat eine Nummer, die sogenannte **SID-Nummer** ("SCCS Identification Number"). Der Benutzer bezeichnet eine Version normalerweise mit der SID-Nummer. Welche SID-Nummer eine Version bekommt, ist im Abschnitt "Die SID-Nummer und der SID-Baum" beschrieben.

Zusätzlich zur SID-Nummer bekommt jede Version eine **laufende Nummer**. Die laufenden Nummern geben an, in welcher zeitlichen Reihenfolge die Versionen erstellt wurden. Die erste Version erhält die laufende Nummer 1, die weiteren Versionen werden fortlaufend nummeriert.

Aufbau einer s-Datei

Eine s-Datei enthält ASCII-Text. Der Text ist in 6 Teile gegliedert:

- **Prüfsumme**
Die erste Zeile der s-Datei enthält eine Art "Summe" aller Zeichen, die in den restlichen Zeilen der s-Datei stehen. Die Summe dient der Konsistenzprüfung.
- **Delta-Tabelle**
Die Tabelle enthält Informationen über jedes Delta, nämlich
 - die Anzahl der Zeilen, die eingefügt, gelöscht bzw. nicht verändert wurden,

- den Typ (gibt an, ob das Delta gelöscht wurde; siehe *rmDEL*),
 - die SID-Nummer,
 - Tag und Uhrzeit der Erstellung,
 - die Benutzerkennung des Erstellers,
 - die laufende Nummer des Deltas und des Vorgängerdeltas,
 - die MR-Nummern (siehe *admin*, Schalter *-m* und *-fv*, und *delta -m*),
 - den Kommentar (siehe *admin -y* und *delta -y*).
- **Benutzerliste**
Die Benutzerliste ist eine Liste mit login-Namen oder/und Gruppennummern der Benutzer, die die SCCS-Datei ändern und Deltas erstellen oder entfernen dürfen (siehe *admin*, Schalter *-a* und *-e*).
 - **Parameter**
Sie können verschiedene Parameter setzen und mit bestimmten Werten belegen (siehe *admin*, Schalter *-f* und *-d*). Die Parameter beeinflussen die Wirkungsweise einzelner SCCS-Kommandos.
 - **Beschreibender Text**
Hier können Sie eine ausführliche Beschreibung des eigentlichen Textes geben, der in der SCCS-Datei steht (siehe *admin -t*).
 - **Verwalteter Text**
Hier steht der Text, den SCCS in der s-Datei verwaltet. Der Text besteht aus der Ursprungsversion und sämtlichen Änderungen, die Sie bei jeder Version vorgenommen haben. Zwischen den Textzeilen können Kontrollzeilen des SCCS stehen.

Die x-Datei

Alle SCCS-Kommandos, die eine SCCS-Datei verändern, erstellen eine temporäre Kopie der SCCS-Datei, die **x-Datei**, und bearbeiten die x-Datei. Die x-Datei hat denselben Namen wie die s-Datei, nur mit *x*. statt *s*. am Anfang. Wenn das Kommando fehlerfrei ausgeführt ist, wird die x-Datei zur s-Datei, und die alte s-Datei wird gelöscht. Wenn beim Ausführen des Kommandos Fehler auftreten, wird höchstens die x-Datei, nicht aber die s-Datei zerstört.

Die x-Datei steht im selben Dateiverzeichnis wie die s-Datei und hat auch dieselbe Schutzbiteinstellung. Wenn die s-Datei noch nicht existiert (z.B. bei *admin*), ist die Schutzbiteinstellung *-r--r--r--*. Der effektive Benutzer ist der Eigentümer der x-Datei.

Der Benutzer kann die Existenz der x-Datei normalerweise ignorieren. Wenn das System zusammenbricht oder sonst etwas Unvorhergesehenes passiert, kann die x-Datei eventuell helfen, den Zustand vor dem Zusammenbruch zu rekonstruieren.

Die z-Datei

Alle SCCS-Kommandos, die eine SCCS-Datei verändern, erstellen eine **z-Datei**. Sie hat denselben Namen wie die s-Datei, nur mit *z.* statt *s.* am Anfang. Die z-Datei existiert nur während der Ausführung des Kommandos und blockiert weitere Zugriffe auf die SCCS-Datei, solange das Kommando abgearbeitet wird.

In der z-Datei steht die Prozeßnummer des Kommandos, das die z-Datei erstellt hat. Wenn eine z-Datei existiert, erkennen andere SCCS-Kommandos, daß die entsprechende SCCS-Datei blockiert ist.

Die z-Datei hat die Schutzbiteinstellung `-r--r--r--`. Sie steht im selben Dateiverzeichnis wie die s-Datei. Der effektive Benutzer ist der Eigentümer.

Der Benutzer kann die Existenz der z-Datei normalerweise ignorieren. Wenn das System zusammenbricht oder sonst etwas Unvorhergesehenes passiert, kann die z-Datei eventuell helfen, den Zustand vor dem Zusammenbruch zu rekonstruieren.

Die g-Datei

Das Kommando *get* erstellt zu einer s-Datei die **g-Datei** (Ausnahme: Schalter *-g* und *-p*). Sie hat denselben Namen wie die s-Datei, aber ohne *s.* am Anfang. In der g-Datei steht der Text, den *get* aus dem SCCS geholt hat. Genau gesagt, holt *get* den eigentlichen Text der Anfangsversion der SCCS-Datei in die g-Datei. Darauf baut *get* solange Nachfolgerdeltas auf, bis die gewünschte Version in der g-Datei steht.

Die g-Datei steht im aktuellen Dateiverzeichnis und gehört dem realen Benutzer. Sie hat die Schutzbiteinstellung

`-rw-r--r--` wenn Sie bei *get* den Schalter *-e* oder *-k* angegeben haben. Sie können die g-Datei verändern und dürfen neue Deltas erstellen.

`-r--r--r--` sonst. Sie können die g-Datei lesen und z.B. übersetzen, dürfen sie aber nicht verändern.

Die p-Datei

Das Kommando *get -e* erstellt zu einer s-Datei die **p-Datei**. Die p-Datei hat denselben Namen wie die s-Datei, nur mit *p.* statt *s.* am Anfang. In der p-Datei steht Information über die Version, die *get* geholt hat, und über die neue Version, die mit *delta* erstellt werden soll. *delta* braucht diese Information, um die neue Version in das SCCS einzugliedern.

Die Informationen in der p-Datei dienen außerdem dazu zu verhindern, daß dieselbe Version ein zweites Mal mit *get -e* zum Editieren geholt wird; dieselbe Version dürfen Sie nur dann noch einmal mit *get -e* holen, wenn

- dem ersten *get* ein entsprechendes *delta* oder *unget* gefolgt ist oder
- der Parameter *j* gesetzt ist (vgl. *admin -ff*).

Die p-Datei hat die Schutzbiteinstellung *-rw-r--r--*. Sie steht im Dateiverzeichnis der s-Datei und gehört dem effektiven Benutzer.

Beim Kommando *get*, Schalter *-e*, können Sie Genaueres über die p-Datei nachlesen.

Die d-Datei

Das Kommando *delta* erstellt zu einer s-Datei die **d-Datei**. Die d-Datei hat den selben Namen wie die s-Datei, nur mit *d.* statt *s.* am Anfang. In die d-Datei kopiert sich *delta* die Vorgängerversion, auf der die neue Version aufbaut. Die Vorgängerversion ist die Version, die *get* in die g-Datei geschrieben hat. *delta* vergleicht die d-Datei mit der g-Datei, die Sie mit einem Editor verändert haben, und stellt so die Unterschiede zwischen der neuen Version und der Vorgängerversion fest.

Die l-Datei

Das Kommando *get* erstellt zu einer s-Datei eine **l-Datei**, wenn Sie den Schalter *-l* angeben. Die l-Datei hat denselben Namen wie die s-Datei, nur mit *l.* statt *s.* am Anfang. Die l-Datei enthält eine Tabelle der Deltas, die dazu beigetragen haben, den Text in der g-Datei zu erstellen.

Die l-Datei hat die Schutzbiteinstellung *-r--r--r--*. Sie steht im aktuellen Dateiverzeichnis und gehört dem realen Benutzer.

Beim Kommando *get*, Schalter *-l*, können Sie Genaueres über die l-Datei nachlesen.

Zugriffsschutz für SCCS-Dateien

- SCCS stellt einige Schalter und Parameter bei *admin* zur Verfügung, die den Zugriff auf SCCS-Dateien schützen:
 - a, -k Bei diesen Schaltern können Sie eine Liste der Benutzer angeben, die Deltas erstellen bzw. nicht erstellen dürfen.
 - fc, -ff Mit diesen Parametern können Sie den Bereich der Releasenummern festlegen. Sie dürfen dann nur Versionen mit Releasenummern aus diesem Bereich erstellen.
 - fl Sie können Releasenummern sperren. Versionen mit gesperrten Releasenummern dürfen nicht mehr verändert werden.
- *admin* gibt neuen SCCS-Dateien die Schutzbit-einstellung *-r--r--r--*. Damit ist sichergestellt, daß die SCCS-Datei nur mit SCCS-Kommandos verändert werden kann. Sie sollten deshalb mit *chmod* keine Schreibberechtigung für die SCCS-Datei erteilen.
- Dateiverzeichnisse, die SCCS-Dateien enthalten, sollten die Schutzbit-einstellung *drwxr-xr-x* haben, damit nur der Eigentümer Einträge ändern darf. Dateiverzeichnisse mit SCCS-Dateien sollten keine Dateien enthalten, die nicht vom SCCS erstellt worden sind.
- Namen von SCCS-Dateien müssen mit *s.* anfangen.

6.1.2 Die SID-Nummer und der SID-Baum

Die SID-Nummern können als die Knoten des **SID-Baumes** oder **SCCS-Baumes** betrachtet werden. Die Wurzel ist die erste Version. Sie hat z.B. die Nummer 1.1. Nachfolgerdeltas heißen z.B. 1.2, 1.3, 1.4, 1.5, 2.1, 2.2, 2.3. Jedes Delta ist von allen früheren Deltas abhängig. Der Baum besteht bis jetzt nur aus dem Stamm und sieht so aus:

```

1.1 ——— 1.2 ——— 1.3 ——— 1.4 ——— 1.5 ——— 2.1 ——— 2.2 ——— 2.3

```

Bild 6-3 Einfacher SID-Baum

Bei einer SID-Nummer heißt

- die erste Komponente **Releasenummer** oder **Release**
- die zweite Komponente **Levelnummer** oder **Level**

2.3 hat z.B. die Releasenummer 2 und die Levelnummer 3.

Wenn Sie eine neue Version einführen, wird normalerweise die Levelnummer um 1 hochgezählt, die Release bleibt unverändert. Wenn Sie die Release verändern wollen, müssen Sie es explizit angeben. Wenn z.B. die höchste Releasenummer eines Deltas 4 war und die neueste Version Releasenummer 7 haben soll, dann müssen Sie die SCCS-Datei holen mit

```
$ get -r7 -e s.datei
```

Oft ist es nötig, Änderungen vorzunehmen, die nicht von allen früheren Deltas abhängen.

Sie haben z.B. folgendes Programmsystem im SCCS:

Die Programme der Version 1.3 sind freigegeben und können von Kunden benutzt werden. Version 2.1 und 2.2 sind bereits entwickelt, aber noch nicht freigegeben. Sie arbeiten gerade an Version 2.3. Ein Kunde meldet einen Fehler in Version 1.3, den Sie möglichst bald beheben wollen. Sie möchten ein Delta zu Version 1.3 erstellen und für den Kunden freigeben. Dieses Delta soll nicht von Version 1.4 oder einer Version mit Releasenummer 2 abhängen. Der SID-Baum muß einen Zweig bekommen.

SID-Nummern von Zweigen bestehen aus vier Komponenten:

- der **Releasenummer (Release)**
- der **Levelnummer (Level)**
- der **Zweignummer (Branch)**
- der **Folgenummer (Sequence)**

Release- und Levelnummer sind die entsprechenden Nummern des Stammknotens, von dem aus der Zweig gebildet werden soll. Im Beispiel also 1.3. Die Zweignummer gibt an, welcher Zweig zu einem Stammknoten gebildet wird.

Der erste Zweig hat die Zweignummer 1, der nächste die Zweignummer 2 usw. Innerhalb eines Zweiges erhält jeder Knoten eine Folgenummer. Die Folgenummer richtet sich danach, an welcher Stelle der Knoten innerhalb des Zweiges liegt. Zweige können sowohl in Stammknoten als auch in Zweigknoten beginnen.

SID-Nummern bestehen also immer aus genau

- zwei Komponenten, wenn sie Stammknoten sind,
- vier Komponenten, wenn sie Zweigknoten sind.

Das Beispiel kann folgenden SID-Baum bekommen:

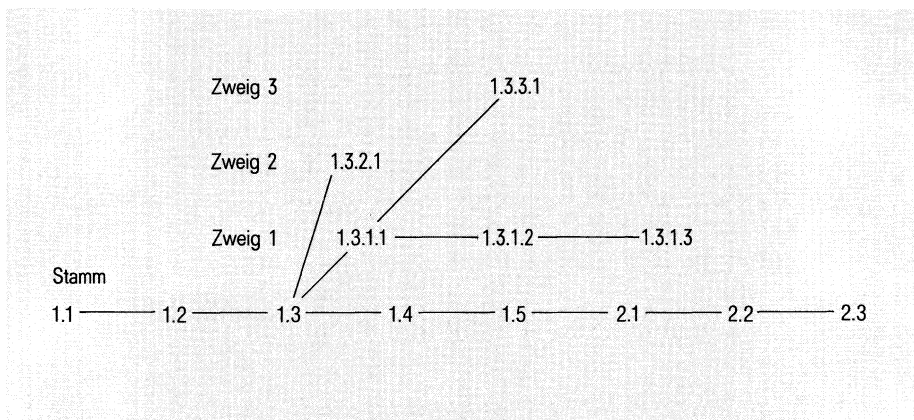


Bild 6-4 SID-Baum mit Zweigen

Der Baum gibt genau wieder, welche Vorgängerversion jede Version hat. Die Vorgängerversion ist die Version, die *get* geholt hat und die Sie geändert haben, um die neue Version zu erstellen. Version 1.3.1.2 und 1.3.3.1 haben z.B. beide den Vorgänger 1.3.1.1.

Im Stammknoten 1.3 beginnen zwei Zweige: Zweig 1 mit der Zweignummer 1 und Zweig 2 mit der Zweignummer 2. Im Zweigknoten 1.3.1.1 beginnt ein weiterer Zweig - der dritte, der von 1.3 abhängt. Er bekommt deshalb die Zweignummer 3.

Die Folgenummer wird fortlaufend innerhalb des Zweiges gezählt.

Alle drei Zweige hängen vom Stammknoten 1.3 ab. Sie haben deshalb genau wie dieser Stammknoten die Releasenummer 1 und die Levelnummer 3.

Nummern des ersten Zweiges haben also die Form 1.3.1.n, die des zweiten Zweiges 1.3.2.n, und die des dritten Zweiges haben die Form 1.3.3.n.

Aus der SID-Nummer eines Zweiges können Sie

- erkennen, von welchem Stammknoten der Zweig abhängt,
- nicht erkennen, in welchem Zweigknoten der Zweig gegebenenfalls beginnt.

Wie erstellen Sie nun den ersten Zweig? Geben Sie das folgende Kommando ein:

```
$ get -r1.3 -e s.datei
```

get holt Version 1.3 aus dem SCCS. Die neue Version wird die Nummer 1.3.1.1 haben. Haben Sie Version 1.3.1.1 mit *delta* ins SCCS eingegliedert, dann holen Sie sie wieder mit

```
$ get -r1.3.1.1 -e s.datei
```

zum Editieren. Die neue Version wird nun die Nummer 1.3.1.2 haben, u.s.w.

Nun erstellen Sie den zweiten Zweig. Dazu holen Sie wie vorher mit

```
$ get -r1.3 -e s.datei
```

die Version 1.3 aus dem SCCS. Die neue Version wird aber diesmal die Nummer 1.3.2.1 haben, da bereits ein Zweig zum Stammknoten 1.3 existiert.

Wollen Sie schließlich auf Version 1.3.1.1 den dritten Zweig aufbauen, dann geben Sie ein:

```
$ get -r1.3.1.1 -e s.datei
```

Weitere Informationen finden Sie beim Kommando *get* im Buch SINIX Kommandos.

Haben Sie mehrere Versionen in der s-Datei gespeichert, dann sagt Ihnen das Kommando *prs*, wie der zugehörige SID-Baum aussieht:

Geben Sie ein:

```
$ prs s.datei
```

prs liefert Ihnen Informationen über alle Versionen der SCCS-Datei *s.datei*. Unter anderem gibt *prs* zu jeder Version aus:

- die SID-Nummer,
- die laufende Nummer und
- die laufende Nummer der Vorgängerversion

Daraus können Sie ersehen, welchen Vorgänger jeder Knoten des SID-Baumes hat und wie demzufolge der SID-Baum aussieht.

6.1.3 Was Sie beim SCCS beachten sollten

- Sie sollten nicht zu viele Zweige einführen, um nicht den Überblick zu verlieren.
- Sie sollten SCCS-Dateien nur mit SCCS-Kommandos verwalten, verändern, lesen, vergleichen und holen.
- Falls mehrere Benutzer gleichzeitig auf eine SCCS-Datei zugreifen dürfen, ernennen Sie einen Benutzer zum Verwalter. Der Verwalter kontrolliert die Zugriffsrechte. Nur er darf *admin* aufrufen.
- Wenn Sie mehrere Dateien, die voneinander abhängig sind, mit SCCS verwalten, sollten Sie die SID-Nummern konsistent halten, d.h. Sie
 - schreiben alle s-Dateien ins gleiche Dateiverzeichnis,
 - erstellen Deltas immer von allen s-Dateien gleichzeitig, selbst wenn Sie nicht alle s-Dateien verändert haben.
- Trotz SCCS müssen Sie die s-Dateien regelmäßig sichern. Schäden an der s-Datei können schwerwiegende Folgen haben: Geht zum Beispiel eine Zwischenversion verloren, können die Folgeversionen eventuell nicht mehr rekonstruiert werden. Verlieren Sie die s-Datei, dann sind sämtliche Versionen verloren.
- Jede Version einer SCCS-Datei sollte Identifikations-Schlüsselwörter enthalten, damit sie jederzeit identifiziert werden kann (siehe *get*, Tabelle der Identifikations-Schlüsselwörter, und *what* im Buch SINIX Kommandos).

Versionen, die Sie an Kunden ausliefern wollen, holen Sie mit *get* ohne den Schalter *-e* oder *-k*. *get* ersetzt dann die Identifikations-Schlüsselwörter durch ihren Wert, und der Kunde kann jederzeit abfragen, mit welcher Version er arbeitet.

Beispiel

Sie erstellen eine SCCS-Datei *s.prog.c* für verschiedene Versionen von C-Quellcode. Sie können die Identifikations-Schlüsselwörter als Kommentar in das C-Programm einfügen. Wenn Sie aber die Schlüsselwörter z.B. folgendermaßen einfügen:

```
static char SccsId[] = "%W%      %G%";
```

dann können Sie das Programm auch dann identifizieren, wenn es kompiliert ist.

Wenn *get* eine Version zum Lesen aus dem SCCS holt, dann ersetzt es diese Zeile z.B. durch

```
static char SccsId[] = "@(#)prog.c      1.3      11/16/87";
```

Mit *what* kann die Version dann jederzeit identifiziert werden, egal, ob das Programm compiliert ist oder nicht.

- Sie sollten neue Stamm-Versionen erstellen:
 - vor einer Freigabe oder Installation,
 - bei großen Veränderungen der Texte oder Programme (z.B. bei Änderung der internen Datenstruktur),
 - nicht allzu oft (z.B. nicht wegen Fehlerbehebungen oder kleinen Veränderungen).
- Eine zentrale Shell-Prozedur kann bei großen Dateisystemen die Verwaltung der SCCS-Dateien und der SCCS-Dateiverzeichnisse übernehmen.

Fehlerdiagnose

Die Fehlermeldungen der SCCS-Kommandos bestehen i.a. aus

- dem Wort "ERROR",
- dem Namen der betroffenen s-Datei,
- der Fehlerangabe,
- einem Fehlerkennwort in runden Klammern.

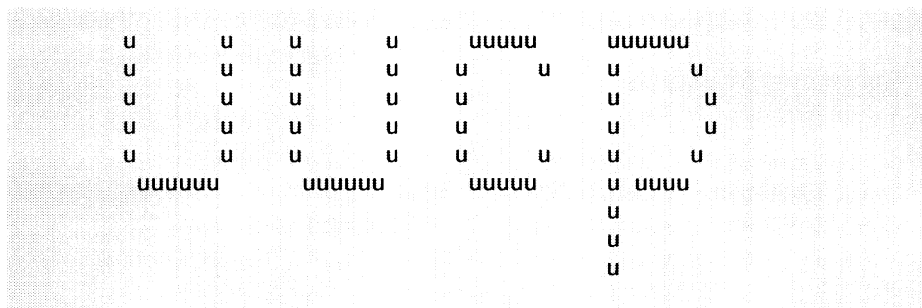
Wenn ein schwerer Fehler auftritt, bricht SCCS die Bearbeitung der aktuellen s-Datei ab und geht zur nächsten s-Datei, die angegeben ist, über.

6.1.4 Übersicht über die SCCS-Kommandos

admin	– Neue SCCS-Dateien erstellen – Parameter für eine SCCS-Datei setzen, ändern oder löschen und damit die Wirkungsweise einzelner SCCS-Kommandos beeinflussen – Beschreibenden Text in einer SCCS-Datei ablegen – Prüfen, ob eine SCCS-Datei beschädigt ist – Benutzerliste erstellen
bdiff	Große Dateien vergleichen
cdc	Kommentar bzw. MR-Nummern eines Deltas ändern
comb	Deltas zusammenfassen
delta	Delta erstellen
get	Version aus einer SCCS-Datei holen; beim Kommando <i>get</i> finden Sie auch eine Tabelle der Identifikations-Schlüsselwörter
prs	Information über eine SCCS-Datei ausgeben
rmDEL	Delta löschen
sact	Ausgeben, welche Versionen anstehen
scsdiff	Zwei Versionen vergleichen
unget	<i>get</i> -Kommando rückgängig machen
val	SCCS-Dateien auf Konsistenz prüfen
vc	Textdarstellung kontrollieren
what	Versionen identifizieren

Die Beschreibung der SCCS-Kommandos finden Sie im Buch SINIX Kommandos.

6.2 Das Programm UUCP



6.2.1 Allgemeines

Die Abkürzung UUCP steht für UNIX/UNIX Coupling Program. UUCP ist ein Kommunikations-Software-Paket für UNIX-Rechner.

Neben komfortableren Verfahren ist UUCP eine *einfache Möglichkeit* UNIX-Rechner miteinander zu koppeln und einige Kommandos auf dem Partner-Rechner auszuführen.

UUCP erlaubt den File-Transfer sowohl zwischen SINIX-Computern als auch zwischen SINIX- und UNIX-Computern, wenn für sie eine geeignete UUCP-Version erhältlich ist.

Die hier beschriebene **SINIX-Version** des UUCP, die Sie zusammen mit Ihrem SINIX-Computer erhalten haben, läuft nicht auf anderen UNIX-Rechnern.

Die Hardware der SINIX-Computer unterstützt Standleitungen. Die Kopplung über Telefonleitungen ist mittels Akustikkoppler möglich.

UUCP wurde als einfache Kommunikations-Software für Rechner-Gruppen entwickelt, die Hardware von verschiedenen Herstellern verwenden aber keine speziell modifizierten Betriebssysteme verwenden sollten. Deshalb werden nur geringe Anforderungen an die lokalen Rechner eines UUCP-Netzes gestellt.

Das Transfer-Programm ähnelt anderen Benutzern, die sich an den Rechner über normale dial-up-login-Ports anschließen und alle lokalen Schutzregeln befolgen.

Wie funktioniert UUCP?

Man unterscheidet "aktive" (auch MASTER-Rechner genannt) und "passive" (SLAVE) Rechner im UUCP-Netz.

Bei jeder Kopplung zweier Rechner müssen Sie einen Partner als aktiv31 (= Master) und den anderen Partner als passiv (= Slave) konfigurieren.

Das bedeutet für Sie, daß eine auf dem Master-Rechner angestoßene Übertragung sofort ausgeführt wird.

Ein auf dem Slave-Rechner abgesetztes UUCP-Kommando bewirkt erst dann eine Übertragung, wenn der Master-Rechner dazu auffordert.

Eine doppelte Verbindung (einmal Master-Slave und einmal Slave-Master) kann nicht eingerichtet werden, da dann keine eindeutige Rechnerkennung existiert.

Der Betrieb eines UUCP-Netzes ist einfach. Jeder teilnehmende Rechner hat ein Spool-Dateiverzeichnis (/usr/spool/uucp), in dem die auszuführende Arbeit gespeichert wird (Dateien sind zu kopieren oder Kommandos sind auf fernen Rechnern auszuführen).

Da der File-Transfer via UUCP viel Zeit in Anspruch nimmt, wird die Arbeit des Übertragens im Hintergrund ausgeführt. Die Dateien werden temporär in ein Spool-Dateiverzeichnis kopiert und dann übertragen.

Das Standardprogramm **uucico** bewerkstelligt alle Transfers. Das Programm startet, indem es einen bestimmten Übertragungskanal zu einem fernen Rechner erkennt, mit der es einen Austausch ausführen will. **uucico** sucht dann einen Anschluß, richtet die Verbindung ein, logt sich auf dem fernen Rechner ein und startet dort das **uucico**-Programm, um einen Kommunikationspartner zu haben.

Sind zwei dieser Programme verbunden, vereinbaren sie zuerst ein Protokoll und beginnen dann, Daten auszutauschen. Jedes Programm, beginnend mit dem aufrufenden (aktiver Rechner) Programm, überträgt alles was benötigt wird und fragt dann das andere Programm, was ausgeführt werden soll.

Falls keine Arbeit mehr vorhanden ist, beenden sich beide Programme.

Übertragungsmittel

Bei kleinen Entfernungen zwischen den Rechnern eines UUCP-Netzes werden Standleitungen verwendet. Diese erlauben eine schnellere Übertragung und das Vervielfachen von Verbindungsketten.

Bei Standleitungen sind maximal 2400 Baud möglich.

Sollten Sie auf Telefonverbindungen zurückgreifen müssen, können Sie UUCP mit Akustikkopplern betreiben. Der von Ihnen gewählte Akustikkoppler muß beim Einsatz in der Bundesrepublik Deutschland eine FTZ-Nummer besitzen, also von der Deutschen Bundespost geprüft und zugelassen sein.

Als Übertragungsgeschwindigkeit werden 300 und 1200 Baud angeboten.

Wer benutzt dieses Programm?

Dieses Programm ist für den versierten Systemverwalter gedacht. Ein "Benutzer" sollte sich auf einige Kommandos beschränken: uucp, uux, mail, rmail und uulog.

6.2.2 Beschreibung

6.2.2.1 Allgemeines zum Übertragen

Der normale Benutzer sollte nur die Kommandos `uucp`, `uux`, `mail`, `rmail` und `uulog` verwenden.

Es empfiehlt sich, die Abschnitte "Hinweise", "Dateien" und "siehe auch" nicht zu beachten, falls es sich um eine SINIX zu SINIX Kommunikation handelt. Bei einer SINIX zu anderen-UNIX-Versionen Kommunikation ist eine genauere Beschreibung nötig.

Als UUCP-Administrator sollten Sie den Anwendern eine Liste der anrufbaren Rechner, insbesondere deren Rechnerkennungen (`systemid`), und eine Kurzbeschreibung der Master-Slave Beziehung geben. Wichtig ist die unterschiedliche Abarbeitung von Kommandos.

In Stichpunkten:

Das Kommando `uulog` wird auf beiden Rechnern sofort ausgeführt.

Die Kommandos `uucp` und `uux` werden auf dem

- Master-Rechner sofort ausgeführt und auf dem
- Slave-Rechner erst ausgeführt, wenn der Master-Rechner den Slave-Rechner dazu auffordert. (Siehe den Zeiteintrag auf dem Master-Rechner in der Datei `/usr/lib/crontab`)

Bemerkung

Die folgenden Abschnitte dieses Kapitels beschreiben den Ablauf einer `uucp`- und einer `uux`-Übertragung. Lesen Sie diese Beschreibung genau durch, um im Debug-Fall möglichst bald die Fehlerquelle finden zu können. Des weiteren lesen Sie bitte die Texte bei den Kommandos `uucp`, `uucico` und `uux` durch. Dies sind die Mindestkenntnisse, um wirkungsvoll als UUCP-Administrator tätig zu werden.

Hinweis

Die in diesem Kapitel angegebenen Beschreibungen sind größtenteils Übersetzungen des Artikels "UUCP Implementation Description" von D.A. Nowitz (siehe Literaturverzeichnis). Dieser Text sollte somit nicht nur für SINIX-Computer gelten, sondern auch UNIX-Standard sein.

6.2.2.2 Beginn einer uucp-Übertragung

Nach dem Aufruf eines uucp-Kommandos werden der Quell-Dateiname, der Ziel-Dateiname und der Rechnernamen überprüft, um das Kommando nach einem der folgenden Typen zu klassifizieren:

- 1) Kopieren der Quelle in ein Ziel des lokalen Rechners
- 2) Erhalten von Dateien aus anderen Rechnern
- 3) Senden von Dateien auf ferne Rechner
- 4) Senden von Dateien eines fernen Rechners auf einen anderen fernen Rechner
- 5) Erhalten von Dateien eines fernen Rechners, wenn der Quell-Dateiname Shell-Sonderzeichen enthält, wie ? * [] .

Nachdem die Aufgaben im Spool-Dateiverzeichnis hinterlegt wurden, wird das uucico-Programm gestartet (außer der Schalter -r wurde angegeben; siehe Abschnitt 6.2.2.2 uucp). Dieses Programm versucht eine Verbindung aufzubauen, um die hinterlegten Aufgaben auszuführen.

Im Einzelnen wird bei den oben angegebenen Typen wie folgt verfahren:

Typ 1 Ein cp-Kommando wird benutzt, um die Aufgaben auszuführen. Die Schalter -d und -m des uucp-Kommandos werden in diesem Fall nicht unterstützt.

Typ 2 Eine einzeilige Kommandodatei (der Dateiname beginnt mit C.*) wird für jede Datei erstellt und in das Spool-Dateiverzeichnis (/usr/spool/uucp) eingehängt. Die Datei besteht aus den folgenden Feldern, wobei jedes Feld durch ein Leerzeichen getrennt ist. (Alle Kommandodateien und Befehlsdateien benutzen das Leerzeichen als Trennzeichen.)

- [1] R
- [2] Der volle Pfadname der Quelle oder ein ~benutzer/pfadname. Der Teil ~benutzer wird auf dem fernen Rechner ausgedehnt.
- [3] Der volle Pfadname der Ziel-Datei. Falls die ~benutzer-Schreibweise ~benutzt wurde, wird diese sofort in das Login-Dateiverzeichnis des Benutzers ausgedehnt.
- [4] Der Login-Name des Benutzers
- [5] Ein "-" gefolgt von einer Schalterliste. (Nur der -d und -m Schalter erscheinen in dieser Liste)

Typ 3 Für jede Quelldatei wird eine Kommandodatei erzeugt und die Quelldatei wird in eine Textdatei des Spool-Dateiverzeichnisses kopiert. (Ein Schalter "-c" im uucp-Kommando bewahrt davor, daß die Textdatei angelegt wird. In diesem Fall wird die Datei aus der angegebenen Quelle übertragen.) Die Felder sehen wie folgt aus:

- [1] S
- [2] Der volle Pfadname der Quelldatei
- [3] Der volle Pfadname des Zieles oder ~benutzer/dateiname
- [4] Der Login-Name des Benutzers
- [5] Ein "-" gefolgt von einer Schalterliste
- [6] Der Name der Textdatei im Spool-Dateiverzeichnis
- [7] Die Datei-Modus-Bits der Quelldatei in Oktalzahl-Format

Typ 4
und

Typ 5 UUCP erzeugt ein uucp-Kommando und sendet es zum fernen Rechner. Das ferne uucico-Programm führt dann das uucp-Kommando aus.

Weiter siehe Ablauf eines uucico-Aufrufs

6.2.2.3 Beginn einer uux-Übertragung

Das uux-Kommando wird benutzt, um die Ausführung eines UNIX-Kommandos zu verlagern, wobei der ausführende Rechner nicht der lokale Rechner und/oder einige Dateien nicht auf dem lokalen Rechner sind.

uux erzeugt eine Befehlsdatei (der Dateiname beginnt mit X.*), in der aufgeführt sind: die Namen der für die Ausführung benötigten Dateien (einschließlich der Standardeingabe), der Login-Name des Benutzers, das Ziel der Standardausgabe und das auszuführende Kommando. Diese Datei befindet sich entweder im Spool-Dateiverzeichnis zur lokalen Ausführung oder wird zu einem fernen Rechner mittels eines erzeugten "Sende"-Kommandos (siehe Typ 3 uucp-Beschreibung) gesandt.

Für benötigte Dateien, die nicht auf dem ausführenden Rechner sind, erzeugt uux "Erhalte"-Dateien (siehe Typ 2 uucp-Beschreibung). Diese Befehls-Dateien werden auf den ausführenden Rechner übertragen und durch das uucico-Programm abgearbeitet. Dies gelingt nur dann, falls der lokale Rechner die Erlaubnis hat, Dateien in das ferne Spool-Dateiverzeichnis einzuhängen. Die Kontrolle erfolgt durch die Datei USERFILE (/usr/lib/uucp/USERFILE) des fernen Rechners.)

Die Befehlsdatei wird durch das uuxqt-Programm auf dem ausführenden Rechner abgearbeitet. Diese Datei besteht aus mehreren Zeilen, jede davon enthält ein Identifikationszeichen und eines oder mehrere Argumente. Die Reihenfolge der Zeilen ist nicht relevant. Die Zeilen sehen wie unten beschrieben aus:

Benutzerzeile U benutzer rechner

wobei benutzer der Login-Name und rechner die Rechnerkennung des Auftraggebers sind.

Dateienzeile F dateiname realname,

wobei dateiname der erzeugte Name einer Datei für den ausführenden Rechner und realname der letzte Teil des aktuellen Dateinamens ist (er enthält keine Pfadinformation). Es können keine oder mehrere von diesen Zeilen in der Befehlsdatei vorkommen. Das uuxqt-Programm wird die Existenz der gewünschten Dateien überprüfen, bevor das Kommando abgearbeitet wird.

Standard-Eingabe-Zeile I dateiname

Die Standardeingabe ist entweder durch ein "<" in der Kommandofolge angegeben oder folgt aus der Standardeingabe des uux-Kommandos, falls der Schalter "-" angegeben ist. Falls keine Standardeingabe angegeben ist, wird "/dev/null" benutzt.

Standard-Ausgabe-Zeile O dateiname rechnername

Die Standardausgabe wird durch ein ">" in der Kommandofolge angegeben. Falls keine Standardausgabe angegeben wurde, wird "/dev/null" benutzt. (Der Gebrauch von ">>" ist nicht implementiert.)

Kommandozeile C kommando [argument [argument ...]]

Die Argumente sind diejenigen, die in der Kommandofolge angegeben wurden. Die Standardeingabe und -ausgabe erscheinen nicht in dieser Zeile. Alle benötigten Dateien werden in das Befehlsdateiverzeichnis (ein Unterdateiverzeichnis des Spool-Dateiverzeichnisses) eingetragen und das UNIX-Kommando wird abgearbeitet, indem die Shell benutzt wird. Zusätzlich wird ein Shell-"PATH"-statement, wie in dem uuxqt-Programm angegeben, der Kommandozeile hinzugefügt.

Weiter siehe Ablauf eines uucico-Aufrufs und Kommando uuxqt

6.2.2.4 Ablauf eines uucico-Aufrufs

Das uucico-Programm erfüllt die folgenden Hauptfunktionen:

- Durchsuchen des Spool-Dateiverzeichnisses nach Arbeit
- Einen Ruf an einen fernen Rechner absetzen
- Umsetzen des benutzten Protokolls
- Ausführen der Anforderungen beider Rechner
- Protokollanforderungen und Arbeitsabschluss

Uucico kann auf verschiedene Weisen gestartet werden:

- von einem System daemon (z.B.: cron)
- von einem der Programme uucp, uux, uuxqt oder einem uucico-Aufruf eines fernen Rechners
- direkt vom Benutzer (gewöhnlich zu Testzwecken)
- von einem fernen Rechner (Das uucico-Programm sollte angegeben sein als das "Shell"-Feld in der Datei /etc/passwd für die UUCP-logins.). Auf den SINIX-Computer somit beim Eintrag `remuucp::.....:/usr/lib/uucp/uucico` .

Wenn uucico auf die ersten drei Arten aufgerufen wird, wird das Programm als im MASTER-Modus angenommen. In diesem Modus wird eine Verbindung zu einem fernen Rechner aufgebaut. Falls uucico von einem fernen Rechner gestartet wird, wird das Programm als im SLAVE-Modus angenommen. Der MASTER-Modus wird auf eine der zwei Arten operieren:

- Falls keine Rechnerkennung angegeben ist (der Schalter -s wurde nicht angegeben; siehe Abschnitt 6.2.2.2 uucp), überprüft das Programm das Spool-Dateiverzeichnis nach Rechnern, die gerufen werden sollen.
- Falls eine Rechnerkennung angegeben ist, wird der betreffende Rechner gerufen und nur die Arbeit für dieses System wird ausgeführt.

Überprüfen

Die Namen der Arbeitsdateien im Spool-Dateiverzeichnis (/usr/spool/uucp/...) haben das Format:

Typ . systemid Trennzeichen Zahl

Typ	ist ein Großbuchstabe und zwar:
	C für die Kommandodatei (command file)
	D für die Textdatei (data file)
	X für die Befehlsdatei (executive file)
systemid	die Rechnerkennung des fernen Rechners
Trennzeichen	ein Buchstabe (meist n)
Zahl	eine vierstellige, fortlaufende Zahl (entnommen der Datei /usr/spool/uucp/SEQF)

z.B.: Die Datei C.res45n0031 enthält die Kommandos zur Übertragung einer Datei auf den Rechner res45

Die Überprüfung erfolgt in der Weise, daß das Spool-Dateiverzeichnis nach Dateien mit dem Präfix C durchsucht wird. Es wird dann eine Liste der zu rufenden Rechner erstellt. Uucico ruft dann jeden Rechner auf und führt die Übertragungen aus.

Aufruf

Der Aufruf wird ausgeführt, indem Informationen von verschiedenen Dateien, die im UUCP-Programm-Dateiverzeichnis (/usr/lib/uucp/...) vorliegen, benutzt werden. Zu Beginn des Aufruf-Prozesses wird eine Sperre angelegt, um mehrfache Unterhaltung zwischen zwei gleichen Rechnern zu vermeiden.

Die Rechnerkennung wird in der Datei L.sys gefunden. Die für jeden Rechner enthaltene Information ist:

- [1] Rechnerkennung
- [2] Zeit, wann der Rechner gerufen wird (Wochentage und Tageszeit)
- [3] Anschluß oder Anschlußtyp, der für den Anruf benutzt werden soll
- [4] Übertragungsgeschwindigkeit
- [5] Anschlußname (der gleiche wie in Feld [3]) oder Telefonnummer, falls in Feld [3] ACU angegeben ist.
- [6] Login-Informationen

Das Zeitfeld wird mit der momentanen Zeit verglichen, um zu sehen, ob der Anruf durchgeführt werden soll. Zu den in der /usr/lib/crontab dem MASTER-Rechner eingetragenen Zeiten wird uucico gestartet.

Die Telefonnummer kann Abkürzungen enthalten (z.B.: mh, py, boston), die in "dial sequences" übersetzt werden unter Verwendung der Datei L.dialcodes (siehe Abschnitt 6.2.2.5 Sonstige Dateien).

(weiteres zu L.sys im Abschnitt "Dateien, die bei einer UUCP-Installation angelegt werden" siehe Abschnitt 6.2.2.5 Sonstige Dateien)

Die Datei L.devices (siehe Abschnitt 6.2.2.5 Sonstige Dateien) wird mittels der Felder [3] und [4] der Datei L.sys überprüft, um einen wählbaren Anschluß für den Aufruf zu finden. Das Programm wird alle Anschlüsse versuchen, bis der Anruf erfolgte oder keine Anschlüsse mehr versucht werden können. Falls ein Anschluß erfolgreich geöffnet wurde, wird eine Sperrdatei erzeugt, so daß eine andere Kopie von uucico diesen Anschluß nicht versuchen wird. Falls der Anruf aufgebaut ist, wird die Login-Information (Feld [6] von L.sys) zum Einloggen benutzt.

Die Unterredung zwischen zwei uucico-Programmen beginnt mit einem gegenseitigen Gruß, der von dem gerufenen "SLAVE" Rechner begonnen wird. Der SLAVE sendet eine Nachricht, um dem MASTER mitzuteilen, daß der SLAVE bereit ist, die Rechnerkennung und die fortlaufende Übertragungsnummer zu empfangen. Die Antwort des MASTER wird von dem SLAVE erkannt und, falls annehmbar, beginnt die Protokollführung. Der SLAVE kann auch mit einer "call-back required" Nachricht antworten, wodurch die laufende Unterredung beendet ist.

Protokoll

Der ferne Rechner sendet eine Meldung

Pproto-list

wobei proto-list eine Zeichenfolge ist, jedes repräsentiert eine Protokollzeile.

Das rufende Programm überprüft proto-list nach einem Buchstaben, der mit einer verfügbaren Protokollzeile übereinstimmt und schickt eine use-protocol Mitteilung zurück. Die use-protocol Mitteilung ist

Ucode

wobei code entweder ein Einzeichen-Protokoll-Brief ist oder N, was bedeutet, daß es kein allgemeines Protokoll gibt. Danach wird das Protokoll, auf das sich beide Partner in dieser Art Handshaking geeinigt haben, für den folgenden Datentransfer benutzt. wobei code entweder ein Einzeichen-Protokoll-Brief ist oder N, was bedeutet, daß es kein allgemeines Protokoll gibt. Danach wird das Protokoll, auf das sich beide Partner in dieser Art Handshaking geeinigt haben, für den folgenden Datentransfer benutzt.

Ausführung

Die ursprünglichen Rollen (MASTER oder SLAVE) für die auszuführende Arbeit sind der Modus, in welchem jedes Programm startet. (Der MASTER wird mit dem Schalter "-r1" angegeben; siehe Abschnitt 6.2.4.3 uucp.) Das MASTER-Programm führt eine Sucharbeit ähnlich derjenigen aus, die im Abschnitt "Beginn einer uucp-Übertragung" beschrieben ist.

Es gibt fünf Mitteilungen, die während des Arbeitsprozesses benutzt werden, jede klassifiziert durch das erste Zeichen der Mitteilung. Diese sind:

- S Senden einer Datei
- R Erhalten einer Datei
- C Kopie vollständig
- X Führe ein uucp-Kommando aus
- H Endesignal (hangup)

Der MASTER wird R, S oder X Mitteilungen senden, bis alle Arbeit vollendet ist. Zu diesem Zeitpunkt wird eine H Mitteilung gesendet. Der SLAVE antwortet mit SY, SN, RY, RN, HY, HN, XY, XN, je nachdem ob mit ja oder nein geantwortet wird.

Die Sende- und Erhalte-Antworten basieren auf der Erlaubnis, auf die gewünschten Dateien bzw. Dateiverzeichnisse zugreifen zu dürfen. Diese Information erhält man unter Benutzung der Datei USERFILE und der Lese-/Schreib-Erlaubnis für die Datei bzw. das Dateiverzeichnis. Nachdem jede Datei in das Spool-Dateiverzeichnis kopiert ist, wird eine "Kopie vollständig" Mitteilung von dem Empfänger der Datei gesandt. Die Mitteilung CY wird gesandt, falls die Datei erfolgreich von der temporären Spool-Datei in das tatsächliche Ziel übertragen ist. Andernfalls wird eine CN Mitteilung gesandt. (Im Falle von CN, ist die übertragene Datei im Spool-Dateiverzeichnis (/usr/spool/uucp/...) mit einem Dateinamen, der mit TM. beginnt, vorhanden.) Die Anforderungen und die Ergebnisse werden in beiden Rechnern protokolliert.

ENDE

Wenn eine HY Mitteilung von dem MASTER empfangen wird, wird dies an den SLAVE zurück gesandt und die Protokolle werden beendet. Jedes Programm sendet ein abschließendes "OO" zu dem anderen. Das ursprüngliche SLAVE-Programm wird aufräumen und beenden. Der MASTER wird fortfahren andere Rechner anzurufen oder die Arbeit beenden, falls ein -s Schalter angegeben wurde (siehe Abschnitt 6.2.4.3 uucp).

6.2.2.5 Sonstige Dateien

Übersicht der von UUCP betroffenen Dateien bzw. Dateiverzeichnisse:

/bin/mail
/bin/rmail

/etc/group
/etc/passwd
/etc/systemid
/etc/ttys
/etc/ttytype

/usr/bin/uucp
/usr/bin/uulog
/usr/bin/uux

/usr/lib/crontab
/usr/lib/uucp/L-devices
/usr/lib/uucp/L-dialcodes
/usr/lib/uucp/L.sys
/usr/lib/uucp/USERFILE
/usr/lib/uucp/uucico
/usr/lib/uucp/uuclean
/usr/lib/uucp/uuday
/usr/lib/uucp/uuhour
/usr/lib/uucp/uuxqt

/usr/spool/uucp/.XFDDIR/...
/usr/spool/uucp/.XQTDIR/...
/usr/spool/uucp/LOGFILE
/usr/spool/uucp/SEQF
/usr/spool/uucp/SYSLOG
/usr/spool/uucp/...

/usr/spool/uucp/uucppublic/...

Dateien, die bei einer System-Installation angelegt sind:

```
/usr/bin:      (drwxr-xr-x  2 bin 192 /usr/bin)
-rws--x--x  1 uucp   23226 Apr  1 10:49 uucp      *)
-rws--x--x  1 uucp   13418 Apr  1 10:52 uulog      *)
-rws--x--x  1 uucp   22206 Apr  1 10:52 uux        *)

/usr/lib/uucp:  (drwxrwxrwx  2 uucp 176 /usr/lib/uucp)
-rw-r--r--  1 uucp      0 Apr  1 13:27 L-devices
-rw-r--r--  1 uucp     17 Apr  1 13:27 L-dialcodes
-rw-r--r--  1 uucp      0 Apr  1 13:27 L.sys
-rw-r--r--  1 uucp     21 Apr  1 13:26 USERFILE
-rws--x--x  1 uucp   50718 Apr  1 12:50 uucico      *)
-rws--x--x  1 uucp    8886 Apr  1 12:54 uuclean     *)
-rwxr-xr-x  1 daemon   153 Mai 20 12:34 unday
-rwxr-xr-x  1 daemon   192 Apr  1 13:27 unhour
-rws--x--x  1 uucp   22308 Apr  1 10:55 uuxqt       *)

/usr/spool/uucp: (drwxrwxrwx  2 uucp  64 /usr/spool/uucp)
-rw-----  1 uucp      0 Apr  2 08:41 LOGFILE
-rw-----  1 uucp      0 Apr  2 08:41 SYSLOG
```

*) Beschreibung dieser Dateien: siehe Kommandobeschreibungen.

Bemerkung

Die Datei /usr/spool/uucp/SEQF wird durch den ersten UUCP- oder uux-Aufruf angelegt.

Datei: L.sys (/usr/lib/uucp/L.sys)

Jede Zeile in dieser Datei repräsentiert einen Rechner, die von dem lokalen uucp-Programm angesprochen werden kann.

Die Zeilen sind wie folgt aufgebaut:

(Die Einträge sind Beispiele. Ihre Werte entnehmen Sie bitte unten folgenden Erklärungen.) z.B.:

```
gruen Any,10.tty05 1200 tty05 -remuucp-Kennwort: siemens
```

Rechnerkennung der fernen Rechner

Zeit

Anschluß

Übertragungsgeschwindigkeit

Teilnehmeranschluß

Login-Kennung

Anlagenkennung

Hier ist die Rechnerkennung des fernen Rechners eingetragen. (Eindeutig! Also für jeden Rechner nur eine Zeile.)

Zeit

1) Master-Rechner

Dies ist eine Zeichenfolge, die den Wochentag und die Tageszeit angibt, wann der Rechner gerufen werden soll (z.B.: MoTuTh0800-1730).

Der Wochentag kann eine Liste sein, die aus Su Mo Tu We Th Fr Sa aufgebaut sein kann oder es kann hier Wk für einen beliebigen Wochentag oder Any für einen beliebigen Tag stehen.

Die Tageszeit sollte ein Zeitbereich sein (z.B.: 0800-1230). Falls keine Tageszeit angegeben ist, wird jede beliebige Zeit als ok für den Anruf angenommen. (Der Eintrag ,10 gibt an, daß 10 Minuten gewartet wird bis ein neuer Aufruf erfolgt -> siehe Datei STST.*)

2) Slave-Rechner

Hier wird 0 eingetragen.

Anschluß

Hier steht entweder ACU oder der Standleitungsanschluß, der für den Aufruf benutzt wird. Im Falle der Standleitung wird der letzte Teil des für den Transfer benutzten Dateinamens angegeben (z.B.: tty05).

Übertragungsgeschwindigkeit

Hier ist die Übertragungsgeschwindigkeit hinterlegt. (Zur Zeit: 300 oder 1200 [Baud])

Teilnehmeranschluß

Die Telefonnummer ist aufgebaut aus einer frei wählbaren alphabetischen Abkürzung und einem Zahlenteil. Die Abkürzung muß eine der in der Datei L-dialcodes (siehe Abschnitt 6.2.2.5 Sonstige Dateien) angegebenen sein (z.B.: mh5900, boston995-9980).

Im Falle einer Standleitung enthält dieses Feld die gleiche Zeichenfolge wie das Anschluß-Feld.

Login-Kennung

In diesem Feld steht bei SINIX-Computern:

-remuucp-Kennwort: siemens

Die Übersetzung der allgemeinen Beschreibung lautet:

Die Login-Information ist in der Form einer Reihe von Feldern und Unterfeldern angegeben:

expect send [expect send] ...

wobei expect die Zeichenfolge ist, die als Eingabe erwartet wird, und send ist die Zeichenfolge, die gesandt wird, falls eine expect-Zeichenfolge erhalten wurde.

Das expect-Feld kann aus Unterfeldern folgender Form aufgebaut sein:

expect[-send-expect]...

wobei die send-Zeichenfolge gesandt wird, falls das vorhergehende expect nicht erfolgreich gelesen wurde und die expect-Zeichenfolge nach der send-Zeichenfolge als nächste Zeichenfolge erwartet wird. Entspricht die eingelesene Zeichenfolge nicht der erwarteten Sequenz, wird mit Fehlerfall die Übertragung abgebrochen.

Es gibt zwei bestimmte Namen, die ausgewählt werden können, um während der Login-Sequenz gesandt zu werden. Die Zeichenfolge EOT wird das EOT-Zeichen senden und die Zeichenfolge BREAK wird einen BREAK-Zustand erzeugen. (BREAK ist ein Zustand auf der Leitung. Er wird simuliert unter Benutzung von Übertragungsgeschwindigkeitswechseln und Null-Zeichen und kann nicht an allen Anschlüssen und/oder Rechnern ablaufen.)

Beispiele zu L.sys

- 1) Ein typischer Eintrag in der L.sys-Datei ist zum Beispiel:
sys1 Any ACU 300 mh7654 login uucp ssword: word

Der send-Algorithmus gleicht im letzten Teil der Zeichenfolge dem Eintrag in der Datei /etc/passwd des Kommunikationspartners. Die Sequenz uucp word ist nötig, um sich dort einzuloggen.

- 2) Bei einer SINIX-SINIX-Kopplung ist folgender Eintrag ein Beispiel für den Aufbau einer Zeile auf einem Master-Rechner:

grün Any,10 v24 1200 v24 -remuucp-Kennwort: siemens

wobei:

grün der Name des Partnerrechners ist, der dort in der Datei /etc/systemid hinterlegt ist

Any,10 der Zeiteintrag ist. Auf der Slave-Seite ist der Eintrag eine 0

v24 der Anschluß ist, über den die Kopplung erfolgt

1200 Übertragungsgeschwindigkeit: 1200 Baud

-remuucp die Benutzerkennung auf des fernen Rechners ist, unter der sich uucp einloggt

-Kennwort die Meldung ist, auf die der lokale Rechner wartet

siemens das Kennwort des fernen Rechners für die Benutzerkennung remuucp ist. Das ist die Standardeinstellung auf Ihrem Rechner nach Systeminstallation.

Hinweis

Als weiteres Beispiel sind hier die Einträge in der Datei L.sys einer Vax angegeben.

Die erste Zeile ist der Eintrag betreffend einer SINIX-Anlage, die als MASTER konfiguriert wurde (daher kein Login-Eintrag auf der SLAVE-Seite = Vax). Die zweite Zeile ist der Eintrag betreffend einer SINIX-Anlage, die als SLAVE konfiguriert wurde.

```
" siem1 Never DIR 1200 tty15
  siem2 Any DIR 1200 tty09 "" "" kennung:--kennung: remuucp "
```

Datei: L-dialcodes (/usr/lib/uucp/L-dialcodes)

Diese Datei enthält auf SINIX-Computern:

dummy 0123456789

Diese Datei enthält Einträge mit den lokalen Abkürzungen für Teile von Telefonnummern, die in der Datei L.sys benutzt werden.

Das Format ist:

abb dial-seq

wobei

abb	die Abkürzung ist und
dial-seq	die Zahlenfolge ist, die gewählt werden muß, um den gemeinten Ort zu erreichen.

Der Eintrag py 165- wird den Eintrag py7777 ersetzen durch 165-7777 und der Wählvorrichtung senden.

Datei: L-devices (/usr/lib/uucp/L-devices)

Diese Datei enthält Einträge für die Wählvorrichtungen und Standleitungen, die von uucp benutzt werden. Von den speziellen Anschlußdateien wird angenommen, daß sie im Dateiverzeichnis /dev hinterlegt sind. Das Format für jeden Eintrag ist:

line call-unit speed

wobei

line	der Anschluß für die Verbindung ist,
call-unit	die automatische Wählvorrichtung verbunden mit der "line" ist,
speed	Standleitungen haben die Zahl 0 in diesem Feld und die Übertragungsgeschwindigkeit ist.

Der Eintrag cul0 cua0 300 würde für einen Rechner eingetragen sein, der den Anschluß cul0 hat, der mit der Wählvorrichtung cua0 verbunden ist und die Übertragungsgeschwindigkeit 300 Baud benutzt.

Beispiel für eine Standleitung, die über v24 installiert ist und mit 1200 Baud betrieben wird:

v24 0 1200

(wobei v24 der unter /dev erzeugte Eintrag ist. Das Kommando, z.B. auf dem PC-MX lautet: /etc/mknod /dev/v24 c 3 131)

Datei: USERFILE (/usr/lib/uucp/USERFILE)

Diese Datei enthält die Informationen über die Zugriffsberechtigungen der Benutzer. Es werden vier Arten von Beschränkungen unterschieden:

- [1] auf welche Dateien kann ein normaler Benutzer des lokalen Rechners zugreifen,
- [2] auf welche Dateien kann ein ferner Rechner zugreifen,
- [3] welcher Login-Name wird von einem bestimmten fernen Rechner benutzt,
- [4] ob bei einem fernen Rechner zurückgefragt werden soll, um sich ihre Identität bestätigen zu lassen.

Jede Zeile in der Datei hat das folgende Format:

login,sys [c] pfadname [pfadname...]

wobei

login	der Loginname für einen Benutzer oder einen fernen Rechner ist,
sys	die Rechnerkennung eines fernen Rechners ist,
c	der Ihnen anheimgestellte "call-back required" Schalter ist und
pfadname	ein Pfadnamen-Präfix ist, auf das der Benutzer zugreifen dürfen soll.

Die Beschränkungen sind wie folgt eingebaut:

- [1] Wenn das Programm ein auf dem lokalen Rechner gespeichertes Kommando ausführt (MASTER-Modus), sind die erlaubten Pfadnamen jene, die in der ersten Zeile in der Datei USERFILE angegeben sind. Diese Zeile enthält einen Login-Namen der mit dem Login-Namen des Benutzers, der das Kommando absetzte, verglichen wird.
- [2] Wenn das Programm auf ein Kommando eines fernen Rechners antwortet (SLAVE-Modus), sind die erlaubten Pfadnamen jene, die in der ersten Zeile, die die Rechnerkennung enthält, die mit der Rechnerkennung des fernen Rechners verglichen wird. Falls keine Zeile dieser Art gefunden wird, wird die erste Zeile ohne Rechnerkennung herangezogen.

[3] Wenn ein ferner Rechner sich einloggt, muß der Login-Name, den sie benutzt, in der Datei USERFILE auftauchen.

[4] Falls die von [3] betroffene Zeile ein c enthält, wird bei dem fernen Rechner zurückgefragt, bevor irgendein Transfer stattfindet.

Die Zeile u,m /usr/xyz erlaubt dem Rechner m sich mit dem Namen u einzuloggen und den Transfer von Dateien zu verlangen, deren Name mit "/usr/xyz" beginnt. Die Zeile dan, /usr/dan erlaubt dem normalen Benutzer dan Kommandos auf Dateien anzuwenden, deren Name mit "/usr/dan" beginnen. Die Zeilen u,m /usr/xyz /usr/spool u, /usr/spool erlauben jedem fernen Rechner sich mit dem Namen u einzuloggen, aber falls die Rechnerkennung nicht m ist, kann dieser Rechner nur verlangen, Dateien zu transferieren, deren Name mit "/usr/spool" beginnen. Die Zeilen root, /, /usr erlauben jedem Benutzer Dateien zu transferieren, deren Name mit "/usr" beginnen, jedoch der Benutzer root kann jede Datei transferieren. Der Standardeintrag auf Ihrem Rechner remuucp, /usr lautet nach der Installation: /usr

uuday (/usr/lib/uucp/uuday)

Uday bereinigt täglich das Spool-Dateiverzeichnis:

- Im Spool-Dateiverzeichnis (/usr/spool/uucp) werden die verbliebenen temporären Dateien, wie z.B: Sperrdateien etc., und die nicht abgearbeiteten Kommandodateien und die dazugehörenden Textdateien gelöscht (siehe uuclean).
- Im allgemein zugänglichen Spool-Dateiverzeichnis (/usr/spool/uucp/uucppublic) werden alle Dateien gelöscht, die älter als 7 Tage sind.

uuhour (/usr/lib/uucp/uuhour)

Stündlich werden alle angefallenen Protokolle in die Datei LOGFILE eingebunden und die letzten zehn Einträge bleiben erhalten.

Datei: LOGFILE (/usr/spool/uucp/LOGFILE)

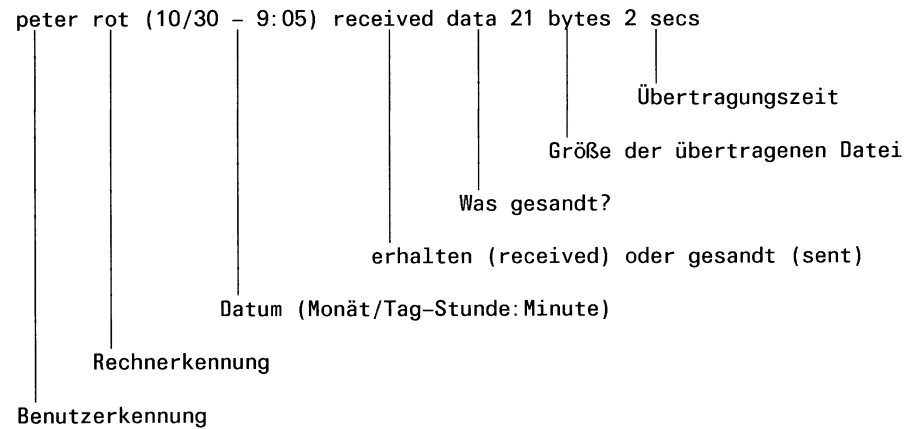
Während der Ausführung von Programmen werden individuelle LOG-Dateien im Spool-Dateiverzeichnis angelegt, die Informationen enthalten betreffend: aufgelaufener Anforderungen, Aufrufe an ferne Rechner, Ausführungen von uux-Kommandos und Ergebnisse vom Dateien-Kopieren. Diese Dateien sind zusammengefaßt in der Datei LOGFILE unter Zuhilfenahme des Programmes uulog (siehe Abschnitt 6.2.4.2 uulog). Dieses Programm wird die neue LOG-Datei an den Anfang der existierenden Datei LOGFILE einfügen. Das Kommando "uulog" wird die Beimischung vervollständigen. Es stehen einige Schalter zur Verfügung um einige oder alle Einträge auszugeben. Die Datei sollte periodisch bereinigt werden. Weiteres siehe dazu auch im Kapitel Kommandos uulog (siehe Abschnitt 6.2.4.2 uulog) und uuclean (siehe Abschnitt 6.2.4.7 uuclean) (und s.o. uuhour, uuday)

Datei: SEQF (/usr/spool/uucp/SEQF)

In dieser Datei ist eine vierstellige Zahl hinterlegt, die nach jeder Übertragung weiter hochgezählt wird. Diese Datei wird durch den ersten uucp-Aufruf angelegt.

Datei: SYSLOG (/usr/spool/uucp/SYSLOG)

In dieser Datei wird ein Zeilenprotokoll zu den Übertragungen mit dem lokalen Rechner hinterlegt. Der Eintrag ist wie folgt aufgebaut; z.B.:



3) Dateien, die bezüglich UUCP geändert wurden

In der Datei /etc/group ist eingetragen:

```
uucp::4: uucp, remuucp
```

In der Datei /etc/passwd ist folgender Eintrag:

```
uucp: cHUkyKx8z6QP.:6:4::/usr/spool/uucp/uucppublic:
/bin/sh *) **)
remuucp: cHUkyKx8z6QP.7:4::/usr/spool/uucp/uucppublic:
/usr/lib/uucp/uucico **)
```

*) hier wäre sinnvoll:

```
uucp:x:6:4::/usr/spool/uucp/uucppublic:/bin/sh
```

**) das verschlüsselte Passwort ist: siemens

In der Datei /usr/lib/crontab befinden sich folgende Einträge:

```
#      56 * * * * /usr/lib/uucp/uuhour
#      58 12 * * * /usr/lib/uucp/uuday
```

Die eingetragenen Nummernzeichen sind zu löschen, wenn Sie mit UUCP arbeiten wollen.

An dem MASTER-Rechner sollten folgende Einträge hinzugefügt werden:

```
0 * * * * /usr/lib/uucp/uucico -r1 -sgrün
```

(grün ist die Rechnerkennung des Partner-Slave-Rechner)

Tip: Falls Sie häufiger von dem SLAVE-Rechner aus übertragen, so empfiehlt es sich von dem Einstundenintervall auf einen anderen z.B. 5-Minuten Abstand überzugehen. Dies bewerkstelligen Sie, indem Sie statt der 0 eintragen: 0,5,10,15,20,25,30,35,40,45,50,55.

- 3a) Vor einer Kopplung mittels UUCP müssen Sie folgende Dateien ändern:

In der Datei /etc/ttytype wird

- a) auf dem MASTER-Rechner nichts hinzugefügt
- b) auf dem SLAVE-Rechner die folgende Zeile hinzugefügt:

none	tty05	1200	1200	RW	RW
				Raw-Modus	
			Übertragungsgeschwindigkeit		
	Anschluß				
kein Terminal					

In der Datei /etc/ttys wird

- a) auf dem MASTER-Rechner nichts hinzugefügt
- b) auf dem SLAVE-Rechner folgende Zeile hinzugefügt:

10tty05

wobei tty05 der Anschluß ist, über den die Übertragung realisiert wird und 10 anzeigt, daß auf diesem Anschluß ein Login-Prozeß liegt. (00tty05 = kein Login-Prozeß am Port tty05.)

Mit dem Kommando `kill -2 1` (nur `su`) teilen Sie die Änderung dem System mit.

Hinweis

Die Einträge auf dieser Seite geben nur die Einträge bei Ihrer Installation auf SINIX-Seite wieder.

Bemerkung

Anweisungen zur Inbetriebnahme von UUCP finden Sie im Abschnitt 6.2.3 Installation.

- 4) Temporäre Dateien, die durch eine uucp-Übertragung angelegt sein können

Datei: TM.* (/usr/spool/uucp/TM.*)

Diese Dateien werden in dem Spool-Dateiverzeichnis erzeugt, während Dateien von einem fernen Rechner übertragen werden. Ihre Namen haben die Form: TM.pid.ddd, wobei pid eine Prozeßnummer (process-id) und ddd eine dreistellige fortlaufende Nummer ist, die für jeden Aufruf von uucico mit 000 beginnt und fortgezählt wird mit jeder empfangenen Datei. Nachdem die ganze ferne Datei empfangen wurde, wird die TM-Datei an das gewünschte Ziel übertragen. Falls der Prozeß abnormal beendet wurde oder die Übertragung fehlerhaft war, verbleibt die Datei im Spool-Dateiverzeichnis. Die übriggebliebenen Dateien sollten periodisch entfernt werden; das uclean Programm ist dafür sehr nützlich. Das Kommando uclean -pTM wird alle TM-Dateien löschen, die älter als drei Tage sind (siehe auch Eintrag in der Datei uuday).

Datei: STST.* (/usr/spool/uucp/STST.*)

Diese Dateien werden im Spool-Dateiverzeichnis durch das uucico-Programm erzeugt. Sie enthalten Informationen von Fehlerquellen wie login, dialup oder sequence check und werden einen TALKING-Status enthalten, wenn die Rechner übertragen. Der Dateiname hat die Form: STST.sys, wobei sys die Rechnerkennung des fernen Rechners ist. Bei normalen Fehlerquellen (dialup, login) ermöglicht diese Datei für circa eine Stunde keine weiteren Versuche. Bei der Fehlerquelle sequence check muß die Datei entfernt werden, bevor irgendein weiterer Versuch unternommen wird, zu diesem fernen Rechner zu übertragen. Falls diese Datei von einem Programm/System-Abbruch stammt, kann sie einen TALKING-Status enthalten. In diesem Fall muß die Datei entfernt werden, bevor eine Übertragung versucht wird.

(Während der Übertragung tritt auch der Status CONVERSATING auf.)

Datei: LCK.* (/usr/spool/uucp/LCK.*)

Diese Sperrdateien (lock files) werden für jeden Anschluß erzeugt, der momentan benutzt wird und für jeden Rechner, mit dem übertragen wird. Dies bewahrt vor doppelter Übertragung und mehrfacher Benutzung desselben Anschlusses. Der Name der Sperrdatei hat die Form: LCK., wobei str entweder ein Anschluß oder eine Rechnerkennung ist. Diese Dateien können im Spool-Dateiverzeichnis verbleiben, falls der Ablauf unterbrochen wurde. Sie werden nach einer Zeit von circa 24 Stunden ignoriert. Falls der Ablauf unterbrochen wurde und Übertragungen vor dieser Zeit gewünscht werden, müssen diese Dateien entfernt werden.

6.2.3 Installation

6.2.3.1 Hardware

Bestellnummern der Kopplungskabel

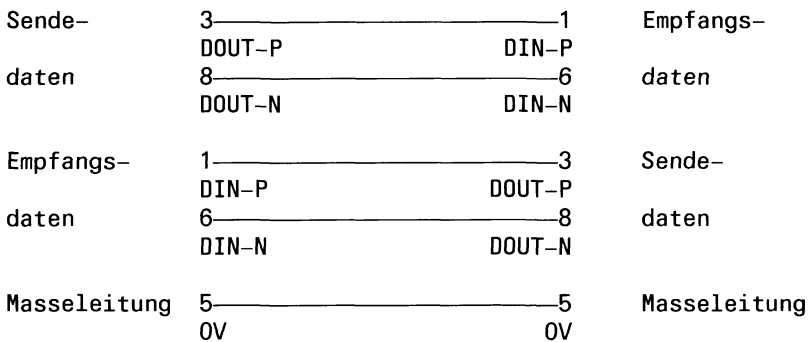
Zur Kopplung zweier Rechner können Sie bei Ihrer Zweigniederlassung folgende Kabelversionen bestellen:

- Kopplung über SS97 : 97001-10
- Kopplung über v24 : 97001-11

Sollten diese Kabel nicht verfügbar sein, ist in den folgenden Abschnitten beschrieben, wie Sie sich ein Kabel erstellen können.

1) SS97 Anschluß

Erstellen Sie sich ein Kabel mit den folgenden Verbindungen (Kreuzung TxD RxD)



Verbinden Sie jedoch nicht andere Anschlüsse, da Sie sonst beim Ausschalten eines Rechners die mit ihm verbundenen Rechner mit abschalten.

Vorschlag

Sie bestellen die unten angegebenen Teile, entfernen beim Kabel den Mutterstecker und montieren den Vaterstecker aus dem bestellten Steckersatz nach obiger Anweisung.

Bestelldaten:

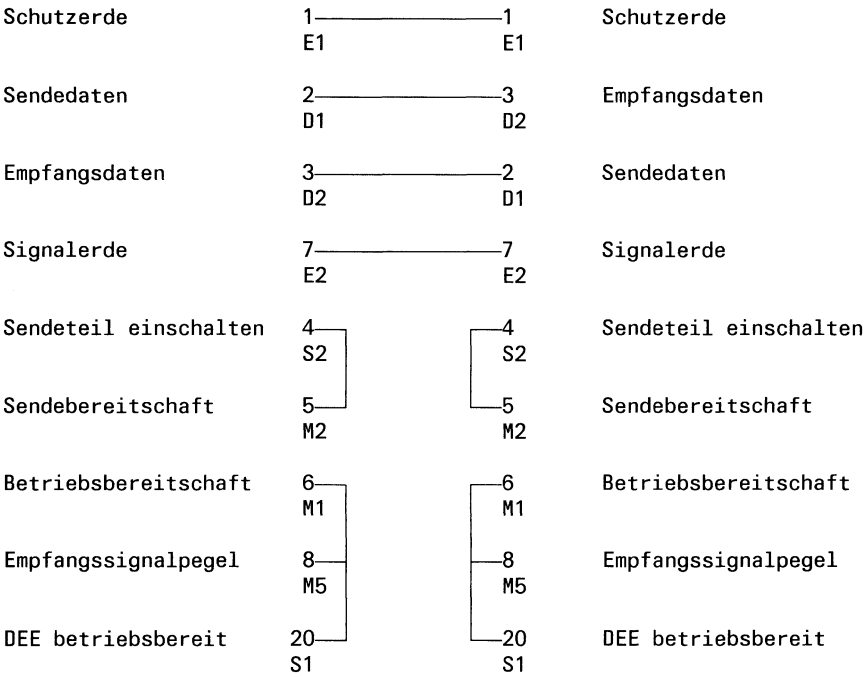
	Bezeichnung	Sachnummer	Lagernummer
1 Kabel mit Stecker	SS 97	--	1000829
1 Steckersatz	1 Satz V11 Stecker	V26893B9V1	1078186

2) V24 Anschluß

– Für die Kabelerstellung haben Sie drei Möglichkeiten

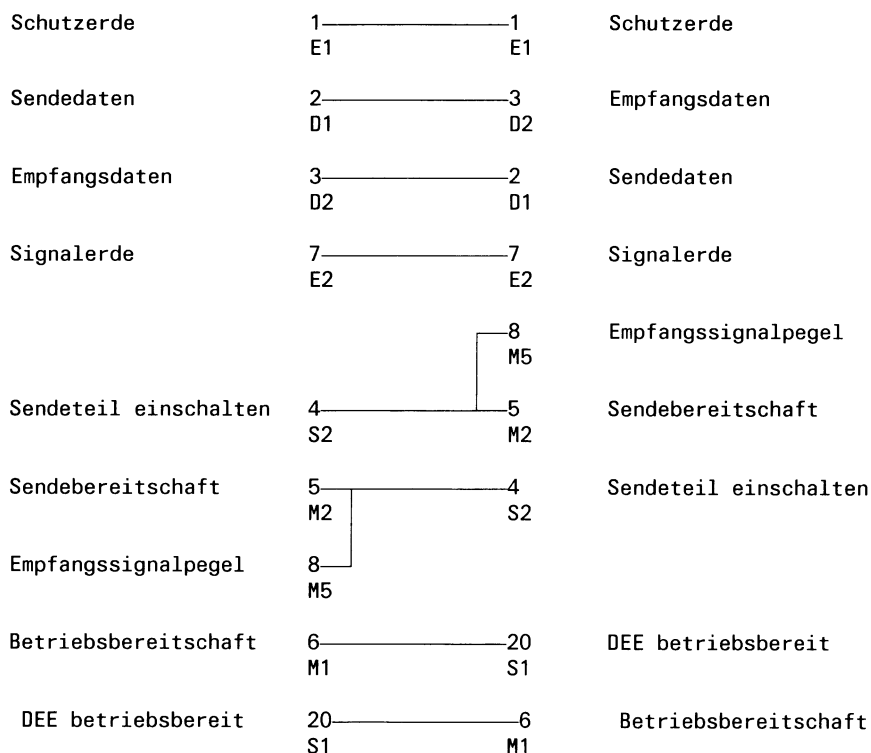
1. Die "optimistische" Version:

Die Bezeichnung "optimistisch" heißt: Es wird davon ausgegangen, daß die Partneranlage betriebsbereit ist. Mittels dieser Kabelversion können Sie dies nicht kontrollieren.



– Auf der CONAC-Baugruppe müssen die Schalter S4-S7 in die Stellung 2 ("schräg") gebracht werden.

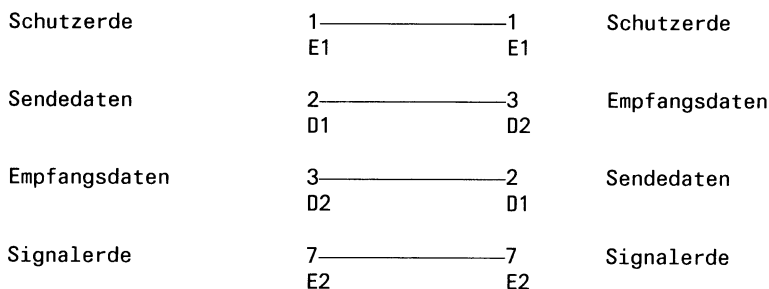
2. Die ideale Version:



- Auf der CONAC-Baugruppe müssen die Schalter S4 - S7 in die Stellung 2 ("schräg") gebracht werden.

3. Für eine Kopplung PC-MX - PC-MX genügt folgende Kabelversion: (Achtung: nicht für PC-X - Kopplung geeignet!)

(Alle anderen Anschlüsse, insbesondere M1 - M5, werden auf dem PC-MX intern simuliert.)



Bitte beachten:

- Auf der CONAC-Baugruppe darf nur der Schalter S4 in die Stellung 2 ("schräg") geschaltet werden.
- Die Schalter S5, S6 und S7 müssen in der Stellung 1 ("gerade") verbleiben.

Empfehlung

Realisieren Sie eine SINIX-SINIX-Kopplung mittels UUCP über die SS97-Schnittstellen.

Bemerkung

Vorschlag zur Realisierung einer Verbindung mit SS97-Steckern:

PC-MX2 - PC-MX2

Auf beiden Seiten den Anschluß tty05 wählen und, wie im Abschnitt 6.2.3.2 Software beschrieben, installieren.

PC-MX2 - PC-X10

Auf der Seite PC-MX den Anschluß tty05 wählen; auf der Seite PC-X den Anschluß ST2 oder ST4 (V11-Anschlüsse) wählen und mittels des Kommandos `/etc/mknod` die entsprechende Einträge unter `/dev` erzeugen.

Lesen Sie bei: Kopplung PC-X10

6.2.3.2 Software

Inbetriebsname einer UUCP-Kopplung

Erstellen Sie folgende Datei

/etc/systemid

In diese Datei tragen Sie bitte nur eine Systemkennung ein. Eine Systemkennung besteht aus bis zu 8 Zeichen, beginnend mit einem Buchstaben, z.B. gruen1

Ändern Sie folgende Datei

/usr/lib/crontab (wie im Abschnitt 6.2.2.5 Sonstige Dateien beschrieben)

Tragen Sie pro Kopplung jeweils eine Zeile in folgende Dateien ein

- a) bei dem Master- und dem Slave-Rechner
/usr/lib/uucp/L.sys (siehe Abschnitt 6.2.2.5 Sonstige Dateien)
/usr/lib/uucp/L-devices (siehe Abschnitt 6.2.2.5 Sonstige Dateien)
- b) bei dem Master-Rechner
/usr/lib/crontab (siehe Abschnitt 6.2.2.5 Sonstige Dateien)
- c) bei dem Slave-Rechner
/etc/ttytype anschließend /etc/ttys (s. 6.2.2.5 Sonstige Dateien)

Richten Sie mit dem Kommando /etc/mknod einen Geräteeintrag ein

über den Sie die Kopplung betreiben wollen. Welche Anschlüsse Sie nie ansprechen können, entnehmen Sie bitte dem Schnittstellenhandbuch. Eine Beschreibung des /etc/mknod-Kommandos finden Sie im Buch SINIX Systemverwaltung. Welche Major- und Minornummer Sie angeben müssen, hängt von dem von Ihnen gewählten Steckplatz ab. Näheres entnehmen Sie bitte dem Schnittstellenhandbuch für Ihren SINIX-Computer. Die folgenden Anschlußbilder sind ein Auszug des Handbuches:

Kopplung eines PC-X10 mittels UUCP

Ablauf einer Installation

- 1) Installation von UUCP wie im Abschnitt 6.2.3 Installation beschrieben.
- 2) Erzeugen Sie einen Eintrag unter /dev mit den folgenden Kommandos
 - a) `cd /dev`
 - b)

Alternative 1 :	Anschluß an ST1 (v24-Stecker)
	<code>/etc/mknod v11 c 3 131</code>
Alternative 2 :	Anschluß an ST2 (SS97-Stecker)
	<code>/etc/mknod v11 c 3 1</code>
Alternative 3 :	Anschluß an ST3 (v24-Stecker)
	<code>/etc/mknod v11 c 3 130</code>
Alternative 4 :	Anschluß an ST4 (SS97-Stecker)
	<code>/etc/mknod v11 c 3 0</code>

(ST5 ist der Anschluß für eine MS-DOS Tastatur (PC-D)).
 - c) `chmod a+ w v11`
 - d) `stty raw 1200 > v11`

Achtung

Es sollten keine zwei Einträge mit der gleichen Major- und Minornummer existieren.

Daher:

Bitte kontrollieren Sie im Dateiverzeichnis /dev mittels des Kommandos `ls -l`, ob zwei Einträge mit gleicher Major- und Minornummer existieren.

Hinweise zur Übertragung auf andere UNIX-Rechner

Falls Sie das UUCP-Programm benutzen wollen, um von Ihrem Rechner aus eine Verbindung zu einem anderen Rechner mit einem beliebigen UNIX-Betriebssystem herzustellen, so beachten Sie bitte folgende Hinweise:

- 1) Aktivieren Sie Ihren SINIX-Computer wie im Abschnitt 6.2.3.2 Software beschrieben.

Hinweis

Einige Fremdrechner benötigen auf der SINIX-Seite (falls Ihr Rechner als Slave konfiguriert wird) in der Datei /etc/ttytype statt der Zeile

```
none V24 1200 1200 RW RW
```

den Eintrag

```
97801 V24 1200 1200 PO;RW;NL1;CR1 PO;EC;CM;F0;T0
```

(d.h.: "als 9780-Terminaltyp konfigurieren").

- 2) Aktivieren Sie UUCP auf Ihrem Fremdrechner und achten Sie bitte darauf, daß auf einem Slave-Rechner ein Login-Prozeß am Port vorhanden sein muß. Auf einem Master-Rechner ist ein solcher nur störend.
- 3) Sie müssen zur Übertragung die Hinweise bei den einzelnen Kommandos beachten.

Einschränkung

Die Ihnen übergebene UUCP-Software dürfen Sie nur auf dem Ihnen genannten SINIX-Computer mit der angegebenen Hardwarenummer einsetzen.

6.2.4 Kommandos

Die hier zusammengetragenen Kommandos sind für die Übertragung von SINIX- auf SINIX-Computer beschrieben.

Mit SINIX-Computern sind die Arbeitsplatzcomputer der SIEMENS AG gemeint, auf denen das Betriebssystem SINIX eingesetzt wird.

Zur Zeit stehen Ihnen folgende Geräte zur Verfügung:

PC - X Einplatz-Rechner (bisheriger Name: APC 9781)

PC - MX Mehrplatz-Rechner (bisheriger Name: APC 9780)

Bei jedem Kommando finden Sie die fettgedruckten Überschriften Hinweis und Dateien. Darunter sind folgende Informationen zusammengefaßt:

1) unter der Überschrift Hinweis

Falls Sie eine Kommunikation zu anderen UNIX-Rechnern installieren wollen, so beachten Sie bitte

- a) den Abschnitt Hinweise zur Übertragung auf andere UNIX-Rechner (siehe Vorseite) und
- b) die unter der Überschrift Hinweise angegebenen Besonderheiten bei dem jeweiligen Kommando.

2) unter der Überschrift Dateien

In diesem Abschnitt stehen die Dateien, die von dem Kommando betroffen sind. In runden Klammern steht das Dateiverzeichnis in dem Sie das Kommando finden.

6.2.4.1 mail, rmail

Zusätzliche Funktionen des Kommandos mail einschließlich rmail

Post senden

Format

mail rechner![rechner!...]benutzerkennung

rmail rechner![rechner!...]benutzerkennung

sonstige Formate von mail siehe Buch SINIX Kommandos

Beschreibung

Um einen Benutzer auf einem fernen Rechner anzugeben, setzen Sie vor die Benutzerkennung die Rechnerkennung mit einem Ausrufezeichen. Alles nach dem Ausrufezeichen wird von dem fernen Rechner interpretiert. Insbesondere, wenn die Empfängerangabe mehrere Ausrufezeichen enthält (so kann eine Reihe von Rechnern angegeben werden, durch die die Post gesandt wird bis zum endgültigen Ziel).

rmail erlaubt nur das Senden von Post. Der Aufruf !kommando wird von rmail nicht unterstützt.

Wenn sich ein Benutzer einloggt, wird er informiert, ob Post für ihn da ist. Die Nachricht "Sie haben neue Post" erhält der Benutzer mit dem nächsten Prompt, wenn er in seiner .profile Datei angegeben hat:

MAIL=/usr/spool/mail/benutzerkennung

Beispiel

Die Empfängerangabe a!b!cde bewirkt, daß die Post dem Benutzer b!cde auf dem Rechner a gesandt wird. Der Rechner a interpretiert dieses Ziel als eine Anforderung, die Post an den Benutzer cde auf dem Rechner b zu senden. Dies kann sinnvoll sein, wenn der sendende Rechner keinen Zugriff auf den Rechner b jedoch auf a und diese auf den Rechner b hat.

Hinweis

Allgemeines uucp-Kommando. Das Senden auf einen weiteren Rechner als dem nächstliegenden ist nicht getestet worden.

Dateien

(/bin/mail/ bin/rmail)

6.2.4.2 uulog

Abfrage der Protokollmeldungen

Format

uulog -schalter...

Schalter

s [rechner]	Ausgabe der Protokollmeldungen zu den Übertragungen mit dem angegebenen Rechner
u [benutzer]	Ausgabe der Protokollmeldungen zu den Übertragungen, die für den angegebenen Benutzer ausgeführt wurden.

Für beide Schalter gilt: Machen Sie keine Angaben für rechner bzw. benutzer, so erhalten Sie alle Protokollmeldungen ausgegeben.

Beschreibung

Das Kommando uulog gibt die Protokollmeldungen aus, die von uucp- und uux-Übertragungen in der Datei /usr/spool/uucp/LOGFILE eingetragen worden sind.

Achtung

Durch den Aufruf von uulog ohne Schalter tragen Sie zwar alle Protokolle in die Datei LOGFILE ein, erhalten jedoch an Ihrem Bildschirm keine Einträge der Datei ausgegeben. Dabei löschen Sie auch ältere Einträge. Daher sollten Sie die Angabe eines Schalters nie vergessen.

Hinweis

Auf einigen UNIX-Rechner heißt die angesprochene Datei `/usr/lib/uucp/LOGF.sys`

Dateien

`/usr/spool/uucp/LOGFILE`

`(/usr/bin/uulog)`

6.2.4.3 uucp

Übertragung von SINIX-Computer zu SINIX-Computer

Format

uucp[-schalter...] **quelle...** **ziel**

Schalter

d Alle notwendigen Dateiverzeichnisse für die Dateiübertragung erzeugen. (Die zur Übertragung notwendigen Dateiverzeichnisse werden auch ohne Angabe dieses Schalters eingerichtet.)

c Beim Übertragen wird aus der Quelldatei kopiert und nicht zuerst in das Spool-Dateiverzeichnis übertragen. (So wird auch verfahren, wenn Sie diesen Schalter nicht angeben.)

m Der Absender erhält eine Nachricht, wenn die Übertragung beendet ist.

nbenutzer Der angegebene Benutzer (auf dem fernen Rechner) erhält eine Nachricht, daß eine Datei gesandt wurde.

Zur Fehlersuche stehen folgende Schalter zur Verfügung:

r Der Auftrag wird angenommen, jedoch das Programm uucico (zum Übertragen) wird nicht gestartet.

sdateiverzeichnis Anstatt des Spool-Dateiverzeichnisses wird das angegebene Dateiverzeichnis benutzt. (Die Zugriffsberechtigung für dieses Dateiverzeichnis muß gegeben sein.)

xn Die angegebene Zahl *n* gibt die Ausführlichkeit des Fehlerprotokolls an. Es kann eine der folgenden Zahlen angegeben werden:

1 - 3 siehe Beispiel 1 (siehe Abschnitt 6.2.4.3 uucp)

4 - 7 siehe Beispiel 2 (siehe Abschnitt 6.2.4.3 uucp)

8 - siehe Beispiel 3 (siehe Abschnitt 6.2.4.3 uucp)

quelle

quelle kann ein Dateiname auf Ihrem Rechner sein oder die Form rechner!dateiname haben.

rechner

Die Rechnerkennung muß aus der Liste der dem UUCP bekannten Rechner sein.

dateiname

Der Dateiname kann wie folgt aussehen:

- 1) voller Pfadname oder
- 2) ein Pfadname, beginnend mit ~benutzer, wobei benutzer der Login-Name des Benutzers auf dem angegebenen Rechner ist und dann von UUCP durch das Login-Dateiverzeichnis ersetzt wird oder
- 3) ein Pfadname, beginnend mit ~/benutzer, wobei benutzer der Login-Name des Benutzers auf dem angegebenen Rechner ist und dann von UUCP durch /usr/spool/uucp/uucppublic/benutzer ersetzt wird oder
- 4) eine beliebige Datei des aktuellen Dateiverzeichnisses.

ziel

ziel kann ein Dateiname auf Ihrem Rechner sein oder kann die Form rechner!dateiname haben.

rechner

Die Rechnerkennung muß aus der Liste der dem UUCP bekannten Rechnern sein.

dateiname

Der Dateiname kann wie folgt aussehen:

- 1) voller Pfadname oder
- 2) ein Pfadname, beginnend mit ~benutzer, wobei benutzer der Login-Name des Benutzers auf dem angegebenen Rechner ist und dann von UUCP durch das Login-Dateiverzeichnis ersetzt wird oder
- 3) ein Pfadname, beginnend mit ~/benutzer, wobei benutzer der Login-Name des Benutzers auf dem angegebenen Rechner ist und dann von UUCP durch /usr/spool/uucp/uucppublic/benutzer ersetzt wird oder

- 4) eine beliebige Datei des aktuellen Dateiverzeichnisses oder
- 5) Statt eines Dateinamens kann das Ziel wie folgt aussehen: rechner!~uucp. Die Quelldatei wird dann in dem Dateiverzeichnis /usr/spool/uucp/uucppublic abgelegt. Falls das Ergebnis ein fehlerhafter Pfadname für den fernen Rechner ist, geht die Übertragung fehl. Falls das Ziel ein Dateiverzeichnis ist, wird der Dateiname aus dem Ziel-Dateiverzeichnis und dem letzten Teil des Quell-Dateinamens gebildet.

Für quelle und ziel gilt:

Die Shell-Metazeichen ? * [], die in Pfadnamen auftauchen können, werden von dem angesprochenen Rechner aufgeschlüsselt.

Modus

UUCP behält die Ausführungserlaubnis einer Datei (falls vergeben) bei und vergibt die Lese- und Schreiberlaubnis.

Wenn Sie an ein fremdes Dateiverzeichnis Dateien senden, so müssen Sie die Schreibberechtigung für dieses Dateiverzeichnis haben. Eigentümer der übertragenen Datei in dem fremden Dateiverzeichnis ist dann uucp.

Dateien

/usr/spool/uucp/...
/usr/spool/uucp/uucppublic/...
/usr/lib/uucp/...

(/usr/bin/uucp)

Warnung

Das Gebiet der in dem fernen Rechner zugänglichen Dateien kann (aus Sicherheitsgründen, normalerweise sollte) streng eingeschränkt sein. Meist wird es Ihnen nicht sehr leicht möglich sein, Dateien per Pfadnamen zu holen. Bitten Sie eine verantwortliche Person an dem fernen Rechner, sie Ihnen zu schicken. Aus denselben Gründen wird es Ihnen nicht möglich sein, Dateien auf willkürliche Pfadnamen zu senden. Wie beschrieben sind die allgemein zugänglichen Dateien, diejenigen, die mit `/usr/spool/uucp/uucppublic` (äquivalent zu `~remuucp` oder nur `~uucp`) beginnen.

Einschränkungen

Alle Dateien, die von UUCP empfangen werden, gehören UUCP. Der Schalter `m` ist nur aktiv, wenn Sie Dateien senden oder eine einzelne Datei erhalten. (Ankommende Dateien angegeben mit den Shell-Metazeichen `? * [ ]` aktivieren den Schalter `m` nicht.)

Hinweis

Einige UNIX-Rechner können beim Schalter `x` nur die Ziffern 1 - 4 interpretieren.

Der Schalter `n` ist eine SINIX-Besonderheit.

Beispiele

Die Beispiele 1 - 3 sind Debug-Beispiele. Sie erhalten mit dem Schalter x nur ein Protokoll für das Übertragen in das Spool-Dateiverzeichnis. Ein ausführliches Debug-Beispiel ist beim Kommando uucico im Abschnitt 6.2.4.4 uucico beschrieben.

zum Beispiel 1 - 3

abgesetztes Kommando: uucp -xn pix gruen!~huma Sie erhalten nun einen Debug für die Einrichtung der Dateien in /usr/spool/uucp, nicht für die Übertragung.

(hm-> ist das in der .profile-Datei hinterlegte Prompt)

Beispiel 1 n = 1 - 3

nichts

Beispiel 2 n = 4 - 7

2 Leerzeilen

** START **

UID 20,USER peter,PATH /usr/peter

file1 - pix

send file - 2

file - C.grünn0025

xcp calledhm->

Beispiel 3 n = 8 -

2 Leerzeilen

** START **

UID 20,USER peter,PATH /usr/peter

file - pix

send file - 2

file - C.grünn0034

xcp calledchild f1 ok f2 ok stat=0

hm->

siehe auch

uuto, uupick, uux insbesondere bei debug siehe uucico

6.2.4.4 uucico

Übertragungsprogramm

Format

uucico[-schalter]

Schalter

r1 uucico wird im MASTER-Modus gestartet. Dies wird benutzt, wenn uucico von einem Programm oder der "cron" Shell gestartet wird.

srechner Nur für den angegebenen Rechner werden Aufträge abgearbeitet. Falls s angegeben ist, wird sogar dann ein Ruf an den angegebenen Rechner gesandt, wenn keine Arbeit für den Rechner im Spool-Dateiverzeichnis vorliegt. Dort anstehende Arbeit kann dann erledigt werden. Dies ist nützlich für anrufende Rechner, die nicht die Hardware haben, um die Verbindungen selbst aufzubauen.

Zur Fehlersuche stehen folgende Schalter zur Verfügung:

ddateiverzeichnis Anstatt des Spool-Dateiverzeichnisses wird das angegebene benutzt.

xn Die Zahl n gibt die Ausführlichkeit des Fehlerprotokolles an. Es kann eine der folgenden Zahlen angegeben werden:

4 siehe Debugbeispiel im Abschnitt 6.2.4.4 uucico

Beschreibung

siehe Abschnitt 6.2.2.4 Ablauf eines uucico-Aufrufs

Hinweis

Einige UNIX-Rechner können beim Schalter x nur die Ziffern 1 - 4 interpretieren.

Datei

(/usr/lib/uucp/uucico)

Einschränkung

Beim Debug-Aufruf kann sich der Rechner aufhängen.

Bitte richten Sie vor dem Aufruf die Datei /usr/spool/uucp/AUDIT ein mit dem Kommando: touch /usr/spool/uucp/AUDIT

Debugbeispiel

Um einen ausführlichen Debug zu erhalten empfiehlt sich das Vorgehen:

- 1) UUCP so starten, daß es nur die Dateien in den Hintergrund spoolt und dann
- 2) uucico explizit als Benutzer starten
Prozedur testerl:
set -x
uucp -r -x4 kom.uucp gruen! ~ huma/testempf
uucico -r1 -x4 -sgrün
(SS97-Kopplung über tty05;
Aufruf an den MASTER-Rechner rot;
Übertragung zum SLAVE-Rechner gruen)

Nach dem Starten der Prozedur doit erschien am Bildschirm:

uucp -r -x4 kom.uucp gruen!~huma/testempf

```

- 2 Leerzeilen
  ** START **
debug  UID 20, USER peter,PATH /usr/peter
      file1 - kom.uucp
uucp    send file - 2
      file - C.grünn0070
- xcp calleduucico -r1 -x4 -sgrün
- gdial called
  finds called
  getto called
  call no.tty05 for sys gruen Speed: want 1200 use 11 ps 6
  login called
  start in send mode
    sent <remuucp><nl>>wanted <(null)> REMOTE returned <r>—ok
    sent <<nl>>msg-ROK
      Rmtname gruen, my Role MASTER, Spool - /usr/spool/uucp
      Ifn - 3, Ofn - 3, Loginuser - peter
      rmesg - 'P' got Pg
      wmesg 'U'g
      Proto started g
debug  protocol g
      *** TOP *** - role=1, wrktype S, USER - peter
      wrktype -S, fileno - 5
uucico  wmesg 'S' /usr/peter/testu/kom.u
      rmesg - S got SY
        PROCESS: msg - SY
      SNDFILE:
      sent data 7572 bytes 71 secs
      rmesg - 'C' got CY
        PROCESS: msg - CY
      RQSTCMPT:
      *** TOP *** - role=1, wmesg 'H'
      rmesg - 'H' got HY
        PROCESS: msg - HY
      HUP:
      ret from cntrl - 0
- send oo 0,exit code 0

```

6.2.4.5 uux

Ausführen von Kommandos auf einem anderen Rechner

Format

uux[-| -schalter] kommandofolge

Das Zeichen "-" bewirkt, daß die Standardeingabe für das uux-Kommando die Standardeingabe für die Kommandofolge ist. (siehe Beispiel 3)

Schalter

Zur Fehlersuche stehen folgende Schalter zur Verfügung:

r	Der Auftrag wird angenommen (also die zur Übertragung notwendigen Dateien angelegt), jedoch das Programm uucico (zum Übertragen) wird nicht gestartet.
xn	Die angegebene Zahl n gibt die Ausführlichkeit des Fehlerprotokolls an. Es kann eine der folgenden Zahlen angegeben werden:

4 siehe Beispiel im Abschnitt 6.2.4.5 uux

Beschreibung

Rechner, führt ein Kommando auf einem bestimmten Rechner aus und sendet dann die Standardausgabe auf einen bestimmten Rechner.

Die Kommandofolge besteht aus einem oder mehreren Argumenten, die wie eine Shell-Kommandozeile aussehen, außer daß das Kommando und die Dateinamen ein vorangestelltes rechner! haben, wobei rechner ein dem UUCP bekannter Rechnernamen ist. Ein nicht angegebener Rechnernamen wird als der des lokalen Rechners interpretiert.

Dateinamen können wie folgt angegeben werden:

- 1) als voller Pfadname oder
- 2) als Pfadname mit vorangestelltem ~benutzer, wobei benutzer ein Loginname auf dem bestimmten Rechner ist und durch das jeweilige Logindateiverzeichnis ersetzt wird oder
- 3) den Namen einer beliebigen Datei des aktuellen Dateiverzeichnisses.

Jedes spezielle Shell-Sonderzeichen, sowie < > ; | sollte mit Anführungszeichen versehen werden und zwar entweder die gesamte Kommandofolge oder jedes Sonderzeichen als einzelnes Argument. (siehe Beispiel 1)

uux wird versuchen alle Dateien ins ausführende System zu bekommen. Für Dateien, die Ausgabe-Dateien sind, muß der Dateiname durch runde Klammern eingeschlossen sein. (siehe Beispiel 2)

Beispiele

- 1) Das Kommando
`uux '!diff usg!/usr/dan/f1 pwba!/a4/dan/f1 > !f1.diff'`
wird die f1-Dateien von dem usg- und dem pwba-Rechner holen, ein diff Kommando ausführen und das Ergebnis in der Datei f1.diff in dem lokalen Dateiverzeichnis ablegen.
- 2) Das Kommando `uux a!uucp b!/usr/file \ (c!/usr/file\)` wird ein uucp-Kommando zum Rechner mit der Kennung a senden, um die Datei /usr/file aus dem Rechner mit der Kennung b zu holen und sie zum Rechner mit der Kennung c zu übertragen.
- 3) Das Kommando `pr abc | uux - usg!lpr` etabliert die Ausgabe von "pr abc" als Standardeingabe für ein lpr-Kommando, das auf einem fernen Rechner "usg" ausgeführt wird. Dieses Beispiel zeigt: Sie können dem Drucker einem fernen Rechner alle Schalter mitgeben (bis auf -q).

Dateien

```
/usr/spool/uucp
/usr/lib/uucp/*
(/usr/bin/uux)
```

Einschränkungen

- Zur Zeit können nur folgende Kommandos auf SINIX-Computern mittels uux ausgeführt werden:
mail rmail who lpr
- Nur das erste Kommando einer Shell-Pipeline darf einen Rechnernamen tragen. Alle anderen Kommandos werden auf dem Rechner des ersten Kommandos ausgeführt.
- Der Gebrauch des Shell-Metazeichens * wird wahrscheinlich nicht das tun, was Sie sich davon erwarten. Die Shell-Sonderzeichen << und >> sind nicht implementiert.

Hinweis

- Beachten Sie, daß, aus Sicherheitsgründen, viele UNIX-Versionen die Liste der ausführbaren Kommandos zugunsten von nachfolgenden uux-Kommandos begrenzen. Einige ermöglichen fast nur den Empfang von Post mittels uux.
- Einige UNIX-Rechner können beim Schalter x nur die Ziffern 1-4 interpretieren.

Beispiel

Zum Schalter x, x=4

Abgesetztes Kommando : pr protokoll | uux -x4 - gruen!lpr

2 Leerzeilen

** START **

arg - gruen!lpr:arg - gruen!lpr

file - D.rotX0075

file - D.rotT0076

xsys gruen

file - D.grünB0077

prm - gruen!lpr

s - gruen, r -lpr, ret - 1

file - C.grünA0078

exit code 0

Bemerkung:
"zweiseitige Datei"

siehe auch:

uuclean, uucp

6.2.4.6 uuxqt

Programm zum Übertragen der mit uux angegebenen Kommandofolge

Beschreibung

Das Programm uuxqt wird benutzt, um ausführbare Dateien, die von dem Programm uux generiert wurden, zu übertragen. Das Programm kann entweder von dem uucico-Programm oder dem uux-Programm gestartet werden. Das Programm durchsucht das Spool-Dateiverzeichnis nach auszuführenden Dateien (Präfix X). Jede wird überprüft, um zu sehen, ob alle benötigten Dateien verfügbar sind und, falls dies erfüllt ist, ob die Kommandozeile oder Sendezeile abgearbeitet ist.

Die auszuführende Datei ist im Kapitel "Beginn einer uux-Übertragung" beschrieben.

Die Ausführung wird vervollständigt, indem ein sh -c der Kommandozeile ausgeführt wird, nachdem die verwendete Standardeingabe und Standardausgabe geöffnet wurden. Falls eine Standardausgabe festgelegt wurde, wird das Programm ein Sende-Kommando erzeugen oder die Ausgabedatei wie angegeben kopieren.

Dateien

y (/usr/lib/uucp/uuxqt)

6.2.4.7 uuclean

Aufräumen des UUCP-spool-Dateiverzeichnisses

Format

uuclean[-schalter ...]

Schalter

ddateverzeichnis

Löschen des angegebenen Dateiverzeichnisses anstatt des Spool-Dateiverzeichnisses.

ppräfix

uuclean bearbeitet alle Dateien mit dem angegebenen Präfix im Dateinamen. Bis zu zehn p-Argumente können angegeben werden. Geben Sie ein p ohne folgendes Präfix an, so werden alle Dateien gelöscht, die älter als eine bestimmte Zeit sind.

nstunden

Dateien, die älter als die angegebene Stundenzahl sind, werden gelöscht, falls der Präfix-Test erfolgreich war. Falls Sie keine Angabe machen, ist der Wert 72 Stunden.

m

Nachricht an Eigentümer der Dateien, wenn sie gelöscht werden.

Zur Fehlersuche steht Ihnen folgender Schalter zur Verfügung:

xn

Die Zahl n gibt die Ausführlichkeit des Fehlerprotokolls an.

Beschreibung

uuclean durchsucht das Spool-Dateiverzeichnis nach Dateien mit dem angegebenen Präfix und löscht alle jene, die älter als eine bestimmte Anzahl von Stunden sind.

Dieses Programm wird normalerweise von cron gestartet.

Dateien

/usr/lib/uucp

Dateiverzeichnis mit den Kommandos, die intern von uuclean benutzt werden.

/usr/spool/uucp

Spool-Dateiverzeichnis

(/usr/lib/uucp/uuclean)

siehe auch:

uucp, uux und

uuhour, uuday und

Einträge in der Datei `/usr/lib/crontab`

Hier sind noch einige Hinweise gegeben:

Literatur

- SINIX Programmer's Manual by Michael Santifaller
- A Dial-Up Network of UNIX Systems by D. A. Nowitz M. E. Lesk (Aufsatz Nr.35 im XENIX-Manual /*/)
- UUCP Implementation Description by D. A. Nowitz (Aufsatz Nr.36 im XENIX-Manual /*/)

/*/ XENIX OS

PROGRAMMER'S MANUAL VOLUME 2A

Einschränkungen

- Da bei der Installation nur eine Rechnerkennung in der Datei /etc/systemid hinterlegt werden kann, ist der doppelte Verbindungsaufbau (1x MASTER-SLAVE und 1x SLAVE-MASTER) über zwei Standleitungen nicht möglich.
- Zur Zeit werden nur Standleitungen unterstützt.
(auf Wunsch: Akkustikkoppler)
- Zur Zeit können nur folgende Übertragungsgeschwindigkeiten gewählt werden:

300 Baud und 1200 Baud

Empfehlung

Auf der SLAVE-Seite in der Datei L.sys den Zeiteintrag von 0 in NONE oder NEVER ändern.

Änderung der Übertragungsgeschwindigkeit

Ändern Sie die Einträge zur Übertragungsgeschwindigkeit in den Dateien:

L-devices

L.sys

/etc/ttytype

und mit dem stty-Kommando den Anschluß, über den Sie UUCP betreiben.

6.2.5 Was tun, wenn ...

- Sie wollen testen, ob Daten über die Leitung geschickt werden können: Übertragung von Rechner 1 auf Rechner 2

Rechner 2:

```
stty 1200 -echo cbreak > /dev/anschluß  
cat -u /dev/anschluß
```

Rechner 1:

```
stty 1200 -echo cbreak > /dev/anschluß  
cat testdatei > /dev/anschluß, wobei /dev/anschluß der jeweilige  
Geräteeintrag ist, den Sie mit dem Kommando /etc/mknod eingetragen  
haben.
```

- Wenn Sie wissen wollen, ob die Übertragung fehlerfrei beendet worden ist:
Lesen Sie mit dem Kommando `uulog -s` die Protokolldatei, durchsuchen das Dateiverzeichnis `/usr/spool/uucp` nach temporären Dateien und kontrollieren Ihre übertragene Datei.
- Wenn die Datei übertragen worden ist, jedoch nicht im Zieldateiverzeichnis eingetragen ist. (Datei `TM.*.*` existiert)
Überprüfen Sie die Zugriffsberechtigungen des Zieldateiverzeichnisses und der darüber angeordneten Dateiverzeichnisse auf Schreibberechtigungen für alle anderen, da im Normalfall der Empfänger nicht Mitglied der Gruppe `uucp` ist.
- Wenn Dateien von `uucp` überschrieben worden sind, obwohl die Schreibberechtigungen nicht vergeben waren.
Überprüfen Sie den Eigentümer der Dateien:

```
/usr/bin/uucp  
/usr/bin/uulog  
/usr/bin/uux  
/usr/lib/uucp/uucico  
/usr/lib/uucp/uuclean  
/usr/lib/uucp/uuxqt
```

Der Eigentümer sollte `uucp` sein und nicht `root`.

- Wenn Sie das Kopieren nur noch in ein bestimmtes Zieldateiverzeichnis erlauben wollen.
Sie können mit einem veränderten Eintrag in der Datei `/usr/lib/uucp/USERFILE` dafür sorgen, daß von einem fernen Rechner aus nur noch

UUCP

in das Dateiverzeichnis `/usr/spool/uucp/uucppublic` übertragen werden kann. Jeder Empfänger muß sich dann aus dem lokalen Dateiverzeichnis `uucppublic` die ihn interessierende Datei in sein Dateiverzeichnis kopieren oder mit dem Kommando `mv` bewegen.

Hinweis

Neuer Eintrag in `/usr/lib/uucp/USERFILE`:
`remuucp, /usr/spool/uucp/uucppublic, /usr/spool/uucp/uucppublic`

Anhang

Inhalt

Kommandolisten	A-2
Shell-Übersichten	A-4
SINIX-Arbeitsumgebung	A-12
Ausdrücke	A-13
Reguläre Ausdrücke	A-14
Erweiterte reguläre Ausdrücke	A-16
ASCII-Zeichen	A-17

Kommandolisten

Alle Kommandos (alphabetisch mit Suffix (1) und (1D)
Shell-Anweisungen und -Funktionen sind ohne Suffix enthalten)

admin(1D)	ar(1)	at(1)	att(1)
awk(1)	banner(1)	basename(1)	batch(1)
bc(1)	bdiff(1D)	bfs(1)	break
bs(1)	cal(1)	calendar(1)	case
cat(1)	cb(1D)	cc(1D)	cd(1)
cdc(1D)	ced(1)	cflow(1D)	chgrp(1)
chmod(1)	chown(1)	chroot(1)	cmp(1)
col(1)	comb(1D)	comm(1)	continue
cp(1)	cpio(1)	crontab(1)	csplit(1)
cut(1)	cxref(1D)	date(1)	dc(1)
dd(1)	delta(1D)	destroy(1)	df(1)
diff(1)	diff3(1)	dircmp(1)	dirname(1)
du(1)	echo(1)	ed(1)	edit(1)
egrep(1)	env(1)	eval	ex(1)
exec	exit	export	expr(1)
factor(1)	false(1)	fgrep(1)	file(1)
find(1)	for	get(1D)	getopt(1)
grep(1)	id(1)	if	ipcrm(1)
ipcs(1)	join(1)	kill(1)	ld(1D)
lex(1D)	line(1)	lint(1D)	ln(1)
login(1)	logname(1)	lorder(1D)	lpr(1)
ls(1)	m4(1D)	machid(1)	mail(1)
mailx(1)	make(1D)	man(1)	mesg(1)
mkdir(1)	mknod(1)	mv(1)	newform(1)
newgrp(1)	nice(1)	nl(1)	nm(1D)
nohup(1)	ns32000(1)	od(1)	pack(1)
passwd(1)	paste(1)	pcat(1)	pg(1)
pr(1)	prof(1D)	prs(1D)	ps(1)
Punkt	pwd(1)	ranlib(1D)	read
readonly	red(1)	regcmp(1)	rm(1)
rm(1D)	rmdir(1)	sact(1D)	sccsdiff(1D)
sdiff(1)	sed(1)	set	sh(1)
shift	sie(1)	size(1D)	sleep(1)
sort(1)	spell(1)	split(1)	strip(1D)
stty(1)	su(1)	sum(1)	sync(1)
tail(1)	tar(1)	tee(1)	test(1)
tic(1D)	time(1)	times	touch(1)
tput(1)	tr(1)	trap	true(1)
tsort(1D)	tty(1)	umask(1)	uname(1)
unset(1D)	uniq(1)	units(1)	universe(1)
unpack(1)	until	val(1D)	vc(1D)
vi(1)	wait(1)	wall(1)	wc(1)
what(1D)	while	who(1)	write(1)
xargs(1)	yacc(1D)		

Kommandos (alphabetisch mit (1))

ar(1)	at(1)	att(1)	awk(1)
banner(1)	basename(1)	batch(1)	bc(1)
bfs(1)	bs(1)	cal(1)	calendar(1)
cat(1)	cd(1)	ced(1)	chgrp(1)
chmod(1)	chown(1)	chroot(1)	cmp(1)
col(1)	comm(1)	cp(1)	cpio(1)
crontab(1)	csplit(1)	cut(1)	date(1)
dc(1)	dd(1)	destroy(1)	df(1)
diff(1)	diff3(1)	dircmp(1)	dirname(1)
du(1)	echo(1)	ed(1)	edit(1)
egrep(1)	env(1)	ex(1)	expr(1)
factor(1)	false(1)	fgrep(1)	file(1)
find(1)	getopt(1)	grep(1)	id(1)
ipcrm(1)	ipcs(1)	join(1)	kill(1)
line(1)	ln(1)	login(1)	logname(1)
lpr(1)	ls(1)	machid(1)	mail(1)
mailx(1)	man(1)	msg(1)	mkdir(1)
mknod(1)	mv(1)	newform(1)	newgrp(1)
nice(1)	nl(1)	nohup(1)	ns32000(1)
od(1)	pack(1)	passwd(1)	paste(1)
pcat(1)	pg(1)	pr(1)	ps(1)
pwd(1)	red(1)	regcmp(1)	rm(1)
rmdir(1)	sdiff(1)	sed(1)	sh(1)
sie(1)	sleep(1)	sort(1)	spell(1)
split(1)	stty(1)	su(1)	sum(1)
sync(1)	tail(1)	tar(1)	tee(1)
test(1)	time(1)	touch(1)	tput(1)
tr(1)	true(1)	tty(1)	umask(1)
uname(1)	uniq(1)	units(1)	universe(1)
unpack(1)	vi(1)	wait(1)	wall(1)
wc(1)	who(1)	write(1)	xargs(1)

Kommandos (alphabetisch mit (1D))

admin(1D)	bdiff(1D)	cb(1D)	cc(1D)
cdc(1D)	cflow(1D)	comb(1D)	cxref(1D)
delta(1D)	get(1D)	ld(1D)	lex(1D)
lint(1D)	lorder(1D)	m4(1D)	make(1D)
nm(1D)	prof(1D)	prs(1D)	ranlib(1D)
rmel(1D)	sact(1D)	sccsdiff(1D)	size(1D)
strip(1D)	tic(1D)	tsort(1D)	unget(1D)
val(1D)	vc(1D)	what(1D)	yacc(1D)

Shell-Übersichten

Kommandos, Anweisungen und Funktionen, die die Shell selbst interpretiert

break	case	cd	continue
eval	exec	exit	export
for	if	newgrp(1)	read
readonly	set	sh(1)	shift
times	trap	umask(1)	until
wait	while	. Punkt	:
#	()	{ }	[]

Ablaufanweisungen für Shell-Prozeduren

break	(Schleifen steuern)
case	(Abfragen und Verzweigen)
continue	(Schleife erneut ausführen)
for	(Liste in Schleifen abarbeiten)
if	(Abfragen und Kommando ausführen)
while	(Schleife mit Abbruchbedingung)

Kurz-Übersicht - Sonderzeichen der Shell

Zeichen	Bedeutung
>	<i>kdo >datei:</i> Umlenkung, <i>kdo</i> -Standardausgabe auf <i>datei</i>
>>	<i>kdo >>datei:</i> Anhängen, <i>kdo</i> -Standardausgabe an <i>datei</i>
<	<i>kdo <datei:</i> Umlenkung, <i>kdo</i> -Standardeingabe von <i>datei</i>
<<wort	<i>kdo <<wort:</i> Die Standardeingabe besteht aus den nächsten Zeilen bis zur ersten Zeile, die nur <i>wort</i> enthält.
*	Textmuster, beliebige Zeichenkette für Dateiname
?	Textmuster, ein beliebiges Zeichen
[...]	Textmuster, je ein Zeichen aus Angabe-Menge; [3c9] [a-z] etc.
[!...]	Textmuster, je nicht ein Zeichen aus der Menge [von-bis] etc.
;	Kommando-Begrenzer
&	<i>kdo&</i> startet <i>kdo</i> als Hintergrundprozeß
`kdo`	führt <i>kdo(s)</i> aus und ersetzt es durch seine Standardausgabe
(kdo)	führt Kommando(s) <i>kdo</i> in einer Subshell aus
{_kdo;}	führt <i>kdo(s)</i> in akt. Shell aus, faßt Std-Ausgaben zusammen
\$1,\$2,...	Stellungsparameter (Argumente) einer Shell-Prozedur
\$var	Wert einer Shellvariablen <i>var</i>
\${var}	Wie <i>\$var</i> ; wenn der Var-Name im Kontext nicht eindeutig ist
\	Hebt Sonderbedeutung des nachfolgenden Zeichens auf
'...'	Keine Sonderbedeutung aller eingeschlossenen Zeichen
"..."	Shell erkennt nur \$, `...`, \ und interpretiert Zeichenkette
#	Leitet Kommentar ein, (Kommentar-Ende ist Zeilenende)
var=wert	Variable erhält Wert
&&	<i>kdo1 && kdo2</i> <i>kdo1</i> erfolgreich: <i>kdo2</i> -Ausführung
	<i>kdo1 kdo2</i> <i>kdo1</i> nicht erfolgreich: <i>kdo2</i> -Ausführung

Sonderzeichen der Shell

Die Sonderzeichen der Shell, haben für diese eine besondere Bedeutung.

Steuerzeichen

- ;** Kommandos, die Sie in einer Zeile durch ein ; (Strichpunkt) getrennt eingeben, werden nacheinander ausgeführt, z.B. `cd dir; pwd`
- &** Wenn Sie eine Kommandoeingabe mit dem Zeichen & beenden, wird das Kommando als Hintergrundprozeß gestartet; z.B. `cc pgm.c&`
- |** Pipezeichen. Kommandos, die Sie jeweils durch ein Pipezeichen getrennt eingeben, bilden eine Pipeline: Die Standard-Ausgabe des Kommandos vor einem Pipezeichen bildet die Standard-Eingabe für das Kommando hinter dem Pipezeichen.
Beispiel: `cat datei1 datei2 | grep muenchen | sort`
- &&** Trennen Sie zwei Kommandolisten durch &&, dann wird die zweite Kommandoliste nur dann ausgeführt, wenn die erste Kommandoliste den Endestatus 0 liefert.
Beispiel: `cc prog.c && a.out`
- ||** Trennen Sie zwei Kommandolisten durch ||, dann wird die zweite Kommandoliste nur dann ausgeführt, wenn die erste Kommandoliste einen Endestatus ungleich 0 liefert.
- (...)** Für die Kommandoliste zwischen den runden Klammern wird eine Subshell erzeugt.
Beispiel: `name=nepomuk; (name=edel; echo $name); echo $name`
- {...;}** Die Kommandos, die in den geschweiften Klammern stehen, werden zusammengefaßt. Auf die Klammern folgende Pipeverbindungen oder Ein-/Ausgabeumlenkungen gelten für alle Kommandos, die in den Klammern stehen.
Beachten Sie das Leerzeichen nach der öffnenden Klammer! Vor der schließenden Klammer muß ein Strichpunkt oder ein Neue-Zeile-Zeichen stehen.
Beispiel: `{ ls -l; ls -l . . ; } | grep meier`

name() {...;}

Der Funktion *name* wird die Kommandofolge zugeordnet, die in den geschweiften Klammern steht. Soll die Kommandofolge ausgeführt werden, müssen Sie dann nur noch *name* eingeben.

Beachten Sie das Leerzeichen nach der öffnenden geschweiften Klammer! Vor der schließenden geschweiften Klammer muß ein Strichpunkt oder das Zeichen für Neue-Zeile stehen. Geben Sie nur ein einziges Kommando an, dann können Sie die geschweiften Klammern und den Strichpunkt weglassen.

Beispiel: wo() {who am i; pwd; }

kommentar

Kommentarzeichen. Die Shell ignoriert alle Zeichen, die zwischen dem Kommentarzeichen und dem nächsten Neue-Zeile-Zeichen stehen.

Neue-Zeile-Zeichen

Mit einem Neue-Zeile-Zeichen schließen Sie eine Eingabe ab.

Leerzeichen Tabulatorzeichen

Mit Leer- und Tabulatorzeichen trennen Sie beim Aufruf eines Kommandos den Kommandonamen und die übergebenen Argumente voneinander.

Generierung von Dateinamen

* steht für eine beliebige Zeichenfolge, auch für die leere Zeichenfolge

? steht für ein beliebiges einzelnes Zeichen

[...] steht für ein beliebiges der in den Klammern eingeschlossenen Zeichen. Zwei Zeichen, die durch - voneinander getrennt sind, stehen für ein beliebiges Zeichen, das in der ASCII-Tabelle zwischen diesen beiden Zeichen steht, z.B. [a-dw-x].

[!...] steht für ein beliebiges Zeichen, das nicht in den Klammern steht. Auch hier können Sie einen Bereich angeben, z.B. [!a-z].

Entwertung von Sonderzeichen und Apostrophierung

- ** Der Gegenschrägstrich \ entwertet das darauffolgende Sonderzeichen. Das Sonderzeichen hat dann keine besondere Bedeutung für die Shell mehr.
Die Shell ignoriert das Zeichenpaar \< Neue Zeile > .
Beispiel: `echo \ $name \ $adresse`
liefert: `$name $adresse`
- '...'** Zeichen, die zwischen Hochkommas stehen, haben keine besondere Bedeutung.
Beispiel: `echo ' $name $adresse '`
liefert: `$name $adresse`
- "..."** Steht eine Zeichenfolge zwischen Anführungszeichen, dann behalten nur die Sonderzeichen \$, ` (Accent grave) und \ (Gegenschrägstrich) ihre besondere Bedeutung. Die übrigen Sonderzeichen werden entwertet. Vorhandene Variablen (Stellungs- oder Kennwortparameter) werden ersetzt, Kommandos zwischen Accents graves `...` werden ausgewertet.
- `...`** Steht eine Zeichenfolge zwischen Accents graves, dann interpretiert die Shell diese Zeichenfolge als Kommando, führt das Kommando aus und setzt die Ausgabe des Kommandos an die Stelle dieser Zeichenfolge. Vorhandene Variablen (Stellungs- oder Kennwortparameter) werden ersetzt.

Ein-/Ausgabe-Umlenkung

- >

datei

Die Standard-Ausgabe soll die Datei *datei* sein. Existiert die Datei bereits, so wird ihr Inhalt überschrieben.
- > >

datei

Die Standard-Ausgabe wird an die Datei *datei* angehängt. Existiert die Datei noch nicht, wird sie neu angelegt.
- 2>

datei

Die Standard-Fehlerausgabe soll in die Datei *datei* umgelenkt werden.
- > &n

Die Standard-Ausgabe soll in die Datei umgelenkt werden, die zur Dateikennzahl *n* gehört.
- > &-

Die Standard-Ausgabe-Datei wird geschlossen.
- <

datei

Die Standard-Eingabe soll die Datei *datei* sein.
- < <

wort

Die Shell-Prozedur wird bis zu einer Zeile, die nur aus *wort* besteht, bzw. bis zum Dateiende-Zeichen gelesen. Das so entstandene Dokument ("Here Document") wird zur Standard-Eingabe.

Ist ein Zeichen in *wort* apostrophiert, dann werden die im Dokument enthaltenen Zeichen nicht interpretiert. Andernfalls führt die Shell eine Parameter- und Kommandosubstitution durch und ignoriert (nicht entwertete) Neue-Zeile-Zeichen; in diesem Fall müssen Sie den Gegenschrägstrich \ verwenden, um die Zeichen \ (Gegenschrägstrich), \$, ` (Accent grave) und das erste Zeichen von *wort* zu entwerfen.
- < <-

wort

Folgt auf ein < < ein -, dann werden in *wort* und allen Zeilen des Here Documents alle führenden Tabulatorzeichen entfernt.
- < &n

Die Standard-Eingabe soll die Datei sein, die zur Dateikennzahl *n* gehört.
- < &-

Die Standard-Eingabe-Datei wird geschlossen.

Shell-Variablen (Parameter)

`${name}` Für `${name}` wird der Wert der Variablen *name* (falls vorhanden) eingesetzt. Die geschweiften Klammern sind nur dann notwendig, wenn auf *name* ein oder mehrere Buchstaben, Ziffern oder Unterstriche folgen, die nicht als Teil des Variablennamens interpretiert werden soll. Durch *name=wert* bzw. *name="wert"* weisen Sie der Variablen *name* einen Wert zu.

`${name:-wort}` Besitzt die Variable *name* einen nichtleeren Wert, dann wird der Ausdruck durch diesen Wert ersetzt. Ist die Variable *name* nicht gesetzt oder hat sie einen leeren Wert, dann wird *wort* eingesetzt.

`${name:=wort}` Ist die Variable *name* nicht gesetzt oder hat sie einen leeren Wert, so wird ihr der Wert *wort* zugewiesen. Danach wird der Ausdruck durch den Wert von *name* ersetzt. Stellungsparametern kann auf diese Weise kein neuer Wert zugewiesen werden.

`${name:?wort}` Ist die Variable *name* gesetzt und besitzt sie einen nichtleeren Wert, dann wird der Ausdruck durch diesen Wert ersetzt. Andernfalls wird *wort* als Fehlermeldung ausgegeben und die Shell beendet. Fehlt *wort*, dann wird stattdessen die Meldung *parameter null or not set* ausgegeben.

`${name:+wort}` Ist die Variable *name* gesetzt und besitzt sie einen nichtleeren Wert, dann wird der Ausdruck durch diesen Wert ersetzt. Andernfalls wird er durch eine leere Zeichenfolge ersetzt.

- \$n** *n* kann eine Ziffer zwischen 0 und 9 sein. Für *\$0* wird der Name der aufgerufenen Shell-Prozedur eingesetzt, für *\$1* der erste Parameter des Aufrufs usw.
Beispiel:
Wenn Sie eine Shell-Prozedur *proc* folgendermaßen aufrufen:
proc Huber Meier
dann wird für *\$0* *proc* eingesetzt, für *\$1* der erste Parameter *Huber* und für *\$2* der zweite Parameter *Meier*.
- \$*** Für *\$** werden alle vorhandenen Stellungsparameter eingesetzt, angefangen bei *\$1* und getrennt durch Leerzeichen. Sie bilden dann eine einzige Zeichenfolge: "*\$1 \$2 ...*".
- \$@** Für *\$@* werden alle vorhandenen Stellungsparameter eingesetzt, angefangen bei *\$1* und getrennt durch Leerzeichen. Sie bilden dann *n* einzelne Zeichenfolgen: "*\$1*", "*\$2*", ...
- \$#** Die Anzahl der beim Prozeduraufruf übergebenen oder durch das eingebaute Shell-Kommando *set* erzeugten Stellungsparameter.
- \$-** Die Schalter, die der Shell beim Prozeduraufruf oder über das eingebaute Shell-Kommando *set* übergeben wurden.
- \$?** Der Endestatus des zuletzt synchron ausgeführten Kommandos.
- \$\$** Die Prozeßnummer der aktuellen Shell.
- #!** Die Prozeßnummer des zuletzt angestoßenen Hintergrundprozesses.

Weitere Informationen finden Sie in der Beschreibung des Kommandos *sh(1)* im Buch *SINIX Kommandos*.

SINIX-Arbeitsumgebung

Unter SINIX V5.2 stehen für den "Benutzer" zwei verschiedene Arbeitsumgebungen zur Verfügung, die man auch "Universen" nennt: *att* und *sie*

att

Arbeitsumgebung ist mit dem X/OPEN-Standard konform.

Hier können die Benutzer arbeiten, die in der Programmentwicklung tätig sind. Im vorliegenden Buch sind nur Kommandos für dieses Universum enthalten. (Keine Systemverwalter-Kommandos.)

sie

Arbeitsumgebung entspricht Betriebssystemversion SINIX V2.0. In SINIX V5.2 können Sie also auch so arbeiten, wie mit dem Betriebssystem SINIX V2.0.

Wenn Sie sich im *att*-Universum befinden, können Sie, ohne das Universum zu wechseln, durch einfaches Vorsetzen von *sie* auch diese Kommandos verwenden, z.B. *sie datum*.

Informationen zu SINIX V2.0 finden Sie im aktuellen Stand, im SINIX Buch 1 V5.0 und im SINIX Buch 2 V5.0 .

Mit dem Kommando *universe*, kann man feststellen, in welchem Universum man sich gerade befindet. Mit den Kommandos *att* und *sie* kann man jeweils in das andere Universum wechseln. Jedes dieser Kommandos ruft eine neue Login-Shell auf. Dabei werden jeweils die Dateien */etc/profile* und *\$HOME/.profile* ausgeführt und in das HOME-Dateiverzeichnis gewechselt (siehe Kapitel Dateisystem). Die neue Login-Shell ist eine Subshell der aufrufenden Shell. Mit der Taste **END** kehren Sie in das zuletzt aufgerufene Universum zurück.

Ausdrücke

Folgende Kommandos verarbeiten Ausdrücke:

Reguläre Ausdrücke

csplit ed ex expr grep pg sed vi

Erweiterte reguläre Ausdrücke

awk egrep lex

Reguläre Ausdrücke

	Syntax	Bedeutung	Beispiel	
	Ein regulärer Ausdruck ist:	Der linksstehende reguläre Ausdruck steht für:	regulärer Ausdruck	passende Zeichenfolge(n)
1	c	c wobei c jedes Zeichen sein kann außer: \ ^ \$. / [] * + -	a	a
2	\c	c wobei c jedes Zeichen sein kann außer: () 1 2 3 4 5 6 7 8 9 0	\a *	a *
3	.	ein beliebiges Zeichen	.	m oder * oder 3 usw.
4a	[s] [a-b]	ein Zeichen aus s, wobei s eine Zeichenfolge ist, in der das Zeichen] nur als erstes Zeichen auftreten darf. In s ist das Zeichen \ kein Metazeichen. ein Zeichen aus dem Bereich a bis b, wobei nach der ASCII-Tabelle a<b sein muß.	[mz] [a-d]	m oder z a oder b oder c oder d
4b	[^ß] [^ä-b]	ein Zeichen, das nicht in s vorkommt, wobei s eine Zeichenfolge ist, in der das Zeichen] nur als erstes Zeichen auftreten darf. In s ist das Zeichen \ kein Metazeichen. ein Zeichen, das nicht im Bereich a-b vorkommt wobei nach der ASCII-Tabelle a<b sein muß.	[^\.5a&] [^ä-z]	jedes Zeichen außer: \ . 5 a & jedes Zeichen außer Kleinbuchstaben
5	r*	null, eine oder mehrere zu r passende Zeichenfolgen, wobei r ein rA der Form 1-4 ist.	p*	Leerzeichen oder p oder pp oder ppp usw.
6	xy	eine zu x passende Zeichenfolge, gefolgt von einer zu y passenden Zeichenfolge, wobei x und y rA sind.	.l [124]r	zl oder 4l oder &l usw 1r oder 2r oder 4r

	Syntax	Bedeutung	Beispiel	
			regulärer Ausdruck	passende Zeichenfolge(n)
7a	$\wedge r$	am Zeilenanfang eine zu r passende Zeichenfolge, wobei r ein rA der Form 1-9 ist.	$\wedge[aA]pfel$	am Zeilenanfang: apfel oder Apfel
7b	$r\$$	am Zeilenende eine zu r passende Zeichenfolge, wobei r ein rA der Form 1-9 ist.	birne\$	am Zeilenende: birne
8a	$r\{m,n\}$	mindestens m , höchstens n Wiederholungen einer zu r passenden Zeichenfolge.	heu\{2,4\}	heu heu oder heu heu heu oder heu heu heu heu
8b	$r\{m\}$	genau m Wiederholungen einer zu r passenden Zeichenfolge.	heu\{3\}	heu heu heu
8c	$r\{m,\}$	mindestens m Wiederholungen einer zu r passenden Zeichenfolge.	heu\{3,\}	heu heu heu oder heu heu heu heu ...
9a	$\backslash(r\backslash)$	eine zu r passende Zeichenfolge, wobei r ein rA ist, in dem höchstens viermal die Zeichenfolge $\backslash($ vorkommt. Die öffnenden $\backslash($ in einem rA werden mit 1 beginnend von links nach rechts durchnumeriert (siehe 9b).	$\backslash(haus\backslash)$ $\backslash(haus\backslash(tür\backslash)\backslash)$	Haus haustür
9b	$x\backslash n$	eine zu r passende Zeichenfolge, wobei r ein mit dem n -ten $\backslash(\backslash)$ -Paar geklammerter Teilausdruck des rA x ist. Für die Anzahl n der $\backslash(\backslash)$ -Paare gilt: $1 < n < 5$.	$\backslash(haus\backslash(tür\backslash)\backslash)\backslash 2$	tür

Erweiterte Reguläre Ausdrücke

Zusätzlich zu den vorstehenden Regeln für reguläre Ausdrücke gelten die folgenden Regeln.

	Syntax	Bedeutung	Beispiel	
	Ein regulärer Ausdruck ist:	Der linksstehende reguläre Ausdruck steht für:	regulärer Ausdruck	passende Zeichenfolge(n)
10a	r+	eine oder mehrere zu r passende Zeichenfolgen, wobei r ein rA ist.	(ok)+	ok oder okokok usw.
10b	r?	null oder eine zu r passende Zeichenfolge, wobei r ein rA ist.	(ok)?	Leerzeichen oder ok
11	u v	eine zu u oder zu v passende Zeichenfolge, wobei u und v reguläre Ausdrücke sind.	tür(schloß riegel)	türschloß oder türriegel
12	(r)	eine zu r passende Zeichenfolge, wobei r ein rA ist.	(aber) (bla)*	aber Leerzeichen oder bla oder blabla usw.

Prioritäten

Priorität der Operatoren in regulären Ausdrücken:

Operator	Priorität
[...]	höchste Priorität
* ? +	.
Verkettung	.
	niedrigste Priorität

ASCII-Zeichen

dezi- mal	oktal	hexa- dez.		Bedeutung	Control
0	00	00	NUL	Null, keine Operation	@
1	01	01	SOH	Start of Heading Vorspannanfang	A
2	02	02	STX	Start of Text Textanfang	B
3	03	03	ETX	End of Text Textende	C
4	04	04	EOT	End of Transmission Übertragungsende	D
5	05	05	ENQ	Enquiry Stationsanruf	E
6	06	06	ACK	Acknowledge Bestätigung	F
7	07	07	BEL	Bell Klingel	G
8	10	08	BS	Backspace Korrekturtaste	H
9	11	09	HT	Horizontal Tabulation Tabulatorzeichen	I
10	12	0A	LF	Line Feed Zeilenvorschub, neue Zeile	J
11	13	0B	VT	Vertical Tabulation	K
12	14	0C	FF	Form Feed Formularvorschub	L
13	15	0D	CR	Carriage Return Wagenrücklauf	M
14	16	0E	SO	Shift Out Umschalten Zeichensatz	N
15	17	0F	SI	Shift In Zurückschalten Zeichensatz	O
16	20	10	DLE	Data Link Escape Austritt aus der Datenverbindung	P
17	21	11	DC1	Device Control 1 Gerätesteuerung 1, Ausgabe fortsetzen	Q
18	22	12	DC2	Device Control 2	R
19	23	13	DC3	Device Control 3	S
20	24	14	DC4	Device Control 4	T
21	25	15	NAK	Negative Acknowledge Fehlermeldung	U
22	26	16	SYN	Synchronous Idle Synchronisierung	V
23	27	17	ETB	End of Transm. Block Datenblockende	W
24	30	18	CAN	Cancel ungültig, Zeilenlöscher	X
25	31	19	EM	End of Medium Datenträgerende, quit (Signal3)	\ oder I
26	32	1A	SUB	Substitute Character Zeichen ersetzen	Z

ASCII-Zeichen

dezi- mal	oktal	hexa- dez .		Bedeutung		Control
27	33	1B	ESC	Escape	Rücksprung	
28	34	1C	FS	File Separator	Dateitrennung	\
29	35	1D	GS	Group Separator	Gruppentrennung	}
30	36	1E	RS	Record Separator	Satztrennung	^
31	37	1F	US	Unit Separator	Einheitentrennung	␣ oder DEL
32	40	20	SP	SPACE	Leerzeichen	
33	41	21	!			
34	42	22	"			
35	43	23	#			
36	44	24	\$			
37	45	25	%			
38	46	26	&			
39	47	27	'			
40	50	28	(			
41	51	29	)			
42	52	2A	*			
43	53	2B	+			
44	54	2C	,			
45	55	2D	-			
46	56	2E	.			
47	57	2F	/			
48	60	30	0			
49	61	31	1			
50	62	32	2			
51	63	33	3			
52	64	34	4			
53	65	35	5			
54	66	36	6			
55	67	37	7			
56	70	38	8			
57	71	39	9			
58	72	3A	:			
59	73	3B	;			
60	74	3C	<			
61	75	3D	=			
62	76	3E	>			
63	77	3F	?			
64	100	40	@	(kaufmännisches "at" oder \$)		
65	101	41	A			
66	102	42	B			
67	103	43	C			
68	104	44	D			
69	105	45	E			
70	106	46	F			
71	107	47	G			
72	110	48	H			
73	111	49	I			
74	112	4A	J			
75	113	4B	K			
76	114	4C	L			
77	115	4D	M			

dezi- mal	oktal	hexa- dez .		Bedeutung	Control
78	116	4E	N		
79	117	4F	O		
80	120	50	P		
81	121	51	Q		
82	122	52	R		
83	123	53	S		
84	124	54	T		
85	125	55	U		
86	126	56	V		
87	127	57	W		
88	130	58	X		
89	131	59	Y		
90	132	5A	Z		
91	133	5B	[	oder Ä	
92	134	5C	\	oder Ò	
93	135	5D	]	oder Û	
94	136	5E	^	oder ↓	
95	137	5F	`	oder →	
96	140	60			
97	141	61	a		
98	142	62	b		
99	143	63	c		
100	144	64	d		
101	145	65	e		
102	146	66	f		
103	147	67	g		
104	150	68	h		
105	151	69	i		
106	152	6A	j		
107	153	6B	k		
108	154	6C	l		
109	155	6D	m		
110	156	6E	n		
111	157	6F	o		
112	160	70	p		
113	161	71	q		
114	162	72	r		
115	163	73	s		
116	164	74	t		
117	165	75	u		
118	166	76	v		
119	167	77	w		
120	170	78	x		
121	171	79	y		
122	172	7A	z		
123	173	7B	{	oder ä	
124	174	7C		oder ö	
125	175	7D	}	oder ü	
126	176	7E	~	oder ß	
127	177	7F	DEL	Delete Löschzeichen, Interrupt (Signal2)	

—

—

—

—

Fachwörter

Fachwörter englisch - deutsch

absolute symbol	absolutes Symbol
access	Zugriff
access mode	Zugriffsart
access permission	Zugriffsberechtigung
action	Aktion
address	Adresse
address space	Adreßraum
administrator	Systemverwalter, Administrator
ambiguous	mehrdeutig
archive	Bibliothek
array	Feld, Vektor
assembler	Assembler
assembly source program	Assembler-Quellcode, Assembler-Quellprogramm
assignment	Zuweisung
automatic	automatisch
automaton	Automat
back space key	Korrekturtaste
background-process	Hintergrundprozeß
backslash	Gegenschrägstrich
base address	Basisadresse
binary file	Binärdatei
binary operator	zweistelliger Operator
blank	Leerzeichen
braces, curly braces	geschweifte Klammern
break	break, break-Anweisung
breakpoint	Haltepunkt
bss	Datensegment, bss-Segment, nicht-initialisierte Daten
buffer	Puffer
bug	Fehler
built-in commands	eingebaute Shell-Kommandos
call	aufrufen
carriage return	Wagenrücklauf
cast	cast, cast-Anweisung
character	Zeichen

check-sum	Prüfsumme	
checker	Prüfprogramm	
cleanup	Bereinigung	⌋
compiler	Übersetzer	
compiler-compiler	Parsergenerator	
concatenation	Verkettung	
console	Konsole	
context sensitivity	Kontextabhängigkeit	
context-free	kontextfrei	
control character	Kontrollzeichen	
control statement	Kontrollanweisung	
core	Speicherabzug	
core dump	Speicherabzug	
core image	Speicherabzug	⌋
corefile	Speicherabzug	
CPU	Zentraleinheit	
curly braces	geschweifte Klammern	
current	aktuell	
current environment	aktuelles Umfeld	
cursor	Schreibmarke	
data area	Datensegment	
data segment	Datensegment	
data symbol	Datensymbol	
debugger	Programm zur Fehleranalyse, Testhilfe	
debugging	Fehlersuche	
default value	Standardwert	
delete	löschen	⌋
delta	Version, Delta	
descriptive text	beschreibender Text	
device	Gerät, externer Datenträger	
digit	Ziffer	
directory	Dateiverzeichnis	
disambiguate	Mehrdeutigkeiten auflösen	
disk	Festplatte	
dot command	Punkt-Kommando	
dummy	Dummy	
effective GID	effektive Gruppennummer	⌋
effective PID	effektive Prozeßnummer	
effective UID	effektive Benutzernummer	
elapsed time	Laufzeit	

end-of-file	Dateiende
entry point	Startadresse
environment	Shell-Umgebung, Umgebung
environmental variable	Umgebungsvariable
error	Fehler
error code	Fehlercode
error diagnostic	Fehlermeldung
error handling	Fehlerbehandlung
escape	Flucht
executable program	ablauffähiges Programm
execute	ausführen, durchsuchen
execute permission	Ausführerlaubnis
execute permission	Ausführberechtigung
execution	Ausführung
exit status	Endestatus
external reference	externe Referenz
external symbol	externes Symbol
field	Feld
field separator	Feldtrenner
file	Datei
file descriptor	Dateikennzahl, Dateideskriptor
file system	Dateisystem
finite automaton	endlicher Automat
flag	Schalter, Parameter
floating point number	Gleitkommazahl, Gleitpunktzahl
floppy	Diskette
floppy disk	Diskette
free list	Freispeicherliste
group ID (GID)	Gruppennummer
group identification (GID)	Gruppennummer
groupname	Gruppenname
head line	Anfangszeile
home-directory	Home-Dateiverzeichnis
id keyword	Identifikations-Schlüsselwort
identification keyword	Identifikations-Schlüsselwort
include file	Include-Datei
initial delta	Originaldatei
inode	Indexeintrag
input	Eingabe
input stream	Eingabestrom
insert	einfügen

instruction	Befehl	
interrupt	Unterbrechung	
inumber	Indexnummer	
kernel	Systemkern	⌋
key	Schlüssel, Taste	
keyboard	Tastatur	
keyword parameter	Kennwortparameter	
lexical analyzer	Scanner	
library	Bibliothek	
link	Verweis	
loader	Binder	
local directory	aktuelles Dateiverzeichnis	
lock	sperrern	
login name	Login-Name	⌋
login-directory	Login-Dateiverzeichnis	
lookahead	Vorausschau	
loop	Schleife	
LR(1) parsing algorithm	LR(1)-Parsing-Algorithmus	
map	Abbildung	
match	passen	
memory	Speicherplatz	
meta character	Metazeichen	
mode	Zugriffsrecht	
modification request	Änderungsgrund, Änderungs- wunsch	
monadic operator	einstelliger Operator	
MR	Änderungsgrund, Änderungs- wunsch	⌋
name-list	Symboltabelle	
new line	Neue Zeile	
node	Knoten	
nonterminal symbol	Nicht-Terminalsymbol, Nicht-Terminalzeichen, syntaktische Variable	
null command	Leeres Kommando	
object code	Objektcode, assembliertes Programm	
object file	Objektdatei	⌋
object program	Objektmodul, Objektdatei	
offset	Offset, Abstand	
optimizer	Optimierer	

option	Schalter
output	Ausgabe
owner	Eigentümer
parentheses	runde Klammern
parser	Parser
parsing table	Übergangstabelle
password	Kennwort
path	Pfad
pathname	Pfadname
pattern	Muster
phase error	Laufzeitinkonsistenz, Phasenfehler
pipe	Pipe, Einwegkanal
pipeline	Pipeline
pointer	Zeiger
portability	Portierbarkeit
portable	portierbar
positional parameter	Stellungsparameter
preprocessor	Präprozessor
priority	Priorität
procedure	Prozedur
process	Prozeß
process termination	Prozeßbeendigung
profile data	Zeittabelle
prompt	Bereitzeichen, Eingabeaufforderungszeichen
protection bit	Schutzbit
queue (FIFO)	Warteschlange
quoting mechanism	Apostrophier-Mechanismus
random number	Zufallszahl
read permission	Leseerlaubnis
real GID	reale Gruppennummer
real PID	reale Prozeßnummer
real UID	reale Benutzernummer
record separator	Zeilentrenner
redirection	umleiten
regular expression	regulärer Ausdruck
relational expression	Vergleichsausdruck
relocation bit	Relokationsbit
reminder	Gedächtnisstütze
request	Anfrage

root directory	Root-Dateiverzeichnis	
rule	Regel	
rule section	Regelteil	
runtime startoff	Laufzeit-Startfunktion	⌋
scanner	Scanner	
scanning pattern	Textmuster	
schedule	steuern	
screen	Bildschirm	
script	Prozedur	
search string	Suchzeichenfolge	
segment	Segment	
shell	Shell	
SID	Versionsnummer, Deltanummer	
sign	Vorzeichen	⌋
size	Größe	
slash	Schrägstrich	
source code	Quellcode	
source programm	Quellprogramm	
space	Zwischenraum	
special character	Sonderzeichen	
special file	Gerätedatei	
stack (LIFO)	Keller	
standard error	Standard-Fehlerausgabe	
standard input	Standard-Eingabe	
standard output	Standard-Ausgabe	
start symbol	Startsymbol	
state	Zustand	
static	statisch	⌋
static data area	statischer Datenbereich	
string	Zeichenreihe, Zeichenfolge, Zeichenkette	
super user	Systemverwalter, Administrator	
symbol table	Symboltabelle	
syntax error	Syntaxfehler	
system call	Systemaufruf	
system time	Systemzeit	
tab	Tabulatorzeichen	
temporary file	Zwischendatei, temporäre Datei	⌋
terminal	Bildschirm, Datensichtstation	
terminal symbol	Token, Terminalzeichen, Terminalsymbol	

text character	Textzeichen
text image	Textsegment
text segment	Textsegment
text symbol	Textsymbol
token	Token, Terminalzeichen, Terminalsymbol
top delta	Topdelta
trace	Ablauf verfolgen
underscore	Unterstrich
user	Benutzer
user ID (UID)	Benutzernummer
user identification (UID)	Benutzernummer
user time	Benutzerzeit
user-defined variable	Kennwortparameter
validate	(auf Konsistenz) prüfen
value	Wert
verifier	Prüfprogramm
version	Version, Delta
working directory	aktuelles Dateiverzeichnis
write permission	Schreiberlaubnis

Fachwörter deutsch - englisch

Abbildung	map	—
Ablauf verfolgen	trace	
ablauffähiges Programm	executable program	
absolute Symbol	absolute symbol	
Administrator	super user	
Adresse	address	
Adreßraum	address space	
Änderungsgrund	MR, modification request	
Änderungswunsch	MR, modification request	
Aktion	action	
aktuell	current	—
aktuelles Dateiverzeichnis	working directory, local directory	
aktuelles Umfeld	current environment	
Anfangszeile	head line	
Anfrage	request	
Apostrophier-Mechanismus	quoting mechanism	
Assembler	assembler	
Assembler-Quellcode	assembly source program	
assembliertes Programm	object program	
aufrufen	call	
Ausführberechtigung	execute permission	
ausführen, durchsuchen	execute	
Ausführung	execution	
Ausführerlaubnis	execute permission	—
Ausgabe	output	
Automat	automaton	
automatisch	automatic	
Basisadresse	base address	
Baum	tree	
Befehl	instruction	
Benutzer	user	
Benutzerkennung	user identification	
Benutzernummer	user ID (UID)	
Benutzerzeit	user time	—
Bereinigung	cleanup	
Bereitzeichen	prompt	
beschreibender Text	descriptive text	
Bibliothek	library, archive	

Bildschirm	screen
Bildschirm, Datensichtstation	terminal
Binder	loader
Binärdatei	binary file
break-Anweisung	break
bss-Segment	bss
cast	cast
cast-Anweisung	cast
Datei	file
Dateideskriptor	file descriptor
Dateiende	end-of-file
Dateikennzahl	file descriptor
Dateiname	file name
Dateisystem	file system
Dateiverzeichnis	directory
Datensegment	data area, data segment
Datensegment, nicht-initialisierte Daten	bss
Datensichtstation	terminal
Datensymbol	data symbol
Delta	Delta, Version
Deltanummer	SID
Diskette	floppy disk, floppy
doppelte Gleitpunktzahl	double floating point
Dummy	dummy
effektive Benutzernummer	effective UID
effektive Gruppennummer	effective GID
Eigentümer	owner
einfügen	insert
Eingabe	input
Eingabeaufforderungszeichen	prompt
Eingabestrom	input stream
eingebaute Shell-Kommandos	built-in commands
einstelliger Operator	monadic operator
Endestatus	exit status
endlicher Automat	finite automaton
externe Referenz	external references
externes Symbol	external symbol
Fehler	error, bug
Fehlerbehandlung	error handling
Fehlercode	error code

Fehlermeldung	error diagnostic
Fehlersuche	debugging
Feld	array, field
Feldtrenner	field separator
Festplatte	disk
Flucht	escape
Freispeicherliste	free list
Gedächtnisstütze	reminder
Gegenschrägstrich	backslash
Gerät, externer Datenträger	device
Geräte-datei	special file
geschweifte Klammern	braces, curly braces
Gleitkommazahl	floating point number
Gleitpunktzahl	floating point number
Gruppenname	group name
Gruppennummer	group ID (GID)
Haltepunkt	breakpoint
Hintergrundprozeß	background-process
Home-Dateiverzeichnis	home-directory
Identifikations-Schlüsselwort	id keyword, identification keyword
Include-Datei	include file
Indexeintrag	inode
Indexnummer	inumber
Keller	stack (LIFO)
Kennwort	password
Kennwortparameter	user-defined variable, keyword parameter
Knoten	node
Konsole	console
Kontextabhängigkeit	context sensitivity
kontextfrei	context-free
Kontrollanweisung	control statement
Kontrollzeichen	control character
Korrekturtaste	back space key
Laufzeit	elapsed time
Laufzeit-Startfunktion	runtime startoff
Laufzeitinkonsistenz	phase error
Leeres Kommando	null command
Leerzeichen	blank
Leseerlaubnis	read permission

Login-Dateiverzeichnis	login-Directory
Login-Name	login name
LR(1)-Parsing-Algorithmus	LR(1) parsing algorithm
löschen	delete
mehrdeutig	ambiguous
Mehrdeutigkeiten auflösen	disambiguate
Metazeichen	meta character
Muster	pattern
Neue Zeile	new line
Nicht-Terminalsymbol	nonterminal symbol
Nicht-Terminalzeichen	nonterminal symbol
Objektcode	object code
Objektdatei	object program, object file
Objektmodul	object program
Offset	offset
Optimierer	optimizer
Originaldatei	initial delta
Parameter	flag
Parser	parser
Parsergenerator	compiler-compiler
passen	match
Pfad	path
Pfadname	pathname
Phasenfehler	phase error
Pipe, Einwegkanal	pipe
Pipeline	pipeline
portierbar	portable
Portierbarkeit	portability
Priorität	priority
Prozedur	procedure, script
Prozeß	process
Prozeßbeendigung	process termination
Präprozessor	preprocessor
Präzedenz	precedence
(auf Konsistenz) prüfen	validate
Prüfprogramm	checker, verifier
Prüfsumme	check-sum
Puffer	buffer
Punkt-Kommando	dot command
Quellprogramm	source program
Quellcode	source code

reale Benutzernummer	real UID
reale Gruppennummer	real GID
reduzieren	reduce
Regel	rule
Regelteil	rule section
regulärer Ausdruck	regular expression
Relokationsbit	relocation bit
Root-Dateiverzeichnis	root directory
runde Klammern	parentheses
Scanner	lexical analyzer, scanner
Schalter	flag, option
Schleife	loop
Schreiberlaubnis	write permission
Schreibmarke	cursor
Schrägstrich	slash
Schutzbit	protection bit
Segment	segment
Shell	shell
Shell-Umgebung	environment
Sonderzeichen	special character
Speicherabzug	core, core image, core dump, corefile
Speicherplatz	memory
sperren	lock
Standard-Ausgabe	standard output
Standard-Eingabe	standard input
Standard-Fehlerausgabe	standard error
Standardwert	default value
Startadresse	entry point
Startsymbol	start symbol
statisch	static
statischer Datenbereich	static data area
Stellungsparameter	positional parameter
steuern	schedule
Suchzeichenfolge	search string
Symboltabelle	symbol table, name-list
syntaktische Variable	nonterminal symbol
Syntaxfehler	syntax error
Systemaufruf	system call
Systemkern	kernel
Systemverwalter	super user, administrator

Systemzeit	system time
Tabulatorzeichen	tab
Tastatur	keyboard
Taste	key
Terminalsymbol	token, terminal symbol
Terminalzeichen	token, terminal symbol
Testhilfe	debugger
Textmuster	scanning pattern
Textsegment	text image, text segment
Textsymbol	text symbol
Textzeichen	text character
Token	token, terminal symbol
Topdelta	top delta
Übergangstabelle	parsing table
Übersetzer	compiler
Umgebung	environment
Umgebungsvariable	environmental variable
umleiten	redirect
Unterbrechung	interrupt
Unterstrich	underscore
Vergleichsausdruck	relational expression
Verkettung	concatenation
Version	delta, version
Versionsnummer	SID
Verweis	link
Vorausschau	lookahead
Vorrang	precedence
Vorzeichen	sign
Wagenrücklauf	carriage return
Warteschlange	queue (FIFO)
Wert	value
Zeichen	character
Zeichenfolge	string
Zeichenkette	string
Zeichenreihe	string
Zeiger	pointer
Zeilentrenner	record separator
Zeittabelle	profile data
Zentraleinheit	CPU
Ziffer	digit
Zufallszahl	random number

deutsch - englisch

Zugriff	access
Zugriffsart	access mode
Zugriffsberechtigung	access permission
Zugriffsrecht	mode
Zustand	state
Zuweisung	assignment
zweistelliger Operator	binary operator
Zwischendatei	temporary file
Zwischenraum	space

Literatur

SINIX-Handbücher

Betriebssystem SINIX V5.2

Kommandos

Teil 1

Teil 2

Betriebssystem SINIX V5.2

Systemverwaltung

Betriebssystem SINIX V5.2

CES C-Entwicklungssystem

Teil 1

Teil 2

Betriebssystem SINIX V5.2

Buch 2

Menüs

Betriebssystem SINIX V5.0

Kommandos

Englischsprachige Ausgabe

Betriebssystem SINIX V5.0A

Buch 1

(heißt ab SINIX V5.2 Einführung)

Betriebssystem SINIX V5.0A

CES Buch 1

Werkzeuge zur C-Programmierung

Grundlagen und Kommandos

Betriebssystem SINIX V5.0A

CES Buch 2

Werkzeuge zur C-Programmierung

Systemaufrufe, C-Funktionen und Makros

Literatur

X/OPEN Guide

X/OPEN Portability Guide

Elsevier Science Publishing Company Amsterdam, 1987

Literatur zu UNIX

M.J. Bach

The Design of the UNIX System

Prentice-Hall Englewood Cliffs, 1986

M. Banahan, A. Rutter

UNIX lernen, verstehen, anwenden

Deutsche Ausgabe von Prof. Dr. A.T. Schreiner, T.Mandry

Carl Hanser Verlag München, 1984

M. Banahan, A. Rutter

UNIX - the book

Wilmslow, UK, Sigma Technical Press, 1982

S.R. Bourne

Das UNIX System

Addison-Wesley Verlag (Deutschland) GmbH, 1985

Kaare Christian

The UNIX Operating System

John Wiley & Sons, 1983

Jürgen Gulbins

UNIX

Springer-Verlag Berlin, 1985

B.W. Kernighan, R. Pike

Der UNIX-Werkzeugkasten

Carl Hanser Verlag München, 1987

D.M. Ritchie

The UNIX Time-Sharing System: A Retrospective

Bell System Technical Journal,

Vol. 57, No. 6, 1947-69

American Telephone and Telegraph Company, 1978

R. Thomas und J. Yates

A User Guide to the Unix System

Berkeley: Osborne/McGraw-Hill, 1982

R. Thomas, L.R. Rogers, J. Yates

Advanced Programmer's Guide to UNIX System V

Berkeley: Osborne/McGraw-Hill, 1986

Literatur zur Sprache C

H. Herold, W. Unger

Das C-Buch

te-wi Verlag München, 1986

B.W. Kernighan, D.M. Ritchie

Programmieren in C

Deutsche Ausgabe von Prof. Dr. A.T. Schreiner, Dr. Ernst Janich

Carl Hanser Verlag München, 1983

T. Plum

Das C-Lernbuch

Carl Hanser Verlag München Wien, Prentice-Hall London, 1985

—

—

—

—

Stichwörter

.profile 2-33

/bin und /usr/bin 3-5

Ablaufanweisungen für Shell 3-38

Absoluter Pfad 2-4

Aktuelles Dateiverzeichnis 2-5

Akzent Gravis Shell 3-12

Andere, Benutzergruppe 2-23

Anführungszeichen Shell 3-12

Apostroph Shell 3-12

Apostrophier-Mechanismus 3-12

aufrufen Prozedur 3-24

ausführen Kommandos 3-55

Ausführungsberechtigung 2-24

ausgeben bildschirmweise 3-9

Baumstruktur 2-2

Begrüßungsbildschirm 1-2

Benutzergruppen 2-22

Benutzerkennung 1-2

- gast 1-2

Benutzerklassen 2-23

Benutzernummer 2-29

Bildschirm dunkel 1-2

break 3-39

Case 3-40

ced

- Arbeits-Modi 5-11

- Aufruf 5-13

- Beispiel klein 5-7

- Bildschirm 5-13

- neu aufbauen 5-46

- Blättern im Dokument 5-16

- Block verschieben 5-18

- Dokument wechseln 5-34

- Eckpunkt markieren 5-21

- einfügen mit Taste 5-10

- einzelne Zeile bearbeiten 5-30

- Fenster verschieben 5-41

- HELP-Taste 5-12

- löschen mit Taste 5-10

- Rechteck 5-21

- bewegen 5-22

- schnelle Schreibmarkenbewegungen 5-16

- Schreibmarke bewegen 5-10

- SINIX-Kommando aufrufen 5-26

- Sitzung beenden 5-38

- Spalten vertauschen 5-19

- Tasten programmieren 5-43

- Tastenfunktionen 5-9

- Text

- eingeben 5-15

- sichern 5-37

- suchen 5-39

- und sort 5-27

- vorübergehend unterbrechen 5-42

- Zeilenbereich verschieben 5-18

- Zwischenspeicher 5-19

continue 3-39

D-Datei (SCCS) 6-11

Datei 2-10

- .profile 2-33

- /etc/profile 2-33

- erweitern 3-6

- erzeugen 2-10

- für Gerät 2-12

- Größe in Byte 2-19

- kopieren 2-7

- letzte Änderung 2-19

- letzter Zugriff 2-19

- löschen 2-11

- starten 3-66

Datei-Identifikationszeichen 2-18

Datei-Link 2-8

Datei-Zugriffsschutz 2-22

Dateien

- Versionen verwalten 6-1

- zusammenfassen 3-6
- Dateiname 2-13
 - Sonderzeichen 2-13
- Dateisystem 2-1
- Dateiverweis 2-19
- Dateiverzeichnis 2-8
 - /dev 2-12
 - aktuelles 2-5
 - erzeugen 2-8
 - löschen 2-9
 - , besondere 2-9
- Dokument bei Editoren 5-2
- Editor
 - Allgemeines 5-1
 - bildschirmorientiert 5-3
 - ced 5-6
 - vi 5-49
 - zeilenorientiert 5-3
- Eigentümer 2-23
- Eingabe prüfen 3-40
- Eingabefehler korrigieren 1-7
- eingeben Kommando 3-3
- Ende-Status 4-4
 - abfragen 3-44, 3-60
 - auswerten 3-61
- Entwerten von Sonderzeichen 2-17
- eval 3-55
- exec 3-56
- exit 3-59
- export 3-64
- exportieren Variable 3-27
- Fehlermeldungen 4-5
- for 3-42
- for-Schleife beenden 3-39
- G-Datei (SCCS) 6-3, 6-10
- Geräteklasse (Majornummer) 2-21
- Gerätenummer (Minornummer) 2-21
- grep 3-43
- Groß- und Kleinbuchstaben 1-4
- Großbuchstaben 1-4

- Gruppe, Benutzergruppe 2-23
- Gruppennummer 2-22, 2-29
- Help (SCCS) 6-3
- Hintergrundprozeß 3-4, 3-37
- Home Dateiverzeichnis 2-9
- If 3-44
- Indexeintrag, Beispiel 2-19
- Kennwort 1-2
 - ändern 1-8
- Kennwortparameter 3-21, 3-24
- kill 3-37, 3-77
- Kindprozeß 3-35
- Kleinbuchstaben 1-4
- Kommando
 - # 3-52
 - () 3-53
 - : 3-52
 - {} 3-54
 - eingeben 3-3
 - Punkt 3-66
 - richtig eingeben 4-3
 - wo beschrieben 4-1
- Kommando-Liste 3-4
 - Sonderzeichen 3-4
- Kommando-Zeile fortsetzen 4-5
- Kommandoeingabe
 - abschließen 1-5
 - Aufforderung 1-4
- Kommandos
 - ausführen 3-55
 - der Shell 3-51
 - übergeben 3-55
 - zusammenfassen 3-53
- Kommandozeile 3-3
 - übergeben 3-72
- Kommandozeilen innerhalb einer Prozedur 3-29
- Kommentar (SCCS) 6-4
- Kommentarzeile schreiben 3-52

L-Datei (SCCS) 6-11
Leseberechtigung 2-24
Levelnummer (SCCS) 6-3
Login 1-2
– **Dateiverzeichnis** 2-9
Login-Shell 3-21, 3-35

MR-Nummer (SCCS) 6-6

P-Datei (SCCS) 6-4, 6-11
Parameter
– **für Shell-Prozedur** 3-21
– **vordefinieren** 3-31
Password 1-3
– **ändern** 1-8
Pfad 2-4
– **absolut** 2-4
– **relativ** 2-4
Pfadname 2-4
Pfeil-Tasten 1-7
Pipe Beispiel 3-11
Pipe-Zeichen 3-8
Pipeline 3-8
Plattenblockadresse 2-19
Prozedur
– **anhalten** 3-67
– **aufrufen** 3-24
– **fortsetzen mit Eingabe** 3-67
Prozeß 3-4
– **abbrechen** 3-37
– **Laufzeit** 3-76
Prozeßabschluß abwarten 3-80
Prozeßhierarchie 3-35
Prozeßkonzept 3-34
Prozeßnummer 3-37
Prozeßstruktur 3-34

Read 3-67
readonly 3-69
Relativer Pfad 2-4
Releasenummer (SCCS) 6-3
Root Dateiverzeichnis 2-9
Root-Directory 2-2

S-Bit 2-29
s-Datei (SCCS) 6-2
SCCS 6-1
SCCS-Datei Zugriffsschutz 6-12
Schalter für Shell setzen 3-71
Schleifen steuern 3-38
Schrägstrich 2-4
Schreibberechtigung 2-24
Schutzbit
– **Beispiel** 2-26
– **Shell-Prozedur** 2-28
– **Standard** 2-25
Schutzbits 2-23
– **setzen** 2-24
– **Standardeinstellung ändern** 3-79
set 3-29, 3-70
sh 3-71
Shell 3-1
– **Ablaufanweisungen** 3-38
– **beendet** 3-56
– **empfangsbereit** 3-3
– **ersetzt** 3-56
– **Kommandos** 3-51
– **Schalter setzen** 3-71
– **Sonderzeichen** 3-18
– **Standard-Variablen** 3-17
– **Variablen** 3-16
Shell-Prozedur 3-2, 3-19, 3-34
– **beenden** 3-59
– **Beispiel** 3-19
– **Parameter** 3-21
– **strukturieren** 3-38
shift 3-74
SID-Nummer (SCCS) 6-3, 6-13
Signal an Shell-Prozedur 3-77
Signalnummer 3-77
SINIX verlassen 1-8
SINIX-Arbeitsumgebung A-12
Sonderzeichen
– **entwerten** 2-17
– **für Dateinamen** 2-13
– **Kommando-Liste** 3-4
– **Shell** 3-18

-
- Standard Schutzbit 2-25
 - Standard-Ausgabe umleiten 3-6
 - Standard-Eingabe
 - für Prozedur umleiten 3-56
 - umleiten 3-7, 3-56
 - Standard-Variablen für die Shell 3-17
 - Standardwerte setzen Parameter 3-31
 - Stellungsparameter 3-21
 - in Prozedur 3-22
 - übergeben 3-22
 - verschieben 3-74
 - Sub-Shell 3-53, 3-59
 - Syntax-Fehler finden 3-70
 - System-Dateiverzeichnisse 2-3
 - Systemkern 3-1
 -
 - Tastatur 1-5
 - Taste
 - DEL 3-3
 - END 1-2, 1-8
 - trap 3-77
 -
 - Übergabe von Variablen 3-28
 - übergeben
 - Kommandos 3-55
 - Kommandozeile 3-72
 - Übersicht
 - Kommando Anwendungsbereiche 4-9
 - Kommandonamen alphabetisch 4-10
 - SCCS-Kommandos 6-19
 - welches Kommando für welche Aufgabe 4-18
 - umask 3-79
 - Umleiten
 - der Standard-Eingabe 3-7
 - Standard-Ausgabe 3-6
 - Umwandeln von Kennwort – in Stellungsparameter 3-30
 - Universum A-12
 - Unterbrechungssignal 3-77
 -
 - until 3-46
 - UUCP
 - andere UNIX-Anlage 6-56
 - Anschluß SS97 6-49
 - Anschluß V24 6-49
 - Beschreibung 6-25
 - Dateien temporär 6-47
 - Installation
 - Hardware 6-49
 - Software 6-53
 - Kommandoformat 6-60
 - Kommandos 6-57
 - Übersicht betroffene Dateien 6-35
 - Was tun, wenn ... 6-75
 -
 - Variable
 - definieren 3-27
 - exportieren 3-27
 - global prüfen 3-71
 - Variablen
 - für die Shell 3-16
 - schützen 3-69
 - weiterreichen 3-64
 - zurückgeben 3-36
 - Variablen-Name 3-16
 - Variablen-Wert 3-16
 - als Kennwortparameter 3-26
 - Vaterprozeß 3-35
 - Version identifizieren (SCCS) 6-2
 - Versionen von Dateien verwalten 6-1
 - Verzweigen mit bestimmter Eingabe 3-40
 - vi
 - Arbeits-Modi 5-54
 - Aufruf 5-63
 - Beispiel klein 5-50
 - Beispiele extra 5-83
 - Bildschirm 5-60
 - Blättern im Dokument 5-67
 - Dateien mehrere 5-82
 - Fehleingabe korrigieren 5-72
 - Fenster verschieben 5-67
 - Kommando mehrfach 5-81

-
- Kommandoformen 5-56
 - Leerzeile einfügen 5-79
 - Optionen 5-90
 - Schreibmarke positionieren 5-74
 - SINIX-Kommando aufrufen 5-66
 - Sitzung beenden 5-65
 - Text
 - ändern und erweitern 5-79
 - bearbeiten 5-77
 - eingeben 5-72
 - löschen 5-77
 - sichern 5-65
 - suchen 5-69
 - Wo bin ich? 5-71
 - Wait 3-80
 - while 3-46
 - while-Schleife beenden 3-39
 - Wurzeldateiverzeichnis 2-2
 - X-Bit 2-28
 - x-Datei (SCCS) 6-9
 - Z-Datei (SCCS) 6-10
 - Zugriffsrecht 2-23
 - Zugriffsschutz, Datei 2-22

1

2

3

4

Kritik – Anregungen – Korrekturen

Schreiben Sie uns bitte, was Ihnen an diesem Buch gefällt und was Ihnen nicht gefällt:

- vor allem, wo Sie Fehler entdeckt haben,
- wo etwas unklar beschrieben ist.

Absender:

Name

Firma/Dienststelle

Straße, Postfach

PLZ Ort

Telefon ()

betreut von ZN

von Hrn./Fr.

Antwort

Bitte mit
60 Pf.
frankieren,
falls Marke
zur Hand.

Siemens AG
K D ST QM2
Manualredaktion
Otto-Hahn-Ring 6
Postfach 830951

D-8000 München 83

Absender:

Name

Firma/Dienststelle

Straße, Postfach

PLZ Ort

Telefon ()

betreut von ZN

von Hrn./Fr.

Antwort

Bitte mit
60 Pf.
frankieren,
falls Marke
zur Hand.

Siemens AG
K D ST QM2
Manualredaktion
Otto-Hahn-Ring 6
Postfach 830951

D-8000 München 83

Nennen Sie uns Ihren »Stolperstein«



Buch: SINIX V5.2 Einführung, U3901-J-Z95-1

Seite: Mein »Stolperstein«: Datum:

Das Handbuch benutze ich: ☐ sehr häufig ☐ gelegentlich zum Nachschlagen
☐ als Programmierer ☐ Systemverwalter
☐ Sachbearbeiter ☐ _____

Buch: SINIX V5.2 Einführung, U3901-J-Z95-1

Seite: Mein »Stolperstein«: Datum:

Das Handbuch benutze ich: ☐ sehr häufig ☐ gelegentlich zum Nachschlagen
☐ als Programmierer ☐ Systemverwalter
☐ Sachbearbeiter ☐ _____