D-NIX 5.3
Användarhandbok

Version A.1 91-04-20

089-9764-10

Diab Data AB
Box 2029
183 02 TÄBY
☎ 08-638 94 00

DIAB DATA

1. Översikt

2. Filsystemets uppbyggnad

3. Det dagliga arbetet

4. D-MACS

1

1. Översikt

Översikt

Kapitel 1

1. Översikt

1.1. D-NIX operativsystem

I denna handbok beskrivs kommandon till operativsystemet D-NIX, som är implementerat på Diabs datorer. Operativsystemet D-NIX är helt kompatibelt med AT&T Unix System V Version 3, och innehåller samma funktioner. Men en del kommandon har fler optioner.

Handboken gör inga anspråk på att ge grunderna till Unix eller D-NIX. Det finns många bra böcker om Unix system. Vi rekommenderar **UNIX SYSTEM V Primer** av *Waite*, *Martin* och *Prata*.

I **Kapitel 2** beskrivs filsystemet i D-NIX. Bl a beskrivs hur åtkomsträttigheterna för filer är uppbyggda.

I **Kapitel 3** beskrivs de vanligaste rutinerna som behövs för det dagliga arbetet, kopiering av filer, utskrift på skrivare samt säkerhetskopiering.

I **Kapitel 4** finns en noggrann beskrivning av editorn *dmacs*. Editorn används ofta i det dagliga arbetet. Den används bl a vid ändring av systemfiler, när du skriver brev för att skicka med *mail*, o s v.

I **Kapitel 5** finns en noggrann beskrivning av shell. I kapitlet finns beskrivning av procedurer, parametertolkning, variabler och hur du kan använda shell för att skriva egna rutiner.

I **Kapitel 6** ges en kort introduktion till elektronisk post, mail, och hur du använder den för att läsa och skriva brev.

I **Kapitel 7** finns en beskrivning av kommunikationsprogrammet kermit. Programmet är ett användbart verktyg för att flytta filer mellan datorer av olika slag.

I **Kapitel 8** finns en kort sammanfattning av de kommandon som finns i D-NIX grundpaket. En noggrann beskrivning av dessa kommandon finns i **Referenshandboken.**

Det bästa sättet att lära sig om Unix är genom övning på terminalen. Var inte rädd för att göra misstag, det är av dem du lär dig.

1.1.1 Teckenuppsättningar

I denna handbok har vi valt att följa den internationella teckenrepresentationen vid beskrivningen av tecken. Det innebär att om man använder sig av ett svenskt tangentbord med svensk 7-bitars teckenuppsättning måste vissa tecken ersättas med andra. Dessa är:

Tecken	Ersätts med
}	å
]	Å
{	ä
[	Ä
|	ö
\	Ö
~	ü
^	Ü
‘	é
@	É
$	¤
#	§

1.2. Kommandoformat

Kommandona i Unix-sfären är något speciella eftersom de med få undantag är kortfattade och inte talar om för användaren vad de utför, t ex *ls*, *su*, *nice*. I regel är kommandona ett sammandrag av ett eller flera ord, t ex *ls* - list contents of directory, *cat* - concatenate (hoplänka).

Kommandona skrivs med små bokstäver. Ett kommando som skrivs med versaler kan inte utföras om inte kommandonamnet också är definierat med versaler i systemet.

Alla kommandon givna i D-NIX tolkas av kommando-processorn Shell. Shell är en förbindelse mellan användare och D-NIX operativsystem. Med Shell kan man omdirigera in- och utdata, tilldela variabler och göra substitutioner (utbyten) på en kommandorad innan ett kommando utförs. I interaktivt mod kan shell användas som programmeringsspråk, vilket gör det möjligt för användaren att skapa egna kommandon genom att kombinera flera kommandon i kommandofiler.

Standardformatet på ett kommando är en sekvens av ord, separerade med ett eller flera mellanslag. Alla kommandon utförs efter nedtryckning av **RETURN**. Flera kommandon kan anges på samma rad om de separeras med ett semikolon ';'

Det första ordet är kommandot och resterande ord är argument till kommandot. Argumenten får ej innehålla mellanslag, men man kan ha flera mellanslag både före, efter och även mellan argumenten. Om ett argument innehåller mellanslag måste det omges av " " eller ' '.

I regel består argumenten av ett av följande uttryck:

Option	Innehåller en eller flera bokstäver - vanligen föregångna av minus-tecken (exempel **-al**). Antingen modifierar optionen kommandot eller anger exakt vilken funktion kommandot skall utföra.
Uttryck	Ett uttryck ('expression') består av en teckensträng som kommandot kan använda.
Filnamn/Bibliotek	Ger namnet på den fil som kommandot ska behandla på ett eller annat sätt. Filnamnet kan bestå av 14 eller färre tecken.

Den vanligaste ordningsföljden för ett kommando är:

```
kommando optioner uttryck filnamn/bibliotek
```

För den exakta ordningsföljden hänvisas till kommandot ifråga.

1.3. Speciella uttryck

Det finns en del specialuttryck som används vid en beskrivning av Unix. De vanligaste beskrivs nedan:

Super-User
En privilegierad användare som har obegränsad åtkomsträtt till alla delar av systemet.

Standard Output
D-NIX skickar resultatet av ett kommando till en fil som heter standard output. Normalt är denna riktad mot terminalen.

Standard Input
Standard Input är den plats varifrån programmet förväntar sig att läsa data. Normalt är detta från tangentbordet.

Standard Error
Felmeddelanden från kommandon skickas normalt till standard error. Normalt är denna fil riktad mot terminalen.

Specialfiler
Detta är specialfiler som står i förbindelse med de externa enheterna såsom terminaler, radskrivare, massminnen. Filerna läses och skrives på liknande sätt som vanliga filer med hänsyn tagen till de yttre enheternas restriktioner.

Wildcards eller Metatecken eller Jokertecken
När filnamn ges på en kommandorad har vissa tecken, wildcards, speciella betydelser. Wildcards kan även heta metatecken eller jokertecken. Shell kommer att avkoda dessa innan argumenten ges till kommandot. Ett argument med metatecken ersätts av flera argument, ett för varje filnamn i filsystemet som matchar tecknen. Tecknen *, ? och [.....] är generella metatecken. Om dessa skall användas utan sin speciella betydelse måste de antingen föregås av tecknet \ eller sättas inom apostrofer '.....'. Se kommandot *sh* för detaljer.

Pathname eller Sökväg
Ett pathname eller en sökväg är ett fullständigt fil- eller biblioteksnamn, vilket innehåller hela sökvägen genom alla bibliotek från rootbiblioteket.

Mounta ett filsystem
Ett eller flera filsystem på yttre skivminnen eller inom en fil kan anslutas (kopplas), till root-filsystemet, så att det yttre filsystemet behandlas som om det vore ett bibliotek inom root-filsystemet. Denna anslutning görs med kommandot mount, varvid filsystemet säges vara mountat.

Utgångsstatus

När ett kommando avslutas returnerar det alltid ett värde till systemet, utgångsstatus. Detta värde är normalt noll (0). Andra värden kan returneras och visar oftast att ett fel har upptäckts, men ibland anger utgångsstatus något resultat från kommandot.

Boota ett system

När en dator startas, måste operativsystemet laddas från ett skivminne eller en diskett och därefter startas. Denna procedur kallas att boota systemet.

Åtkomsttillstånd, Tillståndsmoder Filskydd eller Accesstillstånd

Varje fil eller bibliotek är skyddat mot obehörig användning genom olika åtkomsttillstånd för olika användare. Tillstånden kan även kallas tillståndsmoder, filskydd eller accesstillstånd.

2

2. Filsystemets uppbyggnad

2. Filsystemets uppbyggnad

2.1. Filer och bibliotek i systemet

Efter inloggningen befinner sig super-usern i ett bibliotek, root-biblioteket. Ett bibliotek är en fil som innehåller en lista på filnamn och underbibliotek, dessutom innehåller den information om dessa filer och underbibliotek. Varje underbibliotek innehåller i sin tur en lista på filnamn och underbibliotek och information om var de befinner sig i i filsystemet. I systemet kan det i princip finnas hur många nivåer som helst med underbibliotek. Innehållet i det bibliotek som man för tillfället befinner sig i skrivs ut på skärmen om kommandot l används.

Om inget annat anges förutsätts det i detta kapitel att användaren alltid befinner sig i root-biblioteket. Innehållet i root-bibliotek skrivs ut om följande anges:

- Skriv l (lilla L), tryck **RETURN**.

Exempel på listning av root-biblioteket (innehållet kan variera)

```
total 740
drwxrwxr-x 2  bin  bin     1168 Oct 27 09:13 bin
-rwx------ 1  root sys    51749 Sep 29 09:54 boot
drwxr-xr-x 3  root sys      880 Oct 27 08:14 dev
-rw-r--r-- 1  root sys   231151 Oct 20 16:01 dnix
drwxr-xr-x 2  root sys      688 Oct 27 08:57 etc
drwxrwxrwx 3  root root      80 Oct 23 09:36 hnd
drwxrwxr-x 2  bin  bin       32 Oct 22 13:27 lib
drwxrwxrwx 2  root root    2048 Oct 16 17:56 lost+found
drwxrwxrwx 2  root sys       32 Oct 27 10:58 mf0
drwxrwxrwx 2  root sys       32 Oct 27 12:08 mnt
drwxrwxr-x 3  root root      80 Oct 22 13:26 sas
drwxrwxrwx 2  root sys      400 Oct 27 09:41 tmp
drwxr-xr-x 10 root sys      160 Oct 22 13:43 usr
```

drwxr-xr-x	Åtkomsttillstånd för fil resp bibliotek, förklaras nedan
root	Ägare till biblioteket **usr**
sys	Grupp för biblioteket **usr**
160	Antal bytes i biblioteketsfilen **usr**
Oct 22	Datum för senaste modifiering
13:43	Tid för senaste modifiering
usr	Namn på biblioteket
...	Analog information för övriga filer/bibliotek

Filtyper

Varje rad innehåller information om en fil eller ett bibliotek.

Första positionen på raden anger om det är en fil eller bibliotek, t ex

```
drwxrwxr-x 2  bin  bin    1168 Oct 27 09:13 bin
```

Bokstaven **d** i första positionen anger att detta är ett bibliotek. En fil betecknas på följande sätt:

```
-rwx------ 1  root sys   51749 Sep 29 09:54 boot
```

Om det är en fil anges alltså tecknet - i första positionen.

Ägare/grupp/övriga

Varje fil eller bibliotek som skapas har en ägare. I de flesta fall ägs den av den person som skapade den. Ägaren till en fil eller ett bibliotek kan ge den olika skydd.

I systemet finns det tre olika typer av ägare till en fil eller ett bibliotek:

ägare (user]	Den person som skapade filen eller biblioteket.
grupp (group)	Ett antal olika användare kan bilda en användargrupp. Varje fil eller bibliotek kan ägas av en grupp.
övriga (others)	Övriga består av alla andra användare i systemet. Observera att detta är inte begränsat till den standardgrupp som har namnet 'other'.

I systemet finns det alltså tre ägargrupper.

```
ägare - grupp - övriga
```

Varje användare tillhör en viss grupp, vilken tilldelas då användaren definieras.

Filskydd, åtkomsttillstånd

Varje fil och bibliotek kan i systemet ha tre typer av tillstånd som beskriver vad som får göras med filen eller biblioteket. Beroende på att en fil och ett bibliotek skiljer sig lite från varandra har dessa tillstånd lite olika innebörd. Dessa tillstånd brukar ibland kallas filskydd, åtkomsttillstånd, tillståndsmoder eller accessmoder.

Läs (Read) r	En användare som har lästillstånd i en fil har rätt att titta på innehållet i denna fil. En användare som har lästillstånd i ett bibliotek kan få reda på vilka filer som finns i biblioteket.
Skriv (Write) w	En användare som har skrivtillstånd har rätt att ändra innehållet i denna fil. Notera att detta gäller även om användaren inte har skrivtillstånd i det bibliotek där filen befinner sig.

En användare som har skrivtillstånd i ett bibliotek har rätt att ändra innehållet i detta bibliotek dvs skapa nya filer, ta bort gamla filer.

Använda (Execute) x En användare med bearbetningstillstånd har rätt att använda filnamnet som ett kommando.

En användare som har tillstånd att använda ett bibliotek har rätt att ändra från aktuellt bibliotek till detta bibliotek. Dessutom har användaren rätt att kopiera filer från detta bibliotek förutsatt att användaren har lästillstånd för biblioteket.

Var och en av de tre ägargrupperna kan ha en, två eller alla tre tillstånden. Detta betyder att det totalt finns nio olika tillstånd.

```
Ägare     Grupp     Övriga
rwx       rwx       rwx
```

Om inget tillstånd gäller indikeras detta med tecknet -, vilket kallas för skydd. De nio tillstånden kallas kollektivt för filens eller bibliotekets tillståndsmod. För att exemplifiera detta visas ett par exempel

```
drwxr-xr-x 6 kalle dok 2144 Oct 23 06:17 /usr/karl
```

Ägaren, i det här fallet 'kalle', har läs-, skriv- och bearbetningstillstånd för biblioteket, medan gruppen 'dok' och övriga bara har läs- och bearbetningstillstånd.

För filen boot gäller följande:

```
-rwx------ 1 root sys 51749 Oct 16 14:03 boot
```

Ägaren till filen har rätt att läsa och skriva i filen och även att använda den som ett kommando. Gruppen (sys) och övriga har inga tillstånd alls och kan varken läsa, skriva eller använda den som ett kommando.

Specialfiler

Biblioteket **/dev** innehåller inte vanliga filer. De "filer" som finns i **/dev** kallas specialfiler och används enbart för direkt åtkomst till drivrutiner i operativsystemet för olika fysiska enheter. Dessa beskrivs i **Systemadministration**.

Nya specialfiler kan läggas till enbart med kommandot */etc/mknod* och enbart om motsvarande drivrutin finns i operativsystemet. Specialfiler kan tas bort med kommandot *rm* och kan länkas till varandra med kommandot *ln*.

2.2. Exempel på hur ett system läggs upp

När ett system läggs upp i ett företag finns det många faktorer som måste tas med i beräkningarna. Hur ska de olika grupperna se ut, hur ska användarnumren läggas upp? Alla dessa frågor bör tänkas igenom innan systemet tas i bruk. Det är viktigt att systemet blir upplagt redan från början, annars är det lätt att man hamnar i en situation där det är svårt att ändra systemets uppläggning.

Vi ska här med ett litet exempel försöka visa ett sätt att organisera ett system som fungerar bra i de flesta sammanhang.

Vi antar att vi har ett företag som leds av en VD. Under sig har VD fyra olika avdelningar, ekonomiavdelning, utvecklingsavdelning, produktionsavdelning och marknadsavdelning. I detta företag har man skaffat sig ett datorsystem som ska vara gemensamt för de fyra avdelningarna samt VD. Organisationen ser ut så här:

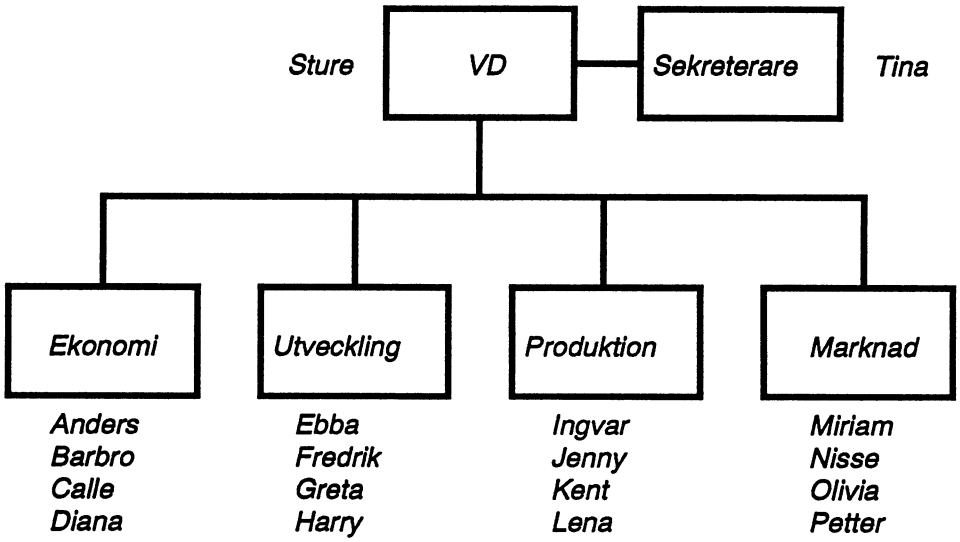

Bild 2.1 Organisationen i vårt exempel

Filen **/etc/passwd** kan se ut på följande sätt när en vanlig användare lagts till (sture):

Exempel på **/etc/passwd** fil:

```
root::0:3:0000-Admin(0000):/:
daemon:*noway*:1:12:0000-Admin(0000):/:
bin:*noway*:2:2:0000-Admin(0000):/bin:
sys:*noway*:3:3:0000-Admin(0000):/usr/src:
adm:*noway*:4:4:0000-Admin(0000):/usr/adm:
uucp:*noway*:5:1:0000-uucp(0000):/usr/lib/uucp:
nuucp::6:1:0000-uucp(0000):/usr/spool/uucppublic:
          /usr/lib/uucp/uucico
rje:*noway:68:8:0000-rje(0000):/usr/rje:
shqer:*noway:69:8:0000-rje(0000):/usr/rje:
lp:*noway:71:2:0000-lp(0000):/usr/spool/lp:
sture:rrFbrSmdXJo0s:111:10:Sture Karlsson:/usr/sture:
```

3

3. Det dagliga arbetet

3. Det dagliga arbetet

3.1. Inloggning

Vid inloggningen frågar systemet efter både login-namn och lösenord (password). För användaren sture kan detta se ut på följande sätt:

```
login: sture
Password: _
```

Före frågan **login:** kan eventuellt systemets nodnamn visas och ett systemmeddelande ur filen **/etc/issue** om dessa definierats. Efter godkänd inloggning, men före prompten kan visas eventuellt ytterligare systemmeddelande ur filen **/etc/motd**, dock inte vid inloggning som **root**.

När systemet är klart skrivs tecknet **$** ut på skärmen. Detta tecken kallas för prompt och den indikerar att det nu är möjligt att börja arbeta med systemet. Prompten kan se ut på andra sätt om den definierats om av systemadministratören eller dig själv. En användare kan befinna sig i olika moder i operativsystemet, där varje mod har sin egen prompt t ex # för super-user mod.

3.2. Utloggning

Vid utloggning trycks **CTRL-D** ned, alternativt anges kommandot *exit* eller *exec login* till systemet.

```
# _
```

- Tryck ned tangenterna **CTRL** och **D** samtidigt. Eller skriv exit och tryck på **RETURN**. OBS! Ingenting skrivs ut på skärmen.

Efter en liten stund skrivs login ut på skärmen.

```
# Console login: _
```

Kommandoavkodaren stoppas och **login:** skrivs ut på skärmen för inloggning av nästa användare. Vid utloggningen stänger systemet alla filer som eventuellt är öppna. Om någon bakgrundsprocess har startats upp väntar inte systemet på att denna process skall avslutas. Om kommandot *exec login* används, sker ny inloggning utan att terminalparametrarna ställs om av systemet vid inloggningen, däremot kan eventuella kommandon i filen **.profile** påverka terminalen.

3.3. Tillägg/ändring av lösenord - passwd

Varje användare har ett lösenord kopplat till sitt hembibliotek för att skydda det från obehörig användning. Eftersom systemet levereras med endast en användare upplagd, super-user, som inte har något lösenord upplagt, bör den systemansvarige lägga upp ett sådant redan vid första kontakten med systemet. Uppläggningen sker med kommandot *passwd*.

När en användare definieras är det möjligt för den systemansvarige att lägga upp ett lösenord för användaren. Men även användaren själv måste lätt kunna ändra sitt lösenord om det skulle visa sig att andra användare fått tillgång till lösenordet. Ändring av lösenordet utförs även den med kommandot *passwd*.

Ett lösenord ska vara minst sex tecken långt och bestå av minst två bokstäver samt ett numeriskt tecken eller specialtecken. Systemet frågar efter det nya lösenordet två gånger för att försäkra sig om att det är korrekt stavat. Skulle systemet inte acceptera lösenordet beror det på att det inte stavats exakt lika vid de två tillfällena. Gör ett nytt försök! Tänk på att stora och små bokstäver tolkas som olika tecken.

Tillägg/Ändring av lösenord

```
passwd
passwd: INFO: Changing password for: kalle
Old password:
New password:
Re-enter new password:
```

Observera! Lösenordet skrivs inte ut på skärmen.

En användare kan tvingas ange ett nytt lösenord efter en maximal tid i veckor och kan även hindras att själv ändra sitt lösenord. Detta styrs av koder i fältet med det krypterade lösenordet i **/etc/passwd** och kan endast definieras av den systemansvarige genom direkt editering i filen.

För användare som ska kunna logga in till systemet via en yttre modemport, finns möjlighet att definiera ett sekundärt lösenord (dial-up password). Se beskrivningen av kommandot *login* i **Referenshandboken**.

3.4. Utskrift på skrivare

Vid leverans är systemet genererat med en skrivarutgång på utgången märkt printer. Den enhet som arbetar mot skrivaren har beteckningen **/dev/lp**. För att ställa utskrifter till skrivaren i kö används programmet *lp*.

Exempel:

```
lp /etc/motd
```

Nu skrivs innehållet i filen **/etc/motd** ut på systemets standardskrivare, vilken normalt är ansluten till utgången **/dev/lp**.

Kommandot *lp* skriver till skrivaren genom ett printer-spoolersystem. lp-systemet är normalt aktiverat vid leveransen. Innan *lp*-kommandot kan användas måste emellertid följande kommando ges för att tillåta utskrifter med lp-systemet. Logga in som **root** eller **lp** för att ge detta kommando:

```
/usr/lib/accept main
```

Det är även möjligt att aktivera flera skrivare i systemet. Dessa kan till exempel kopplas till terminalportar, vilka då inte ska vara aktiverade i filen **/etc/inittab**. Seriekanaler som skall användas för skrivare, kan med kommandot */etc/mknod* döpas om med enhetsnamn som inte börjar på ttyxx, t ex **/dev/laser** eller liknande.

I kommandot *lp* kan anges som parameter vilken logisk skrivare som ska användas. Namnet på den logiska skrivaren är kopplad till en viss fysisk skrivare. Om ingen logisk skrivare anges i kommandot används den som definierats som standard i systemet, normalt **/dev/lp**.

Vid utskrift till skrivare med kommandot *lp*, behandlas texten först av ett program i en kommandofil (shell-script) innan den matas ut till skrivaren. Parametern logisk skrivare anger vilken kommandofil som ska användas.

3.5. Kopiering - copy

Innehållet i ett eller flera bibliotek kan kopieras till ett annat bibliotek eller till ett filsystem på ett annat fysiskt medium. För detta ändamål kan kommandot *copy* användas. När *copy* används bevaras trädstrukturen vid kopieringen hos det filsystem som innehåller den/de bibliotek som kopieras.

Exempel:

Kopiera alla filer i ett bibliotek till en mountad diskett.

```
copy -mrvta /usr/mydir /mf0/usr
```

copy	Kommandot självt.
-m	De filer som kopieras kommer att få samma åtkomst- och modifieringstid som originalfilerna.
-r	Alla bibliotek som påträffas kommer att genomsökas.
-v	På skärmen kommer det att skrivas ut meddelande om hur kopieringen utförs.
-t	Bevarar trädstrukturen i de bibliotek som kopieras, här **/usr/mydir**.
-a	Innan en fil kopieras måste användaren bekräfta detta genom att svara **y** (förkortning för yes).
/usr/mydir	Namnet på det bibliotek som ska kopieras.
/mf0/usr	Anger till vilket medium som biblioteket skall kopieras och under vilket bibliotek det skall sparas. I **/mf0/usr** skapas biblioteket **mydir** om det inte tidigare fanns och alla underliggande filer kopieras över.

3.6. Säkerhetskopiering, Backup

Det är viktigt att det alltid finns aktuella säkerhetskopior (backup) av alla filer.

Säkerhetskopiering bör ske regelbundet av den systemansvarige. I många fall vill dessutom användaren själv göra egna kopior av sina filer.

Kommandot *tar* spar filer till och från en yttre enhet i ett speciellt standardiserat format som är oberoende av det aktuella systemets fysiska filstruktur. Lämpliga medier är kassetter och disketter. Vid användning av *tar* måste disketterna vara formaterade enligt standard. Se kommandot *format*.

Filerna kopieras direkt till det fysiska mediet med ett block som innehåller information om filernas ursprungliga plats i biblioteksstrukturen, filaccesstillstånd etc i det filsystem från vilket kopieringen sker. Tillräcklig information för att ett senare *tar*-kommando skall kunna rekonstruera de kopierade filerna eller biblioteken. Alla eller enstaka filer och bibliotek kan läsas tillbaka till systemet vid behov.

Observera! Det rekommenderas att filer normalt kopieras utan inledande / i filnamnet. Man ställer sig istället med kommandot *cd* i det bibliotek där filerna eller biblioteken finns innan *tar*-kommandot ges. På detta sätt finns en möjlighet att läsa tillbaka biblioteken och filerna till ett temporärt bibliotek utan att förstöra de originalfiler som finns på skivminnet. Alternativt kan inledande / vid inläsning tas bort automatiskt, om tillvalet **-A** anges till *tar*-kommandot.

Vid inläsning av sparade filer till ett annat datorsystem bör användar- och gruppnamnen ändras med tillvalet **-o** till *tar*-kommandot, eftersom *tar* endast spar denna information i numerisk form.

Se även **Systemadministration** och **D-Menu Användarhandbok** för mer information om bl a "back-up"-systemet.

Om säkerhetskopieringen sker med *tar*-kommandot i bakgrunden bör utgångsstatus från *tar* testas för att säkerställa att kopieringen gick bra. Detta kan till exempel göras med kommandot *test*. Exemplet nedan visar kopiering till en QIC-kassett 60 Mb och *tar*-kommandot förklaras senare i detta kapitel. Dessa kommandon lägges lämpligen i en fil som bearbetas som ett shellprogram i bakgrunden (med **&**), varvid texten "Fel vid kopiering" ges till huvudkonsolen om kopieringen inte gick bra. Eventuella felmeddelanden lägges i den temporära filen **tmp/tartmp** för eventuell senare kontroll. De använda kommandona beskrivs i **Referenshandboken**.

Exempel:

Kopiering i bakgrunden av hela biblioteket /**usr** till kassett med *test*. Kommandona nedan är i en kommandofil som startas i bakgrunden.

```
cd /
tar -cfbk /dev/st0 200 60200 usr >tmp/tartmp 2>&1
if test $? -ne 0 ; then
     echo "Fel vid kopiering" >/dev/console
     exit 1
fi
```

Kommandot *cron* kan användas för att ge automatisk säkerhetskopiering, t ex på natten. Då kan det vara lämpligt att den skapade kopian dessutom lästestas för ytterligare kontroll genom att bearbeta följande kommandon efter säkerhetskopieringen.

Exempel:

Testkommando att lägga in efter kommandona ovan vid automatisk säkerhetskopiering.

```
tar -tf /dev/st0 >/dev/null 2>&1
if test $? -ne 0 ; then
     echo "Fel vid lästest av kopiering" >/dev/console
     exit 1
fi
```

3.6.1 Kopiering av egna filer och bibliotek till diskett

Säkerhetskopiering till diskett

En eller flera egna filer och bibliotek kan sparas på diskett med följande rutiner. En formaterad diskett sättes i diskettenheten. Eventuella data på disketten kommer att raderas.

```
tar -cvfk /dev/mf0 720 myfile
```

tar	Kommandot självt.
c	Anger att det är en ny tar-diskett som skall skapas. c(reate) = skriv till angiven enhet.
v	Skriver ut lista över vad som sker på standard output, här bildskärmen.
f	Anger att nästa argument är namnet på enheten dit kopiering ska ske.
k	Anger att nästa argument (720 i exemplet)är maximala lagringskapaciteten.
/dev/mf0	Namnet på den enhet till vilket säkerhetskopieringen skall ske.
720	Detta ska vara diskettens verkliga storlek uttryckt i kbyte.Vanliga storlekar är 720, 1200 och 1440
myfile	Bibliotek eller fil som skall kopieras. En eller flera kan anges.

Om disketten blir full ges ett meddelande på bildskärmen. varvid operatören kan byta till en ny diskett för fortsatt lagring. Vid eventuell senare inläsning ska disketterna då sättas in i samma ordning.

Innehållet på en tar-diskett kan listas på bildskärmen med följande kommando, varvid varje fil listas på en rad.

```
tar -tvf /dev/mf0
```

Inläsning av säkerhetskopia från diskett

Ett utvalt bibliotek eller en fil kan hämtas tillbaka med kommandot tar. Kommandot *cd* ges först, för att filerna skall läsas in till rätt bibliotek.

```
tar -xvf /dev/mf0 myfile
```

tar	Kommandot självt.
x	Anger att inläsning ska ske i tar-format. x = eXtract från angiven enhet och filnamn
v	Skriver ut lista över vad som sker på standard output, här bildskärmen.
f	Anger att nästa argument är namnet på enheten där biblioteket eller filen ska hämtas.
/dev/mf0	Den enhet på vilket säkerhetskopiering har gjorts.
myfile	Bibliotek eller fil som skall läsas in. Flera kan anges. Om inga filnamn ges läses alla filer in från disketten.

För ytterligare information om parametrar till *tar*, se **Referenshandboken.**

3.6.2 Kopiering av alla användarbibliotek

Samtliga användarbibliotek bör regelbundet sparas av den systemansvarige. Detta sker lämpligast till kassett om den ingår i systemet, annars till diskett.

Observera! Superuser-privilegier förutsätts för att alla filer ska kunna sparas. Inga andra användare får vara inloggade under kopieringen.

Kopiering av biblioteket usr till diskett

För att spara på diskett ges följande tar-kommando. Disketten ska vara formaterad. En eller flera disketter kan behövas varvid operatören får ett meddelande att byta diskett vid behov.

```
cd /
tar -cvfk /dev/mf0 720 usr
```

cd /	Flytta till rootbiblioteket där *usr* finns.
tar	Kommandot självt.
c	Anger att det är en ny tar-diskett som skall skapas. c = skriv till angiven enhet.
v	Skriver ut lista över vad som sker på standard output, här bildskärmen.
f	Anger att nästa argument är namnet på enheten dit kopiering ska ske.

k	Anger att nästa argument (720 i exemplet)är maximala lagringskapaciteten.
/dev/mf0	Namnet på den enhet till vilket säkerhetskopieringen skall ske.
720	Detta ska vara diskettens verkliga storlek uttryckt i kbyte. Vanliga storlekar är 720, 1200 och 1440.
usr	Biblioteket usr innehåller samtliga användarbibliotek, vilka ska kopieras.

Inläsning av alla användarbibliotek från disketten/disketterna sker med följande kommando.

```
cd /
tar -xvf /dev/mf0
```

Om flera disketter används, ska dessa läsas in i samma ordning som de skrevs.

Kopiering av biblioteket usr till kassett

För att spara på kassett ges följande kommando. Kassetten ska vara återspolad. Exemplet gäller en QIC-kassett 60 Mbyte.

```
cd /
tar -cvfbk /dev/st0 200 60200 usr
```

cd /	Flytta till rootbiblioteket där **usr** finns.
tar	Kommandot självt.
c	Anger att skrivning ska ske till början av kassetten. c = skriv till angiven enhet.
v	Skriver ut lista över vad som sker på standard output, här bildskärmen.
f	Anger att nästa argument är namnet på enheten dit kopiering ska ske.
b	Anger att nästa argument (200 i exemplet) är blockstorleken.
k	Anger att nästa argument (60200 i exemplet) är maximala lagringskapaciteten.
/dev/st0	Namnet på den enhet till vilket säkerhetskopieringen skall ske.
200	Detta ger blockstorleken vid överföringen. Värdet 200 är lämpligt för denna kassett. Som standardvärde används '20' om inget anges, vilket ger långsammare överföring.
60200	Detta ska vara kassettens verkliga storlek uttryckt i kbyte. Här 59 Mbyte = 59*1024 kbytes, vilket ger lite marginal till 60 Mbyte.
usr	Biblioteket **usr** innehåller samtliga användarbibliotek, vilka ska kopieras.

Notera att tar-kommandot ger en varningstext, för att blockstorleken överstiger 20. Detta gör ingenting, då den verkliga blockstorleken på bandet alltid blir 20 vid skrivning till en QIC-kassett. Värdet 200 används istället av *tar* som bufferstorlek för att erhålla snabb överföring av data.

Inläsning av alla användarbibliotek från kassetten sker med kommandot nedan, vilket förutsätter att hela kassetten ska läsas in.

```
cd /
tar -xvf /dev/st0
```

3.6.3 Kopiering av ett komplett system (totalkopiering)

För säkerhetskopiering av hela systemet med alla filer och bibliotek används följande *tar*-kommandon. Detta bör regelbundet göras av den systemansvarige. För totalbackup rekommenderas att en kassett används. Om disketter används för säkerhetskopiering krävs minst sex disketter för grundsystemet plus ytterligare disketter för optioner och för användarens filer.

Inläsning av ett säkerhetskopierat komplett system sker vanligen till ett tomt nyinitierat skivminne. Efter installation av skivminnet och inloggning som 'root' till root-biblioteket läses då säkerhetskopian in.

Även specialfilerna (enheterna) i biblioteket **/dev** kan säkerhetskopieras med *tar* (tillvalet **S**, dvs stora S), men notera då att vid återinläsning från säkerhetskopian kan inte redan befintliga specialfiler kopieras över.

Observera! Superuserprivilegier förutsätts och inga andra användare får vara inloggade på systemet! Bäst är att övergå till enanvändarnivå med kommandot */etc/shutdown*.

Säkerhetskopiering av hela systemet på disketter

```
cd /
tar -cvfkS /dev/mf0 720 .
```

tar	Kommandot självt.
c	Anger att det är en ny tar-diskett som skall skapas. c = skriv till angiven enhet.
v	Skriver ut lista över vad som sker på standard output, här bildskärmen.
f	Anger att nästa argument är namnet på enheten dit kopiering ska ske.
k	Anger att nästa argument (720 i exemplet) är maximala lagringskapaciteten.
S	Anger att även enheter i **/dev** ska kopieras.
/dev/mf0	Namnet på den enhet till vilket säkerhetskopieringen skall ske.
720	Detta ska vara diskettens verkliga storlek uttryckt i kbyte. Vanliga storlekar är 720, 1200 och 1440.

.	Alla filer i hela systemet kopieras. Argumentet punkt (.) tar med även filnamn som börjar med en punkt.

Inläsning av hela systemet från disketterna sker med kommandot nedan. Disketterna måste sättas in i samma ordning som de skapades. Ett meddelande ges när disketterna ska bytas.

```
tar -xvf /dev/mf0
```

Säkerhetskopiering av hela systemet till kassett

Exemplet gäller en QIC-kassett 60 Mbyte.

Kassetten ska vara återspolad. Ge följande kommando

```
cd /
tar -cvfbkS /dev/st0 200 60200 .
```

tar	Kommandot självt.
c	Anger att skrivning ska ske till början av kassetten. c = skriv till angiven enhet.
v	Skriver ut lista över vad som sker på standard output, här bildskärmen.
f	Anger att nästa argument är namnet på enheten dit kopiering ska ske.
b	Anger att nästa argument (200 i exemplet) är blockstorleken (Se kommentar nedan).
k	Anger att nästa argument (60200 i exemplet) är maximala lagringskapaciteten.
S	Anger att även enheter i **/dev** ska kopieras.
/dev/st0	Namnet på den enhet till vilket säkerhetskopieringen skall ske.
200	Detta ger blockstorleken vid överföringen. Värdet 200 är lämpligt för denna kassett. Som standardvärde används '20' om inget anges, vilket ger långsammare överföring.
60200	Detta ska vara kassettens verkliga storlek uttryckt i kbyte. Här 59 Mbyte = 59*1024 kbyte, vilket ger lite marginal till 60 Mbyte.
.	Alla filer i hela systemet kopieras. Argumentet punkt (.) tar med även filnamn som börjar med en punkt.

Notera att tar-kommandot ger en varningstext, för att blockstorleken överstiger 20. Detta gör ingenting, då den verkliga blockstorleken på bandet alltid blir 20 vid skrivning till en kassett. Värdet 200 används istället av *tar* som bufferstorlek för att erhålla snabb överföring av data.

Inläsning av hela systemet från kassetten sker med kommandot nedan.

```
tar -xvf /dev/st0
```

För ytterligare information om parametrar till *tar*, se **Referenshandboken**.

I exemplen har har vi angett värden för en QIC-kassett 60 Mb. Använder man en QIC-kassett 150 Mb anges storleken till 150000 kb.

Andra vanliga kassetter är:

DAT-kassett som får enhetsnamnet /dev/dat0 och storlek 1200000 kb

Video-8-kassett " /dev/vt0 " 1200000 kb

Anm.

I äldre handböcker används beteckningarna streamerkassett eller kassettstreamer i stället för QIC-kassett.

4

4. D-MACS

4. D-MACS

D-MACS är ett verktyg för att skapa och ändra dokument, program och andra textfiler. Det är relativt enkelt att använda för nybörjaren men är också ett kraftfullt verktyg i händerna på en expert. D-MACS kan i stor utsträckning anpassas till den individuelle användarens behov.

D-MACS tillåter editering av flera filer samtidigt. Skärmen kan delas upp i olika fönster och text kan fritt flyttas mellan de olika fönstren. För att underlätta editeringen kan D-MACS, beroende på typen av fil som editeras, uppträda på olika sätt. Editering av vanliga textfiler, programfiler samt ordbehandling av dokument kan utföras samtidigt.

Det finns omfattande möjligheter för att göra ordbehandling och editering enklare. Detta inkluderar kommandon för att söka och ersätta textsträngar, justera och radera stycken, automatisk avstavning, flytta och radera ord samt automatisk uträkning av antalet ord, rader och tecken.

För att utföra komplexa och upprepade editeringar kan du skriva s.k makron (kortkommandon). Dessa makron ger användaren stor flexibilitet när det gäller att bestämma hur D-MACS skall uppträda. Det är också möjligt att ändra den tangentkombination som används för ett givet kommando.

Speciella funktioner finns för att utföra olika operationer som t ex kryptering av filer, automatisk generering av back-up filer, exekvering av Unix-kommandon och filtrering av text genom andra program (som t ex sort för att sortera en text).

4.1. Grundläggande begrepp

4.1.1 Tangenter och tangentbordet

Vi kommer i fortsättningen av detta kapitel ofta att tala om kommandon och vilka tangenter på tangentbordet som används för dessa. Det finns ett antal "special"-tangenter som kan användas. Dessa är:

<NL>	Ny-rad som också kallas **RETURN** eller **ENTER**, denna tangent används för att avsluta olika kommandon.
CTRL-	Kontrolltangenten kan användas framför bokstavstangenter och vissa symboler. T.ex betyder **CTRL-C** att du skall hålla ned **CONTROL**-tangenten och samtidigt trycka på bokstavstangenten **C**.
CTRL-X	**CTRL-X** tangenten används i början på många kommandon.
META eller M-	Detta är en speciell D-MACS tangent vilken används för att börja många kommandon. Denna tangent skall du först trycka ned och sedan släppa upp innan du trycker ned näs-

ta tecken. På de flesta system är detta **ESC** tangenten, men det kan vara en annan.

När ett kommando skall beskrivas kommer handboken att visa de tangenter som skall användas med fetare stil tillsammans med ovanstående specialtecken samt även ange kommandots namn med kursiv stil.

4.1.2 Att komma igång

För att kunna använda D-MACS måste du anropa det från ditt systems eller din dators kommandoprompt. Detta görs genom att du skriver **dmacs** efter kommandoprompten följt av tangenten **RETURN** eller **ENTER**. Vi kommer fortsättningsvis att använda **NL** (eng. new-line) för att referera till denna tangent.

4.1.3 Utseendet på skärmen

Skärmen är uppdelad i ett antal områden eller fönster. På vissa system innehåller det översta fönstret en beskrivning över hur funktionstangenterna fungerar. Vi kommer att beskriva dessa tangenter senare. I nederkanten på varje fönster finns en D-MACS 'modrad' som talar om för dig i vilken mod editorn befinner sig -- t ex (WRAP) om du har satt D-MACS till att flytta ned texten till nästa rad då den aktuella raden tar slut. Ovanför modraden finns textfönstret där texten uppträder och manipuleras. Den sista raden på skärmen är kommandoraden där D-MACS hämtar kommandon och visar vad den håller på med.

4.1.4 Skriva in Text

Att skriva in text i D-MACS är enkelt. Skriv in följande:

```
Fang Rock lighthouse, center of a series of mysterious and
```

Texten visas längst upp i textfönstret. Fortsätt med att skriva:

```
terrifying events at the turn of the century
```

Observera att texten till vänster om markören försvinner och ersätts med ett $-tecken. Ingen panik -- din text finns fortfarande kvar ! Du har just upptäckt att **D-MACS** inte hoppar ned till nästa rad (eng. wrap) som de flesta ordbehandlare, såvida du inte trycker på **NL**. Men eftersom D-MACS används både för ordbehandling och editering av text har den en något kluven personlighet.

Du kan ändra det sätt på vilket den uppträder genom att sätta olika moder. I detta fall skall du sätta WRAP-mod genom att använda kommandot *add-mode* (tryck på **CTRL-X M**). Kommandoraden längst ned på skärmen väntar nu på att du skall skriva in vilken mod du vill lägga till. Skriv **wrap** följt av **NL** tangenten. Den text du nu skriver in kommer att flyttas ned till nästa rad då den aktuella raden tar slut. Kommandot flyttar dock inte ned text du redan skrivit in. För att få bort den avhuggna raden tar du bort tecken med **BACKSPACE** tangenten tills dess att $-tecknet försvinner. Nu kan du skriva in de ord du tog bort. Observera hur D-MACS hoppar ned till nästa rad då den aktuella raden är slut.

Låt oss nu skriva in någonting längre. Tryck på **NL** ett par gånger för att flytta ned en bit från den text du just skrivit in. Skriv följande. Tryck på **NL** två gånger för att markera nytt stycke.

```
Fang Rock lighthouse, center of a series of mysterious
and terrifying events at the turn of the century, is
built on a rocky island a few miles of the Channel
coast. So small is the island that wherever you stand
its rocks are wet with sea spray.
The lighthouse tower is in the center of the island. A
steep flight of steps leads to the heavy door in its
base. Winding stairs lead up to the crew room.
```

4.1.5 Grundläggande markörsförflyttningar

För att flytta markören tillbaka till ordet "Winding" trycker du på **M-B**, *previous-word*. Detta kommando flyttar markören bakåt ett ord åt gången. Observera att du måste använda tangentkombinationen varje gång som du vill att markören skall hoppa tillbaka ett ord. Att hela tiden hålla ned **META** och bara trycka upprepade gånger på **B** resulterar i ett felmeddelande. För att flytta framåt till ordet "stairs" trycker du på **M-F**, *next-word*, som flyttar markören framåt ett ord åt gången.

Observera att kommandon i D-MACS vanligtvis är enkla att komma ihåg, **F** för det engelska forward och **B** för backward.

För att flytta markören uppåt en rad trycker du på **CTRL-P**, *previous-line*, nedåt en rad **CTRL-N**, *next-line*. Flytta markören till ordet "terrifying" på andra raden.

Markören kan också flyttas framåt eller bakåt med mindre steg. För att flytta framåt ett tecken trycker du på **CTRL-F**, *forward-character*, och bakåt ett tecken **CTRL-B**, *backward-character*. D-MACS tillåter dig också att ange ett nummer vilket normalt används för att tala om för ett kommando att det skall utföras flera gånger.

För att upprepa de flesta kommandon trycker du på **CTRL-U** följt av ett nummer innan själva kommandot. T.ex kommer kommandot **CTRL-U 5 CTRL-F** att flytta markören framåt fem tecken. Försök att flytta runt i texten med hjälp av dessa kommandon. Som extra övning kan du försöka komma så nära som möjligt intill "small" i första stycket med hjälp av ett argument till de kommandon vi gått igenom så här långt.

Två andra enkla markörskommandon som kan vara till hjälp för att flytta runt i texten är **M-N**, *next-paragraph*, som flyttar markören till det nästa stycke och **M-P**, *previous-paragraph*, som flyttar markören till stycket innan. Markören kan också flyttas snabbt från den ena ändan av raden till den andra. Flytta markören till ordet "few" på andra raden. Tryck på **CTRL-A**, *beginning-of-line*. Observera att markören flyttar sig till ordet "events" i början på raden. Genom att trycka på **CTRL-E**, *end-of-line*, flyttar markören till slutet på raden.

Slutligen kan markören flyttas från vilken plats som helst i filen till slutet eller början på filen. Genom att trycka på **M->**, *end-of-file*, flyttar du markören till slutet på bufferten, och med **M-<**, *beginning-of-file*, flyttar du markören till första tecket i filen.

Öva på att flytta markören i texten tills du känner dig säker på de kommandon vi har beskrivit i detta avsnitt.

4.1.6 Spara texten

Din fil finns för tillfället i en *buffert*. Bufferten är en tillfällig lagringsplats för din fil vilken förloras då du stänger av datorn. Du kan spara bufferten i en fil genom att trycka på **CTRL-X CTRL-S**, *save-file*. Observera att D-MACS meddelar att filen inte har något namn och därför inte tillåter dig att spara filen.

För att spara din buffert i en fil med ett annat namn än det aktuella namnet (som är tomt), trycker du på **CTRL-X CTRL-W**, *write-file*. D-MACS kommer nu att fråga dig efter namnet på filen du vill skriva. Skriv in namnet fang.txt och tryck på **NL**. D-MACS noterar att filen håller på att skrivas. När det är klart kommer du att få reda på hur många rader som har skrivits till skivan.

4.2. Grundläggande editering

4.2.1 Fönster, buffertar, skärmar och moder

Starta D-MACS med följande kommando:

```
dmacs fang.txt
```

Kort efter det att du startat skall texten dyka upp på skärmen klar att editeras. Den text du tittar på för tillfället finns i en buffert. Modraden längst ned på skärmen visar buffertens namn, **fang.txt**, och namnet på den fil som denna buffert associeras till, *fang.txt*.

Bildskärmen har vanligtvis 24 rader med vardera 80 tecken. Du kan använda D-MACS för att dela upp skärmen i flera separata arbetsområden, eller fönster, vilka kan användas för att "titta in i" olika filer eller delar av en text. Genom att använda fönster kan du arbeta med flera texter samtidigt. Du kan enkelt kopiera och flytta textblock mellan olika fönster. För att hålla reda på vad du editerar identifieras varje fönster av en modrad på fönstrets sista rad. Denna modrad visar namnet på bufferten, namnet på filen från vilken texten har lästs in samt hur texten editeras.

En mod talar om för D-MACS hur indata från användaren skall behandlas. Som vi tidigare har sett kontrollerar moden WRAP hur D-MACS hanterar långa rader (rader med mer än 79 tecken) då användaren skriver in dem. Moden VIEW tillåter dig att läsa en fil utan möjlighet att modifiera den. Moder associeras med buffertar inte med filer, därför måste en mod sättas eller tas bort varje gång du editerar en fil. En ny fil som läses in till en buffert (med tidigare satt mod) kommer att editeras enligt denna mod. Om du ofta använder en speciell mod tillåter D-MACS dig att sätta de moder som skall gälla för alla nya buffertar, s.k. globala moder.

4.2.2 Infoga

Den tidigare sparade texten bör se ut så här:

```
Fang Rock lighthouse, center of a series of mysterious
and terrifying events at the turn of the century, is
built on a rocky island a few miles of the Channel
coast. So small is the island that wherever you stand
its rocks are wet with sea spray.
The lighthouse tower is in the center of the island. A
steep flight of steps leads to the heavy door in its
base. Winding stairs lead up to the crew room.
```

Låt oss anta att du vill lägga till en mening i det andra stycket efter ordet "base." Flytta markören tills dess att den står på "W" i "Winding". Skriv nu in följande:

```
This gives entry to the lower floor where the big steam
generator throbs steadily away, providing power for the
electric lantern.
```

Om texten inte flyttas ned till nästa rad utan ett $-tecken dyker upp i högermarginalen trycker du bara på **M-Q**, *fill-paragraph*, för att justera stycket så att texten ligger mellan marginalerna.

Observera att alla synliga D-MACS-tecken infogar sig själva. Det enda du behöver göra är att skriva in tecknen eftersom den existerande texten flyttar på sig. Med vissa undantag (som beskrivs senare) är alla icke skrivbara tecken kommandon (som kontroll- och escape-sekvenser). För att infoga blanktecken trycker du bara på mellanslagstangenten.

Flytta till den första raden i filen och tryck på **CTRL-O**, *open-line*, (O, inte noll). Du har infogat en tom rad i texten.

4.2.3 Radera

D-MACS erbjuder en mängd möjligheter till att radera delar av en text. Du kan exempelvis flytta markören så att den står på punkten i slutet av den mening du just skrev in. Tryck på tangenten **BACKSPACE**. Observera hur "n" i "lantern" försvann. Implementeringen av tangenten **BACKSPACE** i D-MACS gör att den kallas en förstörande BACKSPACE - den raderar text direkt till vänster om markören. Tryck nu på **CTRL-H**, *delete-previous-character*. Observera att markören flyttar sig till vänster och raderar "r" - båda dessa kommandon raderar tecken direkt till vänster om markören.

Skriv in de två bokstäverna du tog bort och flytta sedan markören till början av bufferten med **M->**, *beginning-of-file*. Flytta ned markören en rad till början på det första stycket.

För att ta bort det framförvarande tecknet trycker du på **CTRL-D**, *delete-next-character*. "F" i "Fang" försvinner. Fortsätt att trycka på **CTRL-D** tills hela ordet har försvunnit. D-MACS tillåter också borttagning av större textdelar. Flytta markören till ordet "center" på första raden. Genom att trycka på **M-BACKSPACE**, *delete-previous-word*, tar du bort ordet direkt till vänster om markören. **M-CTRL-H** har samma effekt.

Observera att dessa kommandon är väldigt lika de kontrollkommandon du använde för att ta bort enstaka tecken. En generell regel i D-MACS är att kontrollsekvenser påverkar små områden med text och META-sekvenser påverkar större områden. Ordet direkt till höger om markören kan tas bort med **M-D**, *delete-next-word*. Låt oss nu ta bort resterande del av första raden genom att trycka på **CTRL-K**, *kill-to-end-of-line*. Du har nu en tom rad längst upp på skärmen. Om du trycker på **CTRL-K** igen eller på **CTRL-X CTRL-O**, *delete-blank-lines*, tar du bort den tomma raden och den andra raden flyttas upp till toppen på skärmen. Avsluta nu D-MACS genom att trycka på **CTRL-X CTRL-C**, *exit-dmacs*. Observera att D-MACS påminner dig om att du inte har sparat bufferten. Ignorera detta och avsluta ändå. På detta sätt kan du avsluta D-MACS utan att spara några av de ändringar du just gjort.

4.3. Att använda områden

4.3.1 Definiera och ta bort ett område

Markering är en term i D-MACS. Tillsammans med den aktuella markörspositionen bildar markeringen ett s.k aktuellt område i vilket D-MACS utför många editeringsfunktioner.

Börja med att skriva in en ny text. Glöm inte bort att lägga till moden "wrap" om den inte redan är satt på denna buffert. Starta D-MACS och öppna en fil kallad publish.txt. Skriv in följande text:

```
One of the largest growth areas in personal computing
is electronic publishing. There are packages available
for practically every machine from elegantly simple
programs for the humble Commodore 64 to sophisticated
professional packages for PC and Macintosh computers.

Electronic publishing is as revolutionary in its way as
the Gutenburg press. Whereas the printing press allowed
the mass production and distribution of the written
word, electronic publishing puts the means of
production in the hands of nearly every individual.
From the class magazine to the corporate report,
electronic publishing is changing the way we produce
and disseminate information.

Personal publishing greatly increases the utility of
practically every computer. Thousands of people who
joined the computer revolution of this decade only to
hide their machines unused in closets have discovered a
new use for them as dedicated publishing workstations.
```

Nu skall vi editera texten. Det sista stycket verkar felplacerat. För att se hur dokumentet ser ut utan stycket tar vi ta bort det genom att flytta markören till början på stycket. Tryck på **M-SPACE**, *set-mark*, vilket sätter markeringen. D-MACS svarar med **[Mark set]**. Flytta nu markören till slutet på stycket. Du har nu definierat ett område med text. För att ta bort denna text från skärmen trycker du på **CTRL-W**, *kill-region*. Stycket försvinner från skärmen.

Det borttagna stycket skall vi i stället placera direkt efter det första stycket . Flytta markören till slutet på det första stycket och tryck på **CTRL-Y**, *yank*, som hämtar tillbaka det nyss borttagna.

Din text borde nu se ut så här:

```
One of the largest growth areas in personal computing
is electronic publishing. There are packages available
for practically every machine from elegantly simple
programs for the humble Commodore 64 to sophisticated
professional packages for PC and Macintosh computers.
```

```
Personal publishing greatly increases the utility of
practically every computer. Thousands of people who
joined the computer revolution of this decade only to
hide their machines unused in closets have discovered a
new use for them as dedicated publishing workstations.

Electronic publishing is as revolutionary in its way as
the Gutenburg press. Whereas the printing press allowed
the mass production and distribution of the written
word, electronic publishing puts the means of
production in the hands of nearly every individual.
From the class magazine to the corporate report,
electronic publishing is changing the way we produce
and disseminate information.
```

4.3.2 Hämta tillbaka ett område

Den text du tog bort försvann alltså inte ur minnet. Den flyttades till en buffert som innehåller borttagen text, kallad *kill-bufferten*. Kommandot **CTRL-Y** hämtar tillbaka, "yanks", texten från denna buffert till den aktuella bufferten. Om du har en lång rad (som indikeras med ett **$**-tecken), trycker du bara på **M-Q** för att justera stycket.

Det finns andra användningsområden för kill-bufferten. Definiera det sista stycket som en region genom att använda den metod vi nyss har beskrivit. Tryck sedan på **M-W**, *copy-region*. Inget verkar hända, markören står fortfarande på samma ställe och blinkar. Men saker har hänt även om du inte kan se någon förändring.

För att se vad som har hänt med innehållet i kill-bufferten flyttar du ned ett antal rader och använder kommandot **CTRL-Y** för att hämta tillbaka innehållet i kill-bufferten. Observera att det sista stycket upprepas. Den region du definierade har med kommandot **M-W** kopierats till kill-bufferten utan att påverka originaltexten i din arbetsbuffert. Du bör dock vara observant på att innehållet i kill-bufferten uppdateras så fort du tar bort ett område, en rad eller ett ord. Om du skall flytta stora mängder text bör du avsluta den operationen innan du gör några andra borttagningar, annars finner du kanske att den text du ville ha har ersatts av det du senast tog bort. *Kom ihåg att en buffert är ett temporärt område i datorns minne som försvinner då datorn stängs av.* För att dina ändringar skall bli permanenta måste de sparas till en fil innan du lämnar D-MACS.

Ta bort det avsnitt av text du nyss kopierade och spara filen på disken.

4.4. Sök och ersätt

4.4.1 Sökning framåt

Starta D-MACS och hämta in den fil du just sparade. Filen borde se ut så här:

```
One of the largest growth areas in personal computing
is electronic publishing. There are packages available
for practically every machine from elegantly simple
programs for the humble Commodore 64 to sophisticated
professional packages for PC and Macintosh computers.

Personal publishing greatly increases the utility of
practically every computer. Thousands of people who
joined the computer revolution of this decade only to
hide their machines unused in closets have discovered a
new use for them as dedicated publishing workstations.

Electronic publishing is as revolutionary in its way as
the Gutenburg press. Whereas the printing press allowed
the mass production and distribution of the written
word, electronic publishing puts the means of
production in the hands of nearly every individual.
From the class magazine to the corporate report,
electronic publishing is changing the way we produce
and disseminate information.
```

Vi skall söka efter "revolutionary" i det andra stycket. Eftersom D-MACS söker från aktuell markörsposition till slutet på bufferten flyttar vi markören till början av texten med **M-<**. Tryck sedan på **CTRL-S**, *search-forward*. Observera att kommandoraden nu ser ut så här

```
"Search [] <META>:"
```

D-MACS vill att du skall skriva in den sträng du ska söka efter. Skriv in ordet "revolutionary" och tryck sedan på META-tangenten. Markören flyttar nu till slutet av "revolutionary."

Observera att du måste trycka på META-tangenten för att starta sökningen. Om du trycker på **NL** kommer kommandoraden att visa NL. Detta kan verka konstigt för de användare vilka är vana vid att trycka på **RETURN**-tangenten för att exekvera kommandon. Att starta sökningen med META-tangenten ger dock möjligheter till mer noggranna sökningar. Vid varje radslut eller vagnreturtecken ser D-MACS ett nyradtecken, **NL**. Detta gör att om du vill söka efter ett ord i slutet av en rad kan du specificera detta i D-MACS.

I vår referenstext förekommer t.ex ordet "and" ett antal gånger men endast en gång i slutet på en rad. För att söka efter detta speciella "and" flyttar du markören till början på bufferten och trycker på **CTRL-S**. (Observera att D-MACS sparar den senast använda söksträngen. Detta för

att du enklare skall kunna söka efter samma ord flera gånger). Sedan skriver du in ordet **and** följt av NL. Kommandoraden visar nu:

```
search [and<NL>]<META>:
```

Trycker du nu på META-tangenten så flyttar sig markören till ordet "and" i slutet på näst sista raden.

4.4.2 Sökningar i Exact-mod

Om moden **EXACT** är aktiverad i den aktuella bufferten skiljer D-MACS på stora och små bokstäver i sökningen. Du kan skilja på orden "Publishing" och "publishing" då du söker.

4.4.3 Sökning bakåt

För att söka bakåt trycker du på **CTRL-R**, *search-reverse*. D-MACS visar då senast använda söksträng (oavsett sökriktning). Prova med att söka bakåt efter något ord som finns mellan markören och början på bufferten. Observera att när ordet har hittats placeras markören i början på ordet.

Öva nu på att söka efter andra ord i din text.

4.4.4 Sök och ersätt

Att söka och ersätta är ett kraftfullt och snabbt sätt att göra ändringar i texten. Vår referenstext handlar om "electronic publishing" men den korrekta termen är "desktop publishing". För att göra de nödvändiga ändringarna behöver vi byta ut alla förekomster av ordet "electronic" till ordet "desktop". Börja med att flytta markören till början på den aktuella bufferten med kommandot **M-<**. Sedan trycker du på **M-R**, *replace-string*. Kommandoraden visar:

```
Replace []<META>:
```

där hakparanteserna innesluter den senast använda strängen. Skriv in ordet "electronic" och tryck på META-tangenten. Kommandoraden visar:

```
with []<META>
```

skriv **desktop META**. D-MACS byter nu ut alla förekomster av originalordet till ditt ersättningsord. Du måste dock själv göra om den första bokstaven i "desktop" till stor bokstav då ordet inleder en mening.

Du har nu utfört en ovillkorlig ersättning. Detta innebär att D-MACS ersätter alla förekomster av den funna strängen med ersättningssträngen.

4.4.5 Ersättning efter fråga

Det finns också möjlighet att vid varje förekomst av ett sökord ange vad du vill göra. Kommandot **M-CTRL-R**, *query-replace-string*, gör att D-MACS stannar upp efter varje förekomst av sökströngen.

Vi vill t.ex ersätta vissa förekomster av ordet "desktop" med ordet "personal". Gå tillbaka till början av den aktuella bufferten och tryck på **M-CTRL-R**. Proceduren liknar den du tidigare använde vid ovillkorlig sök/ersätt. När sökningen börjar kommer du dock upptäcka att D-MACS

stannar upp vid varje förekomst av ordet "desktop" och frågar om du vill ersätta det med ordet "personal". Du har ett antal olika svarsmöjligheter:

Y(es)	Utför ersättningen och går vidare till nästa förekomst av sökströngen
N(o)	Utför inte ersättningen men fortsätter
!	Utför resterande ersättningar utan att stanna upp efter varje
U(ndo)	Du ångrar föregående ersättning och vill få frågan en gång till (Du kan bara gå tillbaka en ersättning)
CTRL-G	Avbryt ersättningskommandot (Detta påverkar inte redan utförda ersättningar)
.	Samma effekt som **CTRL-G** men markören går tillbaka till den plats där ersättningskommandot gavs
?	Hjälp - beskriver befintliga ersättningskommandon

Öva på att söka och ersätta tills du känner dig säker på kommandona och deras effekt.

4.5. Fönster

4.5.1 Skapa fönster

I detta avsnitt skall vi utforska en av D-MACS kraftfullare funktioner - textbehandling via flera fönster. Genom att arbeta med flera fönster och buffertar på skärmen kan du editera en fil samtidigt som du har tillgång till referenser i andra fönster.

Fönstret innehåller ett avsnitt ur en buffert vilket visas på skärmen. Eftersom D-MACS kan hantera flera fönster på skärmen samtidigt är det möjligt att samtidigit titta på olika avsnitt av samma buffert. Du kan också använda dem för att titta på text i olika buffertar. Du kan alltså editera flera filer på samma gång.

Starta D-MACS och hämta in filen om "desktop publishing". Det gör vi genom att skriva:

```
dmacs publish.txt
```

När texten har kommit upp på skärmen använder du kommandot **CTRL-X 2**, *split-window*. Fönstret delas nu upp i två fönster. Det fönster där markören finns kallas för aktuellt fönster - i detta fall det nedersta fönstret. Observera att varje fönster har ett textområde och en modrad. Kommandoraden är dock gemensam för alla fönster.

De två fönstren på skärmen är kopior av varandra eftersom det nya fönstret öppnas till samma buffert som den du stod i då du exekverade kommandot split-window. Alla kommandon du ger till D-MACS exekveras mot den aktuella bufferten i det aktuella fönstret.

För att flytta markören till det övre fönstret (d.v.s. göra det övre fönstret till aktuellt fönster) trycker du på **CTRL-X P**, *previous-window*. Observera att markören flyttar till det övre fönstret. Genom att trycka på **CTRL-X O**, *next-window*, flyttar du till fönstret nedanför. Öva på att flytta mellan fönstren. Du kommer att upptäcka att du också kan flytta till menyn för funktionstangenterna med dessa kommandon.

Ställ dig i det övre fönstret och öppna en ny fil. Den andra filen vi skrivit heter **fang.txt**. Vi hämtar in den till det övre fönstret genom att trycka på:

```
CTRL-X CTRL-F
```

följt av **RETURN**.

Skriv in filnamnet **fang.txt**.

Efter en kort stund visas filen i fönstret. Vi har nu två fönster på skärmen vilka hämtats från två olika buffertar. Kommandot vi använde var **CTRL-X CTRL-F**, *find-file*, för att söka rätt på en fil och hämta in den till aktuellt fönster.

Du kan flytta dig en sida upp eller ned i valfritt fönster med markörstangenterna eller med de kommandon du har lärt dig så här långt.

Men eftersom det synliga avsnittet text i ett fönster är relativt litet kan du endast flytta en rad åt gången upp eller ned.

Tryck på **CTRL-X CTRL-N**, *move-window-down*

Fönstret flyttar sig nedåt en rad. Den översta textraden försvinner och den understa raden flyttar sig uppåt. Du kan föreställa dig att hela fönstret sakta flyttas mot slutet av bufferten, en rad i taget. Kommandot **CTRL-X CTRL-P**, *move-window-up*, flyttar fönstret i motsatt riktning.

Som vi har sett utförs D-MACS editeringskommandon i det aktuella fönstret men du kan även flytta intilliggande fönster. Kommandot **M-CTRL-Z**, *scroll-next-up*, går upp en sida i nästa fönster. Kommandot **M-CTRL-V**, *scroll-next-down*, går en sida ned i nästa fönster. Från fönstret med filen **fang.txt** kan du nu öva på att flytta fönstret med texten om "desktop publishing" upp och ned.

När du är klar kan du avsluta D-MACS utan att spara några ändringar .

Experimentera med att dela och öppna nya fönster till olika buffertar innehållande några av dina egna filer. Prova att editera texten i varje fönster, men glöm inte bort att spara de ändringar du vill behålla. Du måste fortfarande spara varje buffert för sig.

4.5.2 Radera fönster

Kommandot **CTRL-X 0**, *delete-window*, raderar det aktuella fönstret och flyttar dig till nästa.

Om du har många öppna fönster kan du ta bort alla utom det aktuella genom att använda kommandot **CTRL-X 1**, *delete-other-windows*.

4.5.3 Ändra storlek på fönster

Det är enkelt att ändra storleken på det aktuella fönstret så att man får plats med flera rader. Ändringen visas i följande exempel:

Hämta in valfri D-MACS textfil och dela fönstret i två delar. Tryck nu på **CTRL-X ^**, *grow-window*. Ditt aktuella fönster borde vara det nedre på skärmen. Observera att fönstret har ökat sin storlek uppåt med en rad. Om du befinner dig i det övre fönstret kommer det att öka sin storlek nedåt med en rad. Kommandot **CTRL-X CTRL-Z**, *shrink-window*, minskar på motsvarande sätt det aktuella fönstrets storlek med en rad åt gången.

Det går också att ändra fönstrets storlek mer exakt genom att ange antalet önskade rader i fönstret. För att göra detta trycker du på **CTRL-U** följt av ett numeriskt argument (kom ihåg att det skall vara mindre än antalet rader på din skärm) och därefter trycker du på **CTRL-X W**, *resize-window*. Det aktuella fönstret kommer nu att ökas eller minskas till det angivna antalet rader. Om du trycker:

```
CTRL-U 8 CTRL-X W
```

kommer det aktuella fönstrets storlek att bli 8 rader.

4.5.4 Ändra placering i ett fönster

Markören kan centreras i ett fönster med kommandot **M-!** eller **M-CTRL-L**, *redraw-display*. Detta kommando är speciellt användbart när du snabbt vill lokalisera markören då du flyttar mellan olika fönster. Du kan också använda detta kommando för att flytta raden med markören till valfri position i det aktuella fönstret. Detta gör du genom att använda ett numeriskt argument före kommandot.
Tryck **CTRL-U <n> M-CTRL-L** där <n> är numret på den rad i fönstret där du vill att den aktuella raden skall visas.

Kommandot **CTRL-L**, *refresh-screen*, är användbart för att "rensa" skärmen om du t.ex använder D-MACS där systemet skickar systemmeddelanden.

4.6. Buffertar

Bufferterna är de viktigaste interna enheterna i D-MACS och den plats där alla editeringskommandon utförs. De karakteriseras av sina namn, sina moder och den fil med vilka de associeras. Varje buffert kommer också ihåg sin markering och senaste markörsposition. Du kan lämna en buffert, titta och leta i andra och återgå till den första bufferten och hamna i den exakta markörspositionen där du lämnade den.

Vana användare av D-MACS har ofta flera buffertar i datorns minne samtidigt. I det senaste avsnittet t.ex öppnade du åtminstone två buffertar - en till den text du editerade och en till fang.txt. Om du arbetar med komplexa textfiler t.ex olika avsnitt i en bok, har du kanske fem eller sex buffertar samtidigt i datorns minne. Du kan välja olika buffertar genom att helt enkelt hämta in en fil med kommandot **CTRL-X CTRL-F**, *find-file*, och låta D-MACS öppna en buffert. Men D-MACS tillhandahåller andra enklare, snabbare och mer sofistikerade möjligheter att hantera buffertar.

Låt oss börja med att öppna tre buffertar. Hämta följande filer till minnet: fang.txt, publish.txt och någon annan fil du skapat, i angiven ordning. När du är klar med det kommer du att titta på den fil du skapat själv. Anta att du nu vill flytta dig till bufferten fang.txt. Tryck på **CTRL-X X**, *next-buffer*.

Detta kommando flyttar dig till nästa buffert. Eftersom D-MACS cykliskt går igenom buffertlistan (vilken är i bokstavsordning) kommer du till bufferten fang.txt. Genom att på nytt använda **CTRL-X X** kommer du till bufferten publish.txt. Om du arbetar på en dator med stöd för funktionstangenter kommer kommandot **CTRL-X X** att placera dig i bufferten för funktionstangenterna. Genom att använda **CTRL-X X** en sista gång kommer du tillbaka till början på listan.

Detta cykliska förfarande är långsamt och opraktiskt när man arbetar med många buffertar. Kommandot **CTRL-X B**, *select- buffer*, tillåter dig att specificera den buffert du vill komma till. När du utför kommandot svarar D-MACS med **Use buffer:**. Här skriver du in namnet på den buffert du vill komma till (inte filnamnet). Den blir då aktuell buffert.

Att arbeta med flera buffertar kan vara ett ganska komplicerat förfarande och du kommer förmodligen tycka att det är besvärligt att spara varje buffert då du är klar med den. Kommandot **CTRL-X CTRL-B**, *list-buffers*, skapar ett nytt fönster som ger detaljer om de buffertar D-MACS känner till. Modifierade buffertar identifieras med indikeringen *buffer changed* (en stjärna * i andra kolumnen). Du kan på så sätt snabbt ta reda på vilka buffertar du måste spara till filer innan du avslutar D-MACS.

Fönstret med buffertlistan innehåller även annan information - specifika moder för en buffert, buffertens storlek samt namnet på bufferten. För att stänga detta fönster använder du kommandot **CTRL-X 1**, *delete-other-windows*.

För att radera en buffert trycker du på **CTRL-X K**, *delete-buffer*. D-MACS svarar **Kill buffer:**. Här skriver du in namnet på den buffert du vill ta bort. Eftersom detta är ett "förstörande" kommando kommer D-MACS be dig om en bekräftelse om bufferten var modifierad och inte sparad. Svara **Y(es)** eller **N(o)**. Som vanligt avbryter **CTRL-G** kommandot.

4.7. Moder

Med hjälp av olika moder har du möjlighet att ändra på vilket sätt D-MACS skall arbeta. Detta för att du ska kunna editera som du vill eller är van vid. Dessa moder kan påverka antingen en ensam buffert eller alla nya buffertar som skapas. För att lägga till ny mod till den aktuella bufferten använder du kommandot **CTRL-X M**, *add-mode*. D-MACS kommer då att fråga dig efter namnet på den mod du vill lägga till. När du anger ett tillåtet modnamn följt av **NL** kommer D-MACS lägga till detta modnamn till listan över aktuella moder på modraden för den aktuella bufferten. För att ta bort en existerande mod trycker du på **CTRL-X CTRL-M**, *delete-mode*, vilket resulterar i att D-MACS frågar efter namnet på den mod du vill ta bort från den aktuella bufferten. Detta tar bort moden från listan av moder på den aktuella modraden.

Globala moder är de moder vilka gäller för alla nya buffertar som skapas. Om du t.ex vill att sökning efter strängar alltid skall göras med skillnad på stora och små bokstäver skall du använda EXACT-mod. Då kommer alla nya filer som läses in att ärva moden EXACT. Globala moder sätts med kommandot **M-M**, *add-global-mode*, och tas bort med kommandot **M-CTRL-M**, *delete-global-mode*. De aktuella globala moderna visas på första raden då du utför kommandot **CTRL-X CTRL-B**, *list-buffers*.

4.7.1 ASAVE mod

Moden **ASAVE**, "spara automatiskt", talar om för D-MACS att den aktuella bufferten regelbundet skall skrivas ut till den associerade filen. Normalt sker detta för varje 256 tecken som skrivs in i filen. Environment ("miljö") variabeln **$acount** räknar ned till nästa automatiska lagring och **$asave** är det värde som tilldelas **$acount** efter varje automatisk lagring.

4.7.2 CMODE mod

CMODE är användbar vid programmering i språket C. När CMODE är aktiv försöker D-MACS assistera användaren på olika sätt. Denna mod sätts automatiskt för filer med extension **.c** eller **.h**.

Tangenten **NL** försöker flytta användaren till nästa rad med början lika långt in som på föregående rad. Detta såvida inte den aktuella raden slutar med en vänsterhake ({) vilket gör att radens början flyttas in ytterligare en tab-position.

En högerhake (}) kommer att flyttas en tab-position åt vänster då den skrivs in. Detta för att matcha dess IF, FOR eller WHILE-sats.

Om tecknet (#) står på en rad med endast mellanslag före kommer dessa att automatiskt tas bort. Detta gör att alla direktiv till förprocessorn kommer i vänstermarginalen.

När du skriver in avslutande tecken, d.v.s),], >, }, och det motsvarande inledande tecknet finns på skärmen i det aktuella fönstret kommer mar-

kören att flyttas dit och sedan tillbaka. Detta gör det enklare att balansera uttryck och matcha block.

4.7.3 CRYPT mod

Då en buffert är i **CRYPT**-mod kommer den att krypteras varje gång den skrivs till en fil och dekrypteras då den läses tillbaka från filen. Krypteringsnyckeln kan specificeras på kommandoraden med tillvalet **-k** eller med kommandot **M-E**, *set-encryption-key*. Om du försöker läsa eller skriva en buffert i krypteringsmod utan att en nyckel har angivits kommer D-MACS automatiskt att välja kommandot **sätt-krypteringsnyckel** vilket resulterar i en fråga efter den nödvändiga nyckeln. När D-MACS frågar dig efter en nyckel kommer inte ditt svar att ekas till skärmen (d.v.s. se till att du skriver rätt första gången).

Krypteringsalgoritmen som används ändrar alla tecken till normala skrivbara tecken vilket gör att du kan skicka filen via elektronisk post. Alla versioner av D-MACS skall kunna dekryptera den resulterande filen oavsett vilken dator som utförde krypteringen.

4.7.4 EXACT mod

Med **EXACT** mod kommer alla sökningar eller ersättningar av en sträng att ta hänsyn till stora och små bokstäver. Normalt har inte stora eller små bokstäver någon betydelse i strängen vid sökning eller ersättning.

4.7.5 MAGIC mod

I moden **MAGIC** får vissa tecken speciell betydelse då de används i ett sökmönster. Tillsammans är de kända som s.k. "regular expressions" och ett begränsat antal av dem kan användas i D-MACS. De ger en större flexibilitet då sökkommandot används. Men de påverkar inte det inkrementella sökkommandot. De symboler som har en speciell betydelse i moden MAGIC är: *^, $, ., &, *, [* (och *]* då de används tillsammans) och \.

Tecknen "^" och "$" fixerar sökningen till början respektive slutet av en rad. Tecknet ^ måste vara i början på söksträngen och tecknet $ måste vara i slutet annars förlorar de sin betydelse och betraktas som vilket annat tecken som helst. Om du i MAGIC-mod t.ex söker efter "t$" kommer detta att placera markören i slutet på de rader vilka avslutas med bokstaven 't'. Observera att detta inte är det samma som att söka efter "t<NL>", ett 't' följt av ett nyradtecken. Tecknet $ (och ^) matchar en position, inte ett tecken, varför markören stannar kvar i slutet på raden. Ett nyradtecken är däremot ett tecken vilket måste matchas precis som vilket annat tecken som helst. Detta innebär att markören placeras efter nyradtecknet, d v s. i början på nästa rad.

Tecknet " ." har en mycket enkel betydelse. Det matchar vilket ensamt tecken som helst förutom nyrad-tecknet. Därför skulle en sökning efter "bad.er" kunna matcha "badger", "badder" (slang) eller fram t.o.m. första 'r' i "bad error" (bad error).

Tecknet " * " används som "avslutning" vilket innebär att inget eller flera av de föregående tecknen matchar. Om det inte finns några föregående tecken saknar * speciell betydelse. Och eftersom det inte matchar nyradtecknet har * inte heller någon speciell betydelse om det föregås av tecknet för början på rad, ^, eller nyradtecknet NL.

Begreppet 'inget eller flera tecken' är viktigt. Om markören t.ex stod på raden

```
This line is missing two vowels.
```

och en sökning gjordes med "a*" kommer inte markören att flytta på sig eftersom det inte skall matcha tecknet 'a' vilket uppfyller sökvillkoren. Om du vill söka efter ett eller flera 'a' skall du söka efter "aa*" vilket matchar bokstaven 'a' följt av inget eller flera a.

Tecknet " [" indikerar början på en teckenklass. Det används på samma sätt som ' .' - "vilket tecken som helst" - men du kan välja vilka tecken du vill matcha. Teckenklassen avslutas med tecknet]. En sökning med *ba .e* skulle alltså matcha *bane, bade, bale, bate* o.s.v. Du skulle dock kunna begränsa detta till enbart *babe* och *bake* genom att söka med *ba[bk]e*. Endast ett av tecknen mellan [och] kommer att matcha. Om du skulle vilja matcha alla tecken förutom de i teckenklassen kan du placera ^ som det första tecknet. Det måste vara det första tecknet i klassen annars får det ingen speciell betydelse. En sökning med [^aeiou] kommer alltså att matcha vilket tecken som helst utom en vokal, medan en sökning med [aeiou^] kommer att matcha en av vokalerna eller ett ^.

Om du har ett antal tecken i ordningsföljd som du vill ha med i din teckenklass kan du använda ett bindestreck "-" för att ange ett område. Då kommer alltså [a-z] att matcha alla bokstäver (eller alla små bokstäver - *gemener* - om moden **EXACT** är aktiv) och [0-9a-f] kommer att matcha alla siffror eller bokstäverna 'a' t.o.m. 'f' vilka är de hexadecimala tecknen. Om ett bindestreck står i början eller i slutet av en teckenklass tolkas det bara som det tecknet.

Tecknet "&" (och) är ett ersättningstecken och representerar de tecken vilka matchade sökströngen. När det används i kommandot **M-R**, ersättsträng, eller i kommandot **M-CTRL-R**, *ersätt-efter-fråga*, kommer &-tecknet att vara ersättning för sökströngen.

Escapetecknet, "\", kan användas då du vill arbeta i MAGIC-mod men samtidigt kunna använda de tecken som ingår i s.k. "regular expressions" som vanliga tecken. Det upphäver den speciella betydelsen för tecknet. En sökning med "it\." kommer alltså att söka efter en rad med "it." och inte "it" följt av ett annat tecken. Escape-tecknet tillåter dig också att använda ^, -, och [inuti en teckenklass utan att det får någon sidoeffekt.

4.7.6 OVER mod

OVER-mod står för det engelska "overwrite", skriva över. När moden OVER används och du skriver in nya tecken kommer dessa att skriva över redan befintlig text (istället för att infoga tecknen i texten). Detta

kan vara mycket användbart då du t.ex skall göra ändringar i en tabell eller i ett diagram.

4.7.7 WRAP mod

WRAP-mod används då du skriver in kontinuerlig text. Då markören kommer till slutet på raden (normalt position 72) och du trycker på mellanslag eller **NL** kommer det sista ordet på raden att flyttas ned till början på nästa rad. När du använder detta skriver du bara in texten som en enda lång rad så infogar D-MACS automatiskt nyradtecken, **NL**, då det behövs.

Anmärkning för programmerare: D-MACS anropar den funktion som är bunden till den otillåtna tangentkombinationen **M-FNW**. Denna är normalt bunden till funktionen *wrap-word* men kan istället bindas om för att aktivera olika funktioner eller makron.

4.7.8 VIEW mod

VIEW-mod gör att inga kommandon kan användas på den aktuella bufferten. D-MACS kommer att visa ett felmeddelande och sända en varningssignal varje gång du försöker göra ändringar i en buffert i VIEW mod.

4.8. Filer

I D-MACS arbetar vi med textfiler - namngivna samlingar av text placerade på en disk (eller något annat lagringsmedium). Disk-baserade versioner av filer är endast aktiva i D-MACS när du läser till eller skriver från buffertar. Som vi redan har sett är buffertar och fysiska filer sammanlänkade med associerade filnamn. T ex är bufferten "ch7.txt" associerad med den fysiska diskfilen **ch7.txt**. Filen är vanligtvis specificerad av namnet på filen samt eventuell sökväg. Följaktligen kan du ange fullständiga filnamn i D-MACS, t ex:

```
/filbibliotek/filnamn.extension
```

Om du inte anger ett filbibliotek används aktuellt bibliotek.

Viktigt:

Spara dina buffertar till en fil (eller sätt variabeln &backup till TRUE) annars kommer allt arbete att förloras när du lämnar D-MACS. Det är därför D-MACS frågar efter ett filnamn då du försöker lämna D-MACS. D-MACS skyddar inte dina diskbaserade filer från att skrivas över då filer sparas. När du säger åt D-MACS att spara en fil till disken kommer en ny fil att skapas om den inte existerar tidigare, annars kommer den tidigare versionen att skrivas över. Den gamla versionen försvinner för alltid.

Om du inte vill skriva över de tidigare versionerna av en fil kan du ändra namnet på den associerade filen med kommandot **CTRL-X N**. När denna fil sparas till disken kommer D-MACS att skapa en ny fysisk fil under det nya namnet. Den tidigare filen finns då kvar oförändrad.

Exempel:

Läs in filen fang.txt till D-MACS och tryck på **CTRL-X N**, *change-file-name*. D-MACS svarar **name:** Skriv in ett nytt namn för filen - t.ex **new.txt** följt av ett **NL**. Filen kommer nu att sparas under det nya namnet och filbiblioteket kommer att innehålla både fang.txt och new.txt.

En annan metod är att skriva filen direkt till disken under ett nytt namn. Hämta filen **publish.txt** till D-MACS. För att spara denna fil under ett annat filnamn skall du trycka på **CTRL-X CTRL-W**, *writefile*. D-MACS kommer att uppmana **write file:**. Här skriver du in ett annat filnamn, desktop.txt. Din fil kommer nu att sparas som den fysiska filen **desktop.txt**.

Observera att i exemplen ovan förblir buffertnamnen desamma även om du har ändrat namnen på de relaterade filerna. När du senare hämtar tillbaka den fysiska filen till D-MACS kommer buffertnamnet nu att relatera till filnamnet.

Du arbetar t.ex med bufferten fang.txt relaterad till filen **fang.txt**. Ändra namnet på filen till **new.txt.** D-MACS visar att du arbetar med bufferten fang.txt relaterad till filen **new.txt**. Om du nu hämtar in filen **new.txt** till D-MACS märker du att buffertnamnet ändrats till new.txt.

Om det av någon anledning uppstår en konflikt mellan buffertnamn (om du t.ex har filer med samma namn på olika enheter) kommer

D-MACS att fråga **use buffer:** Här skriver du in ett alternativt buffertnamn om du behöver.

4.9. Textens utseende på bildskärmen

4.9.1 Flytta ned vid radslut

D-MACS är inte en ordbehandlare utan en editor med dess begränsningar. Vissa enkla formateringar kan göras även om de i de flesta fall inte påverkar den färdiga texten. Vi har redan träffat på moden WRAP som flyttar ned sista ordet på en rad när raden överskrider en viss längd (normalt 75 tecken). WRAP aktiveras med kommandot **CTRL-X M**, *add-mode*, följt av ordet wrap.

Du kan också sätta din egen "wrap"-kolumn med kommandot **CTRL-X F**, *set-fill-column*. Observera att D-MACS svarar **[Fill column is 1]**. Försök nu att skriva in någon text. Du kommer märka att det händer konstiga saker - din text flyttas ned för varje ord. Detta beror på att kommandot *set-fill-column* skall föregås av ett numeriskt argument, annars sätter D-MACS denna till första kolumnen. Då kommer all text du skriver som går förbi första kolumnen att flyttas ned vid första mellanslag.

För att återställa "wrap"-kolumnen till 72 tecken skall du trycka på **CTRL-U** följt av **72**. D-MACS kommer då att svara **Arg: 72**. Nu skall du trycka på **CTRL-X F**, *set-fill-column*, D-MACS svarar nu **[Fill column is 72]**. Slutet på dina rader kommer nu åter att flyttas ned vid samma ställe som tidigare.

4.9.2 Justera stycken

Vid editering bildas ofta stycken som innehåller olika långa rader. Med ett kommando justeras texten så man får jämna rader.

Om du befinner dig i WRAP-mod kan du justera stycken med kommandot **M-Q**, *fill-paragraph*. Detta kommando justerar det aktuella stycket mellan marginalerna. Detta kan ta lite tid (speciellt med långa stycken).

4.9.3 Ändra mellan stora/små bokstäver

I D-MACS kan man lätt ändra mellan stora och små bokstäver. (Versaler och gemener).

Skriv in följande text:

```
Throw away your typewriter and learn to use a word
processor. Word processing is relatively easy to learn
and will increase your productivity enormously. Enter
the Computer Age and find out just how much fun it can
be!!
```

De fyra första orden skall ändras till versaler. Första steget är att definiera området i texten (på samma sätt som när man raderar ett område). Sätt markeringen i början på stycket med **M-SPACE**, *set-mark*, och flytta markören till mellanslaget efter "typewriter". Tryck nu på **CTRL-X CTRL-U**, *case-region-upper*. Texten skall nu se ut så här:

```
THROW AWAY YOUR TYPEWRITER and learn to use a word
processor. Word processing is relatively easy to learn
and will increase your productivity enormously. Enter
the Computer Age and find out just how much fun it can
be!!
```

För att ändra tillbaka till små bokstäver trycker du på **CTRL-X CTRL-L**, *case-region-lower*. Du kan också ändra bokstäverna i individuella ord. För att göra om ordet "fun" till FUN placerar du markören precis framför ordet och trycker på **M-U**, *case-word-upper*. För att ändra tillbaka placerar du markören framför ordet igen och trycker på **M-L**, *case-word-lower*.

Du kan även göra om enstaka bokstäver. Kommandot **M-C**, *case-word-capitalize*, gör om första bokstaven efter markören till stor bokstav. Kommandot bör användas då markören står framför första bokstaven i ett ord. Om du använder kommandot i mitten på ett ord kan du få mycket konStig text.

4.9.4 Tabulatorer

När D-MACS startar sätts normaltab till var åttonde kolumn. Så länge du behåller detta kommer ett tab-tecken, **CTRL-I**, att infogas i texten då du trycker på tab-tangenten. Detta tecken är precis som alla andra kontrolltecken osynligt.

Om du i normalmod trycker på tab-tangenten och skriver ordet "Test" blir resultatet att ordet "Test" startar först i åttonde kolumnen. Flytta markören till början av ordet och tryck på **CTRL-H**, *delete-previous-character*. Ordet flyttas tillbaka, inte endast ett tecken utan hela vägen tillbaka till vänstermarginalen.

Orsaken till detta är enkel. Normalt infogar D-MACS ett 'riktigt' tab-tecken då du trycker på tab-tangenten. Detta tecken infogas utan några mellanslag mellan tab-tecknet och vänstermarginalen (eller föregående tab-tecken). Som du kanske kommer ihåg känner D-MACS endast igen tecken som mellanslag eller bokstäver och därför, när tabtecknet tas bort, flyttas texten tillbaka till vänstermarginalen eller föregående tab-tecken.

Du kan påverka detta förfarande genom att ändra normalinställningen. Ange ett numeriskt värde innan du trycker på tabtangenten så ändras inställningen. Som vi sett tidigare kan du genom att trycka på **CTRL-U** följt av ett numeriskt värde specificera hur D-MACS skall utföra en bestämd uppgift.

Ange argumentet 10 och tryck sedan tab-tangenten. Skriv ordet Test. Test kommer att skrivas i tionde kolumnen. Om du nu flyttar markören till början på ordet och trycker på **CTRL-H** kommer Test att flyttas tillbaka endast ett tecken.

D-MACS uppträder olika i dessa fall eftersom funktionen för hantering av tabulatorer, **CTRL-I**, behandlar dessa på två olika sätt. Normalt, eller om det numeriska argumentet är noll, kommer ett 'riktigt' tab-tecken att in-

fogas. Om däremot ett argument skilt från noll anges kommer korrekt antal mellanslag att infogas för att placera markören på nästa specificerade tab-position. Det infogar inte ett ensamt tab-tecken vilket gör att editeringsfunktionerna måste räkna med de mellanslag som då finns mellan tab-kolumnerna.

Det kan finnas tillfällen då du vill ändra en rad med tabtecken till en rad med mellanslag. Kommandot **CTRL-X CTRL-D**, *detab-line*, ändrar alla tabulatorer, från markören och till slutet på raden, till rätt antal mellanslag (radens utseende ändras inte). Detta är mycket användbart då filen måste skrivas ut eller flyttas till en dator som inte förstår tab-bar.

Det finns även möjlighet att använda det motsatta kommandot **CTRL-X CTRL-E**, *entab-lines*, vilket gör om flera mellanslag till ett tab-tecken. Detta kan vara användbart för att minska storleken på stora dokument, speciellt datatabeller. Båda dessa kommandon kan föregås av ett numeriskt argument vilket tolkas som antalet rader som skall bearbetas.

4.10. Kontakt med världen utanför D-MACS

D-MACS har möjligheter att via ett gränssnitt komma i kontakt med andra program och den omgivande datorn.

Detta sker via en serie kommandon vilka tillåter D-MACS att tala med datorns kommandoprocessor eller shell. Exakt vilket varierar mellan olika datorer. Under Unix är det **shell**.

Kommandot **CTRL-X !**, *shell-command*, frågar användaren efter en kommandorad att skicka ut till shell för exekvering. Detta kan vara mycket användbart för att lista filer eller ändra aktuellt filbibliotek. D-MACS ger kontrollen till **shell** som utför kommandot. När shell är klar skriver den **[END]** och väntar på att användaren skall trycka på en tangent. Då detta görs uppdateras skärmen och editeringen kan återupptas. Om shell-command används i ett makro blir det ingen paus.

Kommandot **CTRL-X @**, *pipe-command* tillåter D-MACS att exekvera ett shell-kommando och om datorn tillåter det skickas resultatet till en buffert vilken automatiskt visas på skärmen. Den resulterande bufferten, kallad "command", kan bearbetas precis som vilken annan vanlig buffert som helst. Text kan kopieras från den eller behandlas på vanligt sätt. Denna buffert skapas i VIEW-mod, så kom ihåg att använda **CTRL-X CTRL-M view NL** om du vill ändra i den.

Många datorer tillhandahåller verktyg för att filtrera text (vilket modifierar texten på något sätt). Ett vanligt sådant verktyg är programmet sort vilket tar emot en fil, sorterar den och skriver ut resultatet. D-MACS kommandot **CTRL-X #**, *filter-buffer*, skickar den aktuella bufferten genom ett sådant filter. Om du t.ex vill sortera den aktuella bufferten kan du alltså trycka på **CTRL-X # sort NL**. Du kan också skapa egna filter genom att skriva program vilka läser text från tangentbordet och presenterar resultatet på skärmen. D-MACS kan använda alla sådana program vilka normalt skulle varit tillgängliga från shell.

Om du skulle vilja exekvera något annat program direkt utan att gå via shell kan du använda kommandot **CTRL-X $**, *execute-program*. Du kommer då att få frågan om ett externt program och dess argument. Precis som då D-MACS letar efter kommandofiler kommer nu D-MACS att först leta efter det namngivna programmet i HOME biblioteket, sedan i den s.k. exekverings PATH och sist i det aktuella filbiblioteket. På vissa system kommer korrekt extension att automatiskt läggas till efter filnamnet för att indikera att det är ett program. På de system vilka inte stöder funktionen *execute-program* kommer **CTRL-X $** att vara detsamma som **CTRL-X !**, *shell-command*.

Ibland kanske du vill komma tillbaka till ett shell och exekvera andra kommandon utan att förlora det aktuella innehållet i D-MACS. Kommandot **CTRL-X C**, *i-shell*, lämnar D-MACS och startar en ny kommandoprocessor. De flesta system tar dig tillbaka till D-MACS med kommandot **exit**.

4.11. Tangentmakron

I många sammanhang kan det vara nödvändigt att ofta upprepa en serie tecken eller kommandon. T.ex ett dokument innehåller en ofta förekommande komplex formel eller ett långt namn. Eller du har många D-MACS-kommandon som du ofta använder. Att skapa tangentmakron är en enkel metod för att spara och repetera dessa kommandon.

Antag att du skall skriva en uppsats om Taraxacum vulgare (maskros). Även den mest hängivna botanist skulle säkert tycka att det vore bekvämt att slippa skriva Taraxacum vulgare om namnet förekom ofta i texten. Detta är möjligt om du sparar namnet i ett tangentmakro.

Kommandot **CTRL-X (**, *begin-macro*, startar sparandet av alla tangentnedtryckningar och kommandon du matar in. När du tryckt på **CTRL-X (** skriver du in Taraxacum vulgare. För att avsluta sparandet trycker du på **CTRL-X)**, *end-macro*. D-MACS har nu sparat alla tangentnedtryckningar mellan de två kommandona. För att upprepa det namn du just sparat trycker du på **CTRL-X E**, *execute-macro*, och namnet "Taraxacum vulgare" kommer fram. Du kan upprepa detta så ofta du vill och precis som med alla andra D-MACS kommandon kan det föregås av ett numeriskt argument.

Eftersom D-MACS sparar tangentnedtryckningar är det möjligt att blanda tecken och kommandon. Olyckligtvis kan du bara spara ett makro åt gången. Därför kommer det gamla makrot att försvinna så fort du börjar spara ett nytt. Se därför till att du är klar med ett makro innan du definierar ett nytt. Om du har en serie kommandon som du skulle vilja spara för framtida användning skall du använda möjligheten med makron eller procedurer vilka beskrivs i avsnitt 4.12.

4.12. D-MACS makron

Makron är program vilka används för att specialanpassa editorn och utföra komplicerade editeringar. De kan sparas i filer eller i buffertar och kan exekveras med ett lämpligt kommando eller så kan de bindas till en speciell tangentkombination. Delar av den normala uppstartningsfilen är implementerad med makron liksom menysystemexemplet. Kommandot *execute-marco-n* gör så att ett makro med ett nummer mellan 1 och 40 exekveras. Kommandot *execute-file* ger dig möjlighet att exekvera ett makro som är sparat på en diskfil. Kommandot *execute-buffer* ger dig möjlighet att exekvera ett makro lagrat i en buffert. Makron kan sparas för enklare exekvering genom att exekvera filer innehållande kommandot *store-macro*.

Om du behöver fler än 40 makron kan du använda namngivna makron, s.k. procedurer. Kommandot *store-procedure* tar ett strängargument vilket är namnet på den procedur som skall lagras. Dessa procedurer kan sedan exekveras med kommandot **M-CTRL-E**, *execute-procedure* eller *run*.

Det finns många olika aspekter på makrospråket i D-MACS. Editorkommandon är de olika kommandon som bearbetar text, buffertar, fönster o s v i editorn. Direktiv är de kommandon vilka kontrollerar vilka rader som exekveras i ett makro. Det finns också ett antal olika typer av variabler. Environment-variabler både kontrollerar och rapporterar om olika aspekter i editorn. Användarvariabler innehåller strängvärden vilka kan ändras eller kontrolleras. Buffertvariabler tillåter att text placeras i variabler. Interaktiva variabler tillåter programmet att fråga användaren om information. Funktioner kan användas för att bearbeta alla dessa variabler.

4.12.1 Konstanter

Alla konstanter och variabelinnehåll i D-MACS sparas som teckensträngar. Nummer sparas siffra för siffra som tecken. Detta tillåter D-MACS att vara typlöst, d.v.s. D-MACS har inte olika variabeltyper som är tillåtna beroende av sammanhanget. Nackdelen med detta är att användaren måste vara noggrannare i valet av variabel. Fördelen är större flexibilitet i placeringen av variabeln. Det gör också att D-MACS uttrycksbedömare kan var både snabb och konsis.

På samtliga ställen där en sats i ett makro behöver argument är det tillåtet att använda konstanter. En konstant är en sträng av tecken innanför dubbla citationstecken ("). För att representera olika specialtecken i en konstant används tilde-tecknet (~). Det tecken som följer tilde-tecknet tolkas enligt följande tabell:

~n	CTRL-J	linefeed/newline, (D-MACS nyradtecken)
~r	CTRL-M	vagnretur
~~	~	
~b	CTRL-H	backspace
~f	CTRL-L	formfeed
~t	CTRL-I	tab
~"	"	

Ett tecken som inte finns i tabellen ovan och som föregås av ett tildetecken (~) kommer att passeras utan att modifieras. Detta liknar kommandot **CTRL-Q**, *quote-character*, tillgängligt från tangentbordet.

De dubbla citationstecknen runt konstanten är inte nödvändiga om konstanten saknar mellanslag och inte råkar likna D-MACS kommandon, direktiv, variabler eller funktioner. Detta är användbart för numeriska konstanter.

4.12.2 Variabler

Variabler i D-MACS makron kan användas för att returnera värden i ett uttryck, som repetitionsräknare till editeringskommandon eller som text vilken skall infogas i buffertar eller meddelanden. Värdet på dessa variabler sätts med kommandot **CTRL-X A**, *set*. För att t.ex sätta aktuell fyll-kolumn till 64 tecken kan följande makrorad användas:

```
set $fillcol 64
```

För att infoga innehållet i **%name** vid markören i aktuell buffert kan följande makrorad användas:

```
insert-string %name
```

Environment-variabler

Dessa variabler används för att ändra olika aspekter på hur editorn skall arbeta. De kommer också att returnera aktuell inställning om de används som delar av ett uttryck. Alla variabelnamn börjar med ett dollartecken ($) och skrivs med små bokstäver.

$acount	Nedräkning av inmatade tecken till nästa spara-fil.
$asave	Antalet inmatade tecken mellan automatiskt sparande av filer i ASAVE-mod.
$backup	Gör backup på den gamla filen innan filen sparas.
$cbufname	Namn på aktuell buffert.
$ccount	Nedräkning av inmatade tecken till nästa kontrollpunkt-fil.
$cfname	Filnamn för aktuell buffert.
$cmode	Heltal innehållande mod för aktuell buffert
$csave	Antalet inmatade tecken mellan sparande av kontrollpunktfiler.
$curchar	Tecken vid aktuell markörsposition.
$curcol	Aktuell kolumn för markören i aktuell buffert.
$curline	Aktuell rad för markören i aktuell buffert.
$curwidth	Aktuellt antal kolumner som används.
$cwline	Aktuell rad i aktuellt fönster.
$debug	Flagga för att trigga makroavlusning (try it...you'll like it!)
$discmd	Flagga för att ta bort ekandet av meddelanden till kommandoraden.
$disinp	Flagga för att ta bort ekandet av inmatade tecken på kommandoraden.
$fillcol	Aktuell fyll-kolumn.
$flicker	Flagga satt till TRUE för IBM CGA, satt till FALSE för de flesta andra.
$gflags	Globala flaggor för att kontrollera vissa interna D-MACS funktioner
$gmode	Flaggor för global mod
$lastkey	[READ ONLY] Senast tryckta tangent.
$line	Den aktuella raden i den aktuella bufferten kan hämtas in och sättas med denna variabel.
$lwidth	[READ ONLY] Returnerar antalet tecken på aktuell rad.
$match	[READ ONLY] Senast matchande sträng vid sökning i magic-mod.
$overlap	Antal rader som kommer att överlappa från tidigare visad sida.
$pagelen	Aktuellt antal rader på skärmen.

$palette	Sträng som används för att kontrollera inställningen av palettregistret på grafikversioner. Det vanliga formatet består av grupper om tre oktala siffror vilka sätter nivån för röd, grön och blå.
$pending	[READ ONLY] Flagga för att avgöra om det finns tangentned tryckningar som väntar på att bli behandlade.
$progname	[READ ONLY] Innehåller alltid strängen D-MACS för standard D-MACS. Skulle kunna vara någonting annat om D-MACS ingår som en del i någon annans program.
$replace	Aktuell default ersättningssträng.
$rval	Detta innehåller returvärdet från den senaste underprocess vilken startades från D-MACS.
$search	Aktuell default söksträng.
$seed	Heltalsfrö för slumptalsgeneratorn.
$sres	Aktuell upplösning på skärmen (CGA, MONO eller EGA i IBM-PC drivern).
$status	[READ ONLY] Status för om det senaste kommandot lyckades (TRUE eller FALSE). Detta används vanligtvis med !force för att kontrollera om en sökning eller filoperation lyckades.
$target	Aktuellt mål för flyttande av rader (genom att sätta detta luras D-MACS att tro att det senaste kommandot var en radförlyttning).
$tpause	Kontrollerar längden på pausen innan ett matchande s.k. fence-tecken visas när aktuell buffert är i CMODE och ett å-tecken har skrivits in.
$version	[READ ONLY] Innehåller aktuellt versionsnummer på D-MACS.
$wline	Antal rader i aktuellt fönster.

Användarvariabler

Användarvariabler tillåter dig som användare att lagra och bearbeta strängar. Dessa strängar kan vara delar av en text, nummer (i textform) eller de logiska värdena TRUE eller FALSE. Användarvariablerna kan kombineras, testas, infogas i buffertar och på annat sätt användas för att kontrollera hur dina makron exekverar. För tillfället kan upp till 255 användarvariabler användas i en editeringssession. Alla namn på användarvariabler måste börja med ett procenttecken (%) och kan innehålla alla skrivbara tecken. Endast de tio första tecknen är signifikanta (d.v.s. olikheter efter det tionde tecknet ignoreras). De flesta operatorer kortar av strängar till en längd av 128 tecken.

Buffertvariabler

Buffertvariabler är speciella på det sättet att de bara kan kontrolleras, aldrig ändras. Buffertvariabler är ett sätt att ta text från en buffert och placera den i en variabel. Om du t.ex har en buffert med namnet **RIGEL2** innehållande följande text:

```
Richmond
Lafayette
<*>Bloomington (där <*> är markörens position)
Indianapolis
Gary
```

och i ett kommando refererar du **#rigel2** enligt följande:

```
insert-string #rigel2
```

D-MACS börjar då på den plats i RIGEL2 bufferten där markören befinner sig och tar all text till och med slutet på den raden och skickar den tillbaka. Sedan flyttas markören till början på nästa rad. Då sista komman-

dot exekverats har strängen "Bloomington" satts in i den aktuella bufferten. Bufferten RIGEL2 ser nu ut så här:

```
Richmond
Lafayette
Bloomington
<*>Indianapolis (där <*> är markörens position)
Gary
```

som du förmodligen har upptäckt består en buffertvariabel av buffertnamnet föregått av #-tecknet.

Interaktiva variabler

Interaktiva variabler är i själva verket ett sätt att fråga användaren efter en sträng. Detta utförs genom att använda ett @-tecken följt av antingen en sträng innanför citationstecken eller en variabel innehållande en sträng. Strängen placeras på den nedersta raden där editorn väntar på att användaren skall skriva in en sträng. Den inmatade strängen returneras sedan som värdet av den interaktiva variabeln. Exempel:

```
set %quest "What file? "
find-file @%quest
```

detta kommer att fråga användaren efter ett filnamn följt av ett försök att lokalisera filen. Observera att komplexa uttryck kan byggas upp med dessa operationer, som t.ex:

```
&cat &
cat "File to decode[" %default "]: "
```

vilket ställer en fråga till användaren med den sammanlänkade strängen.

4.12.3 Funktioner

Funktioner kan användas för att bearbeta variabler på olika sätt. Funktioner kan ha ett, två eller tre argument. Dessa argument skall alltid placeras efter funktionen på den aktuella kommandoraden. Om vi t.ex vill öka den aktuella fyll-kolumn med två genom att använda D-MACS kommandot **CTRL-XA**, *set*, skall vi skriva:

```
set $fillcol &add $fillcol 2
 |     |       |     |      |____andra operand
 |     |       |     |_____________första operand
 |     |       |___________________funktion att exekvera
 |     |___________________________variabel som skall sättas
 |_____________________________kommandot sätt-variabelvärde
```

Funktionsnamn börjar alltid med ett och-tecken (&) och är endast signifikant för de tre första tecknen efter och-tecknet. Funktioner förväntar sig normalt ett av tre olika argumenttyper och kommer automatiskt att konvertera typer när det behövs.

num	en ASCII-sträng med siffror vilka tolkas som ett numeriskt värde. En sträng som inte börjar med en siffra eller ett minustecken tolkas som noll.
str	En godtycklig sträng med tecken. För tillfället begränsas längden på strängen till 128 tecken.
log	Ett logiskt värde bestående av strängen TRUE eller FALSE. Numeriska strängar kommer att tolkas som FALSE om de är lika med noll och TRUE om de inte är lika med noll. Godtyckliga textsträngar har värdet FALSE.

En lista över de aktuella funktionerna följer. Funktionsnamnen kan förkortas till fyra tecken (d.v.s. **&div** för &divide) Förkortningen är markerad med fetstil.

Numeriska funktioner: (returnerar *num*)

&add*num num*	Adderar två tal
&sub *num num*	Subtraherar det andra talet från det första
&times *num num*	Multiplicerar två tal
&divide *num num*	Dividerar det första talet med det andra (ger heltal)
&mod *num num*	Returnerar resten efter division av det första talet med det andra
&negate *neg*	Multiplicerar argumentet med -1
&length *str*	Returnerar längden på strängen
&sindex str1 str2	Letar rätt på positionen för str2 i str1. Returnerar noll om inte funnet.
&ascii str	Returnerar ASCII-koden för det första tecknet i str
&rnd num	Returnerar ett slumptal mellan 1 och num (heltal).
&abs num	Returnerar absolutvärdet för num
&band num num	Bitvis AND funktion
&bor num num	Bitvis OR funktion
&bxor num num	Bitvis XOR funktion
&bnot num	Bitvis NOT funktion

Strängmanipulerande funktioner: (returnerar *str*)

&cat *str str*	Slår ihop de två strängarna till en enda sträng
&left str num	returnerar de num vänstra tecknen i str
&right str num	returnerar de num högra tecknen i str
&mid str num1 num2	Startar från position num1 i str och returnerar num2 tecken.
&upper str	gör om str till stora bokstäver
&lower str	gör om str till små bokstäver
&chr num	returnerar en sträng med tecknen representerade av ASCII-koden num
&gtk	returnerar en sträng innehållande en tangentnedtryckning från användaren
&env str	Om operativsystemet klarar av det returneras strängen associerad med str
&bind str	returnerar funktionsnamnet bundet till tangentnedtryckningen str
&env str	Returnerar operativsystemvärdet på variabeln str
&find *str*	Letar rätt på den namngivna filen *str* längs den angivna sökvägen och returnerar dess fulla filspecifikation eller en sträng om ingen existerar

Logiska testfunktioner: (returnerar *log*)

&not log	Returnerar det motsatta logiska värdet
&and log1 log2	Returnerar TRUE om båda logiska argument är TRUE
&or log1 log2	Returnerar TRUE om något av argumenten är TRUE
&equal num num	Om num och num är numeriskt lika returneras TRUE
&less num1 num2	Om num1 är mindre än num2 returneras TRUE.
&greater num1 num2	Om num1 är större än eller lika med num2 returneras TRUE.
&sequal str1 str2	Om de båda strängarna är lika returneras TRUE.
&sless *str1 str2*	Om *str1* är alfabetiskt mindre än *str2* returneras TRUE.

&sgreater *str1 str2*	Om *str1* är alfabetiskt större än eller lika med *str2* returneras TRUE.
&find *str*	Existerar den namngivna filen *str* ?

Speciella funktioner:

&indirect *str*	Behandla *str* som en variabel.

Den sista funktionen kräver lite mer förklaring. Funktionen **&ind** behandlar sitt argument och använder den resulterande strängen som variabelnamn. Med följande kodsekvens:

```
; sätt upp referenstabell

set %one "elephant"
set %two "giraffe"
set %three "donkey"
set %index "two"
insert-string
&ind %index
```

kommer strängen "giraffe" att sättas in vid markören i aktuell buffert. Detta indirekta förfarande kan nästlas upp till tio nivåer.

4.12.4 Direktiv

Direktiv är kommandon vilka endast opererar i ett bearbetande makro, d v s de har ingen funktion som fristående kommandon. De kan inte anropas fristående eller bindas till en tangentkombination. När de används i ett makro kontrollerar de vilka rader som skall exekveras och i vilken ordning.

Direktiv inleds alltid med ett utropstecken (!) och måste placeras först på en rad. Direktiv som exekveras interaktivt (via kommandot execute-command-line) kommer att ignoreras.

!ENDM

Detta direktiv används för att avsluta ett makro som skall sparas. Om t.ex en fil som exekveras innehåller texten:

```
; Read in a file in view mode

26    store-macro
      find-file @"File to view: "
      add-mode "view"
!endm
write-message "[Consult macro has been loaded]"
```

så kommer endast de rader mellan kommandot spara-makro och direktivet !ENDM att sparas i makro 26. Både numrerade makron och namngivna procedurer (via kommandot *store-procedure*) skall avslutas med detta direktiv.

!FORCE

När D-MACS exekverar ett makro och något kommando misslyckas kommer makrot att avslutas vid denna punkt. Om en rad föregås av direktivet !FORCE kommer exekveringen att fortsätta även om kommandot misslyckas. Exempel:

```
; Slå ihop de två översta fönstren

save-window                 ;kom ihåg var vi är
1 next-window               ;gå till översta fönstret
delete-window               ;slå ihop det med det andra
!force restore-window  ;fortsätter oavsett resultat
add-mode "red"
```

!IF, !ELSE, och !ENDIF

Detta direktiv tillåter att ett uttryck exekveras endast om ett villkor specificerat i direktivet uppfylls. Alla rader efter !IF-direktivet fram t.o.m. det första !ELSE- eller !ENDIF-direktivet exekveras endast om utrrycket som följer efter !IF-direktivet beräknas till värdet TRUE. Följande exempel på makrosegment skapar delar av en textfil automatiskt.

```
!if &sequal %curplace "timespace vortex"
      insert-string "First, rematerialize-n"
!endif
!if &sequal %planet "earth" ;If we have landed on earth...
      !if &sequal %time "late 20th century" ;and we are
            then write-message "Contact U.N.I.T."
      !else insert-string "Investigate the situation....-n"
            insert-string "(SAY 'stay here Sara')-n"
      !endif
!else set %conditions @"Atmosphere conditions outside? "
      !if &sequal %conditions "safe"
            insert-string &cat "Go outside......" "-n"
            insert-string "lock the door-n"
      !else insert-string "Dematerialize..try somewhen else"
            newline
      !endif
!endif
```

!GOTO

Flödet i ett D-MACS makro kan kontrolleras med hjälp av direktivet !GOTO. Det tar en "label" som argument. En label består av en rad vilken inleds med en asterisk (*) följt av en alfanumerisk label. Det är endast möjligt att hoppa till en label i det aktuella makrot. Att försöka hoppa till en icke existerande label avslutar exekveringen av makrot. Exempel:

```
;Skapa ett block med DATA-satser för ett BASIC program
      insert-string "1000 DATA "
      set %linenum 1000
*nxtin
      update-screen ;se till att vi ser ändringarna
      set %data @"Next number: "
      !if &equal %data 0
            !goto finish
      !endif
      !if &greater $curcol 60
            2 delete-previous-character
            newline
            set %linenum &add %linenum 10
            insert-string &cat %linenum " DATA "
      !endif
      insert-string &cat %data ", "
      !goto nxtin
*finish
      2 delete-previous-character
      newline
```

!WHILE och !ENDWHILE

Detta direktiv tillåter dig att repetera olika händelser enkelt och effektivt. Om en grupp av satser behöver utföras då ett speciellt villkor är uppfyllt kan du innesluta dessa satser i en slinga. Exempelvis:

```
!while &less $curcol 70
      insert-string &cat &cat "[" #stuff "]"
!endwhile
```

som placerar föremål från bufferten "item" på den aktuella raden tills dess markören står på eller har passerat kolumn 70. While-slingor kan utan problem nästlas, innehålla !GOTO-direktiv samt vara målet för !GOTO-direktiv. Att använda en while-slinga för upprepade bearbetningar blir mycket snabbare än motsvarande konstruktion med !IF-direktiv.

!BREAK

Detta direktiv tillåter användaren att gå ur den aktuella innersta while-slingan, oavsett tillstånd. Det används ofta för att avbryta bearbetningen och skriva ut felmeddelanden. Exempel:

```
; Läs in filer och ersätt "begining" med "beginning"
set %filename #list
!while &not &seq %filename "<end>"
      !force find-file %filename
      !if &seq $status FALSE
            write-message "[File read error]"
            !break
      !endif
      beginning-of-file
      replace-string "begining" "beginning"
      save-file
      set %filename #list
!endwhile
```

Denna while-slinga kommer att bearbeta filer tills dess att listan tar slut eller då ett fel uppträder vid läsning av en fil.

!RETURN

Direktivet !RETURN avslutar aktuellt makro och återvänder till den anropande (om det finns någon) eller till interaktiv mod. Exempel:

```
; Kontrollera typ av monitor och sätt %mtyp
!if &sres "CGA"
      set %mtyp 1
      !return
!else
      set %mtyp 2
!endif
insert-string "You are on a MONOCHROME machine!~n"
```

4.13. Tillval på kommandoraden

När D-MACS startar söker den alltid efter en fil kallad **.dmacsrc** vilken exekveras som D-MACS makron innan den namngivna källfilen läses in. Denna fil innehåller normalt D-MACS makron för att binda funktionstangenterna till användbara funktioner och för att ladda in olika användbara makron. Innehållet i denna fil kommer förmodligen att variera från system till system och det kan modifieras av användaren om så önskas.

När D-MACS söker efter denna fil sker det i följande ordning. Först försöker den att hitta en definition för **HOME** i omgivningen. Den kommer leta i det biblioteket först. Sedan söker den igenom alla bibliotek listade i variabeln **PATH**. Efter det söker den igenom en lista över fördefinierade standardbibliotek vilka varierar från system till system. Slutligen, om de tidigare har misslyckats, söker den i aktuellt bibliotek. Detta är samma metod som D-MACS använder för att söka rätt på filer vilka skall exekveras och för att söka rätt på hjälpfilen **dmacs.hlp**.

På datorer som anropar D-MACS via en kommandoradprocessor, som t.ex Unix, finns det olika saker som kan läggas till på kommandoraden för att kontrollera på vilket sätt D-MACS skall arbeta. Dessa kan vara olika tillval (switchar), d v s ett bindestreck ('-') följt av en bokstav, och andra möjliga parametrar eller en specifikation på startfil (vilket är ett '@'-tecken följt av et filnamn).

fil	Detta gör att den angivna filen exekveras istället för filen **.dmacsrc** innan D-MACS läser in några andra filer. Om fler än ett filnamn placeras på kommandoraden kommer de att exekveras i den ordning de uppträder.
-a	Denna flagga gör att D-MACS automatiskt exekverar startfilen **error.cmd** i stället för **.dmacsrc**. Detta används av olika C kompilatorer för felhantering (t ex Mark Williams C).
-e	De efterföljande källfilerna på kommandoraden kan editeras (i motsats till att vara i moden VIEW). Detta används huvudsakligen för att ta bort effekterna av det tidigare använda tillvalet **-v** på samma kommandorad.
-gnum	Då D-MACS startas placeras markören på den angivna raden *num* i den första filen.
-knyckel	Denna nyckel talar om för D-MACS att placera källfilen i moden CRYPT och läsa in den genom att använda *nyckel* som krypteringsnyckel. Om ingen nyckel anges efter tillvalet kommer D-MACS att fråga efter en nyckel utan att eka tecknen i nyckeln då de skrivs in.

-r	Detta placerar D-MACS i "restriktiv mod" där endast de filer angivna på kommandoraden får läsas och skrivas. De kommandon som i normala fall kan användas för kontakt med operativsystemet fungerar inte. Detta gör D-MACS väldigt användbart för "säker" användning i andra applikationer.
-s*sträng*	Då D-MACS startas kommer det automatiskt att söka efter *sträng* i den första källfilen.
-v	Detta talar om för D-MACS att efterföljande källfiler på kommandoraden skall vara i moden VIEW för att förhindra att några ändringar görs i dem.
-x	Detta talar om för D-MACS att det skall använda XON/XOFF protokoll. I detta fall kommer tecknet **CTRL-]** att ersätta **CTRL-S** och **CTRL-** kommer att ersätta **CTRL-Q**.
-n	Detta talar om för D-MACS att det inte skall använda XON/XOFF. Detta är det normala.

4.14. D-MACS kommandon

Nedan följer en komplett lista över D-MACS kommandon, de tangenter vilka normalt används för kommandot och effekten av kommandot. Kom ihåg att på vissa datorer kan det finnas fler sätt att använda ett kommando (t.ex markörstangenter och speciella funktionstangenter).

abort-command	CTRL-G	Detta tillåter att användaren avbryter ett kommando som väntar på inmatning
add-mode	CTRL-X M	Lägger till en mod till aktuell buffert
add-global-mode	M-M	Lägger till en global mod till alla nya buffertar
apropos	M-A	Listar alla kommandon vars namn innehåller den specificerade strängen
backward-character	CTRL-B	Flyttar markören ett tecken åt vänster
begin-macro	CTRL-X (	Startar sparandet av alla tangent-nedtryckningar
beginning-of-file	M-<	Flyttar markören till början på filen i aktuell buffert
beginning-of-line	CTRL-A	Flyttar markören till början på aktuell rad
bind-to-key	M-K	Bind en tangent till en funktion
buffer-position	CTRL-X =	Listar markörens position i aktuellt fönster på kommandoraden
case-region-lower	CTRL-X CTRL-L	Aktuellt område görs om till enbart små bokstäver
case-region-upper	CTRL-X CTRL-U	Aktuellt område görs om till enbart stora bokstäver
case-word-capitalize	M-C	Första bokstaven efter markören görs om till stor bokstav
case-word-lower	M-L	Följande ord görs om till små bokstäver
case-word-upper	M-U	Följande ord görs om till stora bokstäver
change-file-name	CTRL-X-N	Ändrar namnet på filen i aktuell buffert
change-screen-size	M-CTRL-S	Ändrar antalet rader på skärmen
change-screen-width	M-CTRL-T	Ändrar antalet kolumner på skärmen
clear-and-redraw	CTRL-L	Skärmen raderas helt och skrivs om
clear-message-line	Inget	Rensar kommandoraden
copy-region	M-W	Kopierar aktuellt område till killbufferten
count-words	M-CTRL-C	Räknar antalet ord, rader och tecken i aktuellt område
ctlx-prefix	CTRL-X	Ändrar tangenten vilken används som CTRL-X prefix
delete-blank-lines	CTRL-X CTRL-O	Tar bort alla tomma rader runt markören
delete-buffer	CTRL-X K	Raderar en buffert vilken inte visas i ett fönster
delete-mode	CTRL-X CTRL-M	Tar bort en mod ur aktuell buffert
delete-global-mode	M-CTRL-M	Tar bort global mod
delete-next-character	CTRL-D	Tar bort tecken direkt till höger om markören
delete-next-word	M-D	Tar bort ord direkt till höger om markören
delete-other-windows	CTRL-X 1	Raderar alla fönster förutom det aktuella fönstret
delete-previous-character	CTRL-H	Tar bort tecken direkt till vänster om markören

delete-previous-word	M-CTRL-H	Tar bort ord direkt till vänster om markören
delete-window	CTRL-X 0	Tar bort aktuellt fönster från skärmen
describe-bindings	Inget	Visar en lista över aktuella kommandon
describe-key	CTRL-X ?	Beskriver vilket kommando som är bundet till en tangentsekvens
detab-line	CTRL-X CTRL-D	Byter ut alla tab-tecken mot motsvarande antal mellanslag
end-macro	CTRL-X)	Slutar spara tangentnedtryckningar för makro
end-of-file	M->	Flyttar markören till slutet på aktuell buffert
end-of-line	CTRL-E	Flyttar markören till slutet på raden
entab-line	CTRL-X CTRL-E	Byter ut flera mellanslag mot ett tab-tecken där det är möjligt
exchange-point-and-mark	CTRL-X CTRL-X	Flytta markören till senaste markering, markera platsen markören stod på
execute-buffer	Inget	Exekvera buffert som makro
execute-command-line	Inget	Exekvera en rad skriven på kommandoraden som ett makro-komando
execute-file	Inget	Exekvera en fil som ett makro
execute-macro	CTRL-X E	Exekverar sekvensen av sparade tangentnedtryckningar
execute-macro-*n*	Inget	Exekverar makro nummer *n* där *n* är ett heltal mellan 1 och 40
execute-named-command	M-X	Exekverar namngivet kommando
execute-procedure	M-CTRL-E	Exekverar namngiven procedur
execute-program	CTRL-X $	Exekverar ett program direkt (ej via mellanliggande shell)
exit-dmacs	CTRL-X CTRL-C	Avsluta D-MACS. Om det finns förändrade buffertar vilka ej har sparats kommer D-MACS att fråga efter bekräftelse
fill-paragraph	M-Q	Justera stycke
filter-buffer	CTRL-X #	Filtrerar aktuell buffert genom ett externt filter
find-file	CTRL-X CTRL-F	Finn fil att editera i aktuellt fönster
forward-character	CTRL-F	Flyttar markören ett tecken åt höger
goto-line	M-G	Gå till angivet radnummer
goto-matching-fence	M-CTRL-F	Gå till matchande parantes
grow-window	CTRL-X ^	Gör aktuellt fönster större
handle-tab	CTRL-I	Infogar tab/sätter tabstopp
hunt-forward	Inget	Sök nästa förekomst av söksträng
hunt-backward	Inget	Sök föregående förekomst av söksträng
help	M-?	Hämtar hjälpfil till buffert
i-shell	CTRL-X C	Starta ny kommandoprocessor
incremental-search	CTRL-X S	Inkrementell sökning framåt
insert-file	CTRL-X CTRL-I	Infogar fil vid markören i aktuell fil
insert-space	CTRL-C	Infogar mellanslag till höger om markören
insert-string	Inget	Infogar en sträng vid markören
kill-paragraph	M-CTRL-W	Tar bort aktuellt stycke
kill-region	CTRL-W	Tar bort det senast markerade området och flyttar det till kill-bufferten
kill-to-end-of-line	CTRL-K	Tar bort resterande del av raden
list-buffers	CTRL-X CTRL-B	Listar alla existerande buffertar
meta-prefix	ESC	Tangent som används före alla META-kommandon

move-window-down	CTRL-X CTRL-N	Flytta rader i fönster nedåt
move-window-up	CTRL-X CTRL-P	Flytta rader i fönster uppåt
name-buffer	M-CTRL-N	Ändrar namn på aktuell buffert
newline	CTRL-M	Sätter in ett <NL> vid markören
newline-and-indent	CTRL-J	Sätter in ett <NL> vid markören och flytta in efterföljande rad lika långt som föregående rad
next-buffer	CTRL-X X	Byter till nästa buffert
next-line	CTRL-N	Flyttar markören till nästa rad
next-page	CTRL-V	Flyttar markören till nästa sida
next-paragraph	M-N	Flyttar markören till nästa stycke
next-window	CTRL-X O	Flyttar markören till nästa fönster
next-word	M-F	Flyttar markören till början av nästa ord
nop	M-FNC	Gör ingenting
open-line	CTRL-O	Sätter in tom rad vid markören
overwrite-string	Inget	Skriv över sträng vid markören
pipe-command	CTRL-X @	Exekverar ett externt kommando och placerar utdata i en buffert
previous-line	CTRL-P	Flyttar markören uppåt en rad
previous-page	CTRL-Z	Flyttar markören uppåt en sida
previous-paragraph	M-P	Flyttar markören till föregående stycke
previous-window	CTRL-X-P	Flyttar markören till föregående fönster
previous-word	M-B	Flyttar markören bakåt ett ord
query-replace-string	M-CTRL-R	Ersätter interaktivt alla förekomster av en sträng med en annan sträng
quick-exit	M-Z	Avslutar D-MACS och sparar alla förändrade buffertar
quote-character	CTRL-Q	Insert the next character literally
read-file	CTRL-X-CTRL-R	Läser in en fil till aktuell buffert
redraw-display	M-CTRL-L	Skriver om bildskärmen och centrerar den aktuella raden
resize-window	CTRL-U<n> CTRL-X-W	Ändrar antalet rader i aktuellt fönster
restore-window	Inget	Flyttar markören till det senast sparade fönstret
replace-string	M-R	Ersätter alla förekomster av en sträng med en annan sträng fr.o.m. markören och till slutet på bufferten
reverse-incremental-search	CTRL-X-R	Inkrementell sökning bakåt
run	M-CTRL-E	Exekverar en namngiven procedur
save-file	CTRL-X-CTRL-S	Sparar aktuell buffert om den har ändrats
save-window	Inget	Kommer ihåg aktuellt fönster (för att kunna återskapa senare)
scroll-next-up	M-CTRL-Z	Rullar nästa fönster uppåt
scroll-next-down	M-CTRL-V	Rullar nästa fönster nedåt
search-forward	CTRL-S	Söker efter en sträng
search-reverse	CTRL-R	Söker bakåt efter en sträng
select-buffer	CTRL-X B	Byter buffert som skall visas i aktuellt fönster
set	CTRL-X A	Sätter en variabel till ett värde
set-encryption-key	M-E	Sätt krypteringsnyckel för aktuell buffert
set-fill-column	CTRL-X F	Sätter aktuell fyll kolumn
set-mark	M-SPACE	Sätter markeringen
shell-command	CTRL-X !	Externt shell utför D-NIX kommando
shrink-window	CTRL-X CTRL-Z	Gör aktuellt fönster mindre
split-current-window	CTRL-X 2	Delar aktuellt fönster i två delar
store-macro	Inget	Sparar följande makrorader till ett numrerat makro
store-procedure	Inget	Sparar följande makrorader till en namngiven procedur
transpose-characters	CTRL-T	Byter plats på tecknet under markören och tecknet till vänster

trim-line	CTRL-X CTRL-T	Tar bort onödiga mellanslag och tabulatorer i slutet på raden
unbind-key	M-CTRL-K	Tar bort bindningen mellan en tangent och en funktion
universal-argument	CTRL-U	Exekverar efterföljande kommando fyra gånger
unmark-buffer	M-~	Tar bort ändringsflaggan i bufferten (tar bort markeringen att bufferten är ändrad)
update-screen	Inget	Kan användas i makro för att uppdatera skärmen under exekvering
view-file	CTRL-X CTRL-V	Finn fil och visa den i VIEW-mod
wrap-word	M-FNW	Flytta ned aktuellt ord. Detta är en intern funktion
write-file	CTRL-X CTRL-W	Spara aktuell buffert under nytt filnamn
write-message	Inget	Visa en sträng på kommandoraden
yank	CTRL-Y	Hämtar tillbaka kill-bufferten till markören i aktuell buffert

4.15. Tangentkombinationer för olika kommandon

Nedan följer en komplett lista över de tangentkombinationer som används i D-MACS. Denna kan användas som ett referenskort över kommandon i D-MACS.

Tangentkombinationer för D-MACS.

CTRL-A	Gå till början av raden
CTRL-B	Gå teckenvis bakåt
CTRL-C	Sätt in mellanslag
CTRL-D	Radera nästa tecken
CTRL-E	Gå till slutet av raden
CTRL-F	Gå teckenvis framåt
CTRL-G	Avbryt kommando
CTRL-H	Radera föregående tecken
CTRL-I	Sätter in tab / sätter tabstopp
CTRL-J	<vagnretur>, sedan indrag
CTRL-K	Radera resten av raden
CTRL-L	Skriv om bildskärmen
CTRL-M	Sätt in <vagnretur>
CTRL-N	Gå till nästa rad
CTRL-O	Sätt in tom rad
CTRL-P	Gå till föregående rad
CTRL-Q	Skydda nästa tecken
CTRL-R	Sök bakåt
CTRL-S	Sök framåt
CTRL-T	Byt plats på bokstäver
CTRL-U	Repetera kommando
CTRL-V	Gå till nästa sida
CTRL-W	Radera område
CTRL-Y	Hämta tillbaka radering
CTRL-Z	Gå till föregående sida
ESC A	Beskriv klass av kommandon
ESC B	Gå ord bakåt
ESC C	Inled ord med stor bokstav
ESC D	Radera nästa ord
ESC E	Sätt krypteringsnyckel
ESC F	Gå ord framåt
ESC G	Gå till rad
ESC K	Bind tangent till kommando
ESC L	Sätt små bokstäver i ord
ESC M	Lägg till global mod
ESC N	Gå till slutet av stycke
ESC P	Gå till början av stycke
ESC Q	Justera stycke
ESC R	Sök och ersätt
ESC U	Sätt stora bokstäver i ord
ESC V	Gå till föregående sida
ESC W	Kopiera område till kill-buffert
ESC X	Utför namngivet kommando
ESC Z	Spara alla buffertar och avsluta
ESC ~	Ta bort ändringsflagga i buffert
ESC !	Sätt nuvarande rad mitt på skärmen
ESC <	Gå till början av filen
ESC >	Gå till slutet av filen
ESC .	Sätt markering
ESC space	Sätter markering
ESC rubout	Ta bort föregående ord
rubout	Ta bort föregående tecken
ESC CTRL-C	Räkna ord i område
ESC CTRL-E	Exekvera namngiven procedur
ESC CTRL-F	Gå till matchande parantes
ESC CTRL-H	Radera ord bakåt
ESC CTRL-K	Ta bort tangentbindning
ESC CTRL-L	Sätt nuvarande rad mitt på skärmen

ESC CTRL-M	Radera global mod
ESC CTRL-N	Döp om aktuell buffert
ESC CTRL-R	Sök och ersätt (med frågor)
ESC CTRL-S	Ändra rader på skärmen
ESC CTRL-T	Ändra kolumner på skärmen
ESC CTRL-V	Rulla nästa fönster nedåt
Esc CTRL-W	Radera stycke
ESC CTRL-Z	Rulla nästa fönster uppåt
CTRL-X ?	Beskriv tangentkombination
CTRL-X =	Visa markörens position
CTRL-X ^	Utöka aktuellt fönster
CTRL-X 0	Ta bort aktuellt fönster
CTRL-X 1	Visa endast aktuellt fönster
CTRL-X 2	Dela aktuellt fönster
CTRL-X !	Utför D-NIX kommando
CTRL-X @	Resultat av shell-kommando till buffert
CTRL-X #	Kör buffert genom shell-filter
CTRL-X $	Exekvera externt program
CTRL-X (	Starta makro
CTRL-X)	Sluta makro
CTRL-X A	Sätter variabelvärde
CTRL-X B	Byt till annan buffert
CTRL-X C	Starta ny kommandoprocessor
CTRL-X E	Exekvera makro
CTRL-X F	Sätter fyll kolumn
CTRL-X K	Radera buffert
CTRL-X M	Lägg till mod
CTRL-X N	Byt namn på aktuell fil
CTRL-X O	Gå till nästa fönster
CTRL-X P	Gå till föregående fönster
CTRL-X R	Inkrementell sökning bakåt
CTRL-X S	Inkrementell sökning framåt
CTRL-X W	Ändra fönsterstorlek
CTRL-X X	Byt till nästa buffert
CTRL-X Z	Utöka aktuellt fönster
CTRL-X CTRL-B	Visa buffertlista
CTRL-X CTRL-C	Avsluta D-MACS
CTRL-X CTRL-D	Ersätt tab med mellanslag
CTRL-X CTRL-E	Ersätt mellanslag med tab
CTRL-X CTRL-F	Finn fil
CTRL-X CTRL-I	Infoga fil
CTRL-X CTRL-L	Sätt område små bokstäver
CTRL-X CTRL-M	Radera mod
CTRL-X CTRL-N	Flytta fönster nedåt
CTRL-X CTRL-O	Ta bort tomma rader
CTRL-X CTRL-P	Flytta fönster uppåt
CTRL-X CTRL-R	Hämta fil från skiva
CTRL-X CTRL-S	Spara aktuell fil
CTRL-X CTRL-T	Ta bort mellanslag och tabbar efter radslut
CTRL-X CTRL-U	Sätt område till stora bokstäver
CTRL-X CTRL-V	Visa fil
CTRL-X CTRL-W	Spara fil under nytt namn
CTRL-X CTRL-X	Byt markör och markering
CTRL-X CTRL-Z	Minska aktuellt fönster

Användbara moder

WRAP	Rader som går förbi högermarginalen flyttas ned och fortsätter på nästa rad
VIEW	Endast läsning, inga ändringar tillåts
CMODE	Ändra uppträdandet av vissa kommandon för att fungera bättre i språket C
EXACT	Exakt matchning av stora och små bokstäver vid sökning
OVER	Skriv över skrivna tecken i stället för infoga
CRYPT	Aktuell buffer kommer att krypteras vid skrivning och dekrypteras vid läsning
MAGIC	Använder s.k. "regular expressions" vid sökning
ASAVE	Spara filen efter vart 256:e skrivna tecken

4.16. Modflaggor

De två environment-variablerna **$cmode** och **$gmode** innehåller var-sitt nummer som motsvarar de moder som är satta för den aktuella bufferten och för hela editorn. Dessa kodas som summan av numren för varje möjlig enskild mod:

WRAP	1	Nedflyttning vid skrivning förbi radslut
CMODE	2	Funktioner anpassade för språket C
SPELL	4	Interaktiv stavningskontroll (inte implementerad)
EXACT	8	Exakt matchning vid sökning
VIEW	16	Buffert endast för läsning
OVER	32	Skriv-över mod
MAGIC	64	Använder s.k. "regular expressions" vid sökning
CRYPT	128	Krypteringsmod aktiv
ASAVE	256	Spara automatiskt

Om du t.ex vill sätta den aktuella bufferten i moderna CMODE, EXACT och MAGIC skall du summera värdet för dessa tre moder, CMODE 2 + EXACT 8 + MAGIC 64 = 74, och använda detta i ett uttryck som:

```
set $cmode 74
```

eller så använder du den binära eller-operatorn för att kombinera de olika moderna:

```
set $cmode &bor &bor 2 8 64
```

Interna flaggor

Vissa av de sätt på vilka D-MACS kontrollerar sina interna funktioner kan modifieras av värdet i variabeln **$gflags**. Varje bit i denna variabel används för att kontrollera olika funktioner.

GFFLAG	1	Om denna bit sätts till noll kommer inte D-MACS att automatiskt byta till bufferten för första filen efter exekveringen av startmakron.

5

5. Shell

5. Shell

5.1. Introduktion

Shell är både ett programmeringsspråk och ett kommandospråk men benämns ofta kommandoavkodare till D-NIX.

Shell startas normalt automatiskt när en användare loggar in till systemet enligt den information som finns i filen **/etc/passwd**. Shell laddas från filen **/bin/sh** och motsvarar den shell som i litteraturen kallas Bourne Shell.

Shell är ett kommandospråk som utgör en koppling mellan användarna och D-NIX operativsystem. I shell finns det möjligheter att styra in- och utdata, tilldela värden till variabler och strängar, och utföra substitutioner. Shell kan arbeta både i interaktiv och icke interaktiv mod.

I avsnitt 5.1 beskrivs de detaljer som en 'varje dag' användare behöver känna till, vid läsning av denna del är det en fördel om vissa baskunskaper om D-NIX finns. I avsnitt 5.2 beskrivs de grundläggande funktioner som kan användas i shell procedurer. I dessa inkluderas flödeskontrollerade primitiver och värden på strängvariabler som shell tillhandahåller. Vid läsning av detta kapitel är det en fördel om en viss kännedom om programmering finns. I resten av kapitlet beskrivs de mer avancerade möjligheterna i shell.

Observera! Detta kapitel innehåller en allmän beskrivning med exempel. För en exakt och fullständig definition av alla shell-kommandon och parametrar hänvisas till beskrivningen av kommandot */bin/sh* i **Referenshandboken.**

5.1.1 Enkla kommandon

I D-NIX skrivs kommandona som en sträng bestående av några få ord särskilda med blanktecken. Första ordet i en sträng är kommandot, resterande ord är argument till kommandot. Kommandona är enkla och korta som t ex

```
pwd
```

listar ut fullständiga namnet på aktuellt bibliotek

```
ls -l
```

listar ut alla filnamn i aktuellt bibliotek, -l är ett tillval som anger att inte bara filnamnet skall skrivas ut, utan också information om hur stor filen är, när den modifierades sist, vem som äger den etc.

För de flesta kommando som ges startar shell upp en ny process och väntar tills den har avslutas. Några kommandon är interna i shell men de flesta finns i filer med samma namn som kommandonamnet.

Ett kommando kan ges från tangentbordet när shell har skrivit ut en 'prompt' (normalt $). Det är vanligen möjligt att börja skriva kommandot redan innan prompten kommer eftersom D-NIX sparar en begränsad

mängd inmatade tecken från tangentbordet tills något program läser dem (s k "type ahead").

5.1.2 Bakgrundskommando

En del processer kräver en ganska lång exekveringstid. I D-NIX är det därför möjligt att låta dessa exekveras i bakgrunden, kommandot är *&*. Till skillnad mot de övriga kommandona skall detta kommando placeras sist på raden, på följande sätt:

```
$ cc dts.c &
```

I exemplet kommer programmet dts.c att kompileras i bakgrunden av c-kompilatorn. När processen startas upp kommer shell att skriva ut ett nummer, processnumret, på skärmen. Detta nummer är den enda identifikation som finns till processen, därför är det viktigt att komma ihåg det. Med kommandot *ps* kan en utskrift fås på skärmen över vilka processer som är aktiva just då.

Bakgrundsprocesser kan utföras med minskad prioritet, genom kommandot *nice*, för att hindra dem att fördröja vanliga förgrundsprocesser.

5.1.3 Omdirigering av data

Shell arbetar alltid mot standard output vilket i normala fall är skärmen. Men i många fall skall utdata behandlas på något sätt, innan den skrivs ut på skärmen. I stora beräkningsprogram kan t ex indata vara utdata från andra beräkningsprogram eller utdata skall sparas på en fil.

I dessa fall kan utdata omdirigeras med tecknet >,t ex.

```
$ cat > test.doc
```

Shell tolkar endast notationen > test.doc, den skickas alltså inte till kommandot som något argument. I exemplet ovan, kommer utdata att omdirigeras från skärmen till filen **test.doc**.

```
$ cat > test.doc

Denna text skrivs in på terminalen som ett exempel på
hur cat fungerar. cat kopierar denna text till en
fil. Avsluta texten med CTRL-D.
```

Filen **test.doc** kommer att innehålla den inskrivna texten. Om filen inte existerade skapas den innan utdata från *cat* skrivs in. Existerade den däremot kommer det tidigare innehållet i **test.doc** att raderas och därefter skrivs utdata från *cat* in.

Det är också möjligt att lägga till utdata till en fil som redan existerar, t ex

```
$ cat >> test.doc
```

Det tidigare innehållet i **test.doc** behålls och utdata från *cat* läggs till sist i filen.

```
$ cat > test.doc

Denna text skrivs in på terminalen som ett exempel på
hur cat fungerar. cat kopierar denna text till en
fil. Avsluta texten med CTRL-D.
```

```
$ cat >> test.doc
Detta är ett tillägg till filen test.doc.
Avsluta med CTRL-D.
```

Nu kan *cat* användas för att visa innehållet i filen.

```
$ cat test. doc
Denna text skrivs in på terminalen som ett exempel på
hur cat fungerar. cat kopierar denna text till en
fil. Avsluta texten med CTRL-D.
Detta är ett tillägg till filen test.doc.
Avsluta med CTRL-D.
```

Det är inte bara utdata som kan omdirigeras utan också indata kan omdirigeras. Normalt tas indata från standard input, tangentbordet, men den kan också tas från någon fil.

```
$ wc < test.doc
73 342 2755
```

Kommandot *wc* räknar antalet rader, ord och tecken i den angivna filen. I exemplet ovan kommer *wc* att ta indata från filen **test.doc**.

Många kommandon läser indata från filer som ges som kommandoparametrar men läser istället från standard input om ingen infil angivits. Samma sak gäller utdata. För dessa kommandon behövs normalt inte omdirigering med > eller <. T ex kan kommandot ovan även skrivas:

```
$ wc text.doc
```

Omdirigering kan även göras med andra filnummer ('file descriptors') än standard input och standard output. Ett exempel är standard error till vilken många kommandon sänder eventuella felmeddelanden. Detta är normalt till bildskärmen men kan omdirigeras genom sekvensen

```
2>file
```

Även andra in- och ut-filer kan användas i en shell-procedur genom att kommandot *exec* används. Exempel:

```
cat xyzfile 2>errfile >nyfile
```

I detta exempel kommer eventuella felmeddelanden att skrivas till errfile medan texten från filen xyxfile skrivs till nyfile.

Mer detaljer om omdirigering återfinns under kommandot *sh*.

5.1.4 Pipes och filter

I det föregående avsnittet beskrevs omdirigering av in- och utdata. Förutom detta är det möjligt att dirigera utdata från ett kommando till att bli indata till ett annat kommando, detta kallas för en pipe. Då krävs att det första kommandot verkligen sänder sina ut-data till standard output och nästa kommando läser från standard input. En pipe beteckenas med vertikalt streck, |, generell beteckning är

```
kommando1 argument | kommando2 argument
```

Exempel

```
$ ls -l | wc
```

Kommandot *wc* kommer att som indata få utdata från kommandot *ls -l*, dvs *wc* kommer att räkna antalet rader, ord och tecken som finns i utda-

ta från kommandot *ls -l*. De två kommandon sägs bilda en pipeline, effekten är den samma som om följande hade skrivits:

```
$ ls -l > test.doc
$ wc < test.doc
```

eller

```
$ ls -l > test.doc ; wc < test.doc
```

Pipes är enkelriktade och de synkroniseras genom att processen *wc* gör en paus när något skall läsas in om data saknas i pipen, och processen *ls* gör en paus när pipen temporärt är full.

Semikolon (;) separerar kommandon på samma rad ifrån varandra. Den enda skillnaden mellan en pipe och exemplet ovan är att det i en pipe inte behöver skapas en fil för att mellanlagra resultatet från *ls*. I exemplet ovan kommer processerna att exekveras en i taget, dvs först exekveras kommandot *ls -l* och utdata läggas upp på filen test.doc, därefter exekveras kommandot *wc* och indata tas från filen test.doc. När istället pipe används kommer båda processerna att exekveras parallellt.

Ett filter är en viss typ av kommando som läser från standard input, behandlar indata på ett för kommandot specifikt sätt och därefter skriver ut resultatet på standard output. Som exempel på sådana kommandon kan *fgrep* och *sort* nämnas. Filtret *fgrep* väljer från indata de rader som innehåller en specificerad sträng. Exempel:

```
$ ls -l | fgrep bak
```

skriver ut de rader som innehåller bak i utdata från *ls*, om det finns några. Det andra exemplet på ett filter är *sort*

```
$ who | sort
```

Filtret *sort* sorterar utdata från *who*, vilken består av en lista på de inloggade användarna.

En pipeline kan bestå av mer än två kommandon, t ex

```
$ who | grep putte | sort
```

Denna pipeline skriver ut de rader i *who* som innehåller ordet putte, i alfabetisk ordning.

Man kan även skapa namngivna specialfiler som fungerar som pipes, dvs skrivning sker alltid genom att lägga till data i slutet och läsning raderar alltid lästa data. Men data mellanlagras då alltid på ett skivminne. Se kommandot *mknod*.

5.1.5 Wildcards i filnamn

Shell kan generera filnamn från ofullständiga filnamn som innehåller specialtecken, dessa tecken har en speciell betydelse för shell. När shell på en kommandorad stöter på ett av dessa tecken i något argument, expanderar shell argumentet till en lista av filnamn och sänder denna vidare till kommandot som skall exekveras. Dessa specialtecken brukar kallas för wildcards eller jokertecken. Det går att hindra shell att avkoda dessa tecken som wildcards genom kommandot *set -f* eller genom att innesluta tecknen inom apostrofer '....'.

I shell finns följande olika sätt att bilda mönster på.

*	Detta tecken matchar alla strängar av tecken, inklusive nullsträngen.
?	Detta tecken matchar en ensam bokstav eller tecken.
[....]	Matchar alla tecken som står innanför tecknen [ ]. Detta är hakparenteser med internationell teckenuppsättning.
[.-.]	Ett par tecken som är åtskilda av - inom [ ] matchar alla bokstäver emellan, inklusive de angivna bokstäverna.
[!...]	Om första tecknet är ! matchas istället alla [!.-.] tecken utom de som anges inom [ ].

Observera! Filnamn som börjar med en punkt (.) matchas aldrig av någon wildcard utan måste alltid anges uttryckligen.

Specialtecken *

- matchar ingen eller flera tecken i ett filnamn, utom första tecknet om detta är en punkt.

```
$ ls test*
```

Shell behandlar test* och genererar en lista över de filer i det aktuella biblioteket som börjar på test. Om det inte finns någon fil som matchar test skrivs ett felmeddelande ut. Ett exempel på några filnamn som matchar och några som inte gör det.

Matchar	Matchar inte
testning	mtest
test	tes
test12	.test
test.brv	usr.test
testbrev	brevtest
testdir	TEST

Specialtecken ?

Frågetecknet är ett specialtecken som nästan fungerar på samma sätt som * men skillnaden är att ? bara matchar ett enda tecken i ett filnamn, utom första tecknet om detta är en punkt.

```
$ ls test?
```

Shell genererar en lista över de filnamn som finns i det aktuella biblioteket och som har namn som börjar på test och därefter följs av endast ett enda tecken. Ett exempel på några filnamn som matchar och några som inte gör det.

Matchar	Matchar inte
testa	testning
test2	test12
testb	test.brv
test8	test.1
	tests1

Specialtecken [...] , [.-.], [!...], [!.-.]

En teckengrupp omslutna av [ ] motsvarar ett tecken i ett filnamn som kan vara vilket som helst av de tecken som är skrivna innanför [ ] ([....]) eller alla tecken mellan (inklusive) tecknen ([.-.]). Med tecknet ! som första tecken inom [ ], inverteras valet genom att alla ASCII- tecken utom de som anges av gruppen tas med. Shell expanderar ett argument som innehåller en sådan teckengrupp, varje tecken i gruppen substitueras, ett i taget. Listan över de filnamn som matchar skickas av shell till det kommando som skall exekveras. Filnamn som börjar med punkt kan inte anges inom [ ] utan måste specificeras uttryckligen.

Det första exemplet nedan skriver ut alla filnamn i det aktuella biblioteket som börjar på a e i o u y. I det andra exemplet skrivs de filnamn ut som finns i det aktuella biblioteket som börjar på någon bokstav mellan m och v.

```
ls [aeiouy]*
ls [m-v]*
```

Om filnamnet istället bara består av två bokstäver, kan andra exemplet ovan skrivas på följande sätt:

```
ls [m-v]?
```

I nedanstående exempel listas alla filer som inte börjar på a eller x.

```
ls [!ax]*
```

Observera! Om det inte finns några filnamn som passar till mönstret kommer mönstret att förbli oförändrat och skickas som ett argument till kommandot.

Fördelarna med detta sätt att bilda filnamn är att det sparar in på skrivandet, vilket kan vara bra om filnamnen är långa, och att filnamn kan väljas enligt ett bestämt mönster. Dessutom är det möjligt att söka efter filer med hjälp av dessa tecken, t ex

```
$ echo /usr/pal/*/dts
```

söker efter och skriver ut alla dts filer som finns i alla bibliotek som finns under /usr/bin/pal.

Det finns ett undantag från de ovan givna reglerna. De filnamn som börjar med tecknet '.' måste matcha exakt, om följande skrivs

```
$ echo *
```

kommer alla filnamn som inte börjar på '.' att skrivas ut på standard output. För att få de filnamn som börjar med '.' utskrivna måste kommandot skrivas på följande sätt

```
$ echo .*
```

Detta undantag existerar för att inte en oavsiktlig matchning av namnen skall ske med '.' = aktuellt bibliotek, och '..' = föräldrar-bibliotek.

Observera! Kommandot *ls* undertrycker information angående filerna '.' och '..' om den inledande punkten inte anges uttryckligen.

5.1.6 Metatecken

Alla tecken som har en speciell betydelse för shell, t ex < > * ? & och vissa andra kallas för metatecken. Senare finns en fullständig lista över metatecknen. Om ett tecken föregås av \ mister detta tecken sin speciella betydelse, om det har någon.

Första exemplet nedan skriver ut ett frågetecken och andra exemplet skriver ut tecknet \.

```
$ echo \?
$ echo \\
```

Betydelsen av 'nyrad'-tecknet (return/line-feed) kan tas bort vid inmatning av en textrad varvid det är möjligt att skriva in t ex ett kommando med parametrar längre än 80 tecken, trots att texten visas på flera rader på bildskärmen. Detta görs genom att skriva \ som sista tecken på 'raden', varvid nyrad-tecknet ignoreras av shell vid avkodning av raden som ett kommando.

Detta sätt att skriva är tillämpbart då det bara gäller att ta bort specialbetydelsen för ett eller två tecken itaget, men om det ska göras för flera tecken i en sträng blir den beskrivna metoden både klumpig och tidskrävande. Istället kan apostrofer '' användas, denna metod är den enklaste och mest användbara. Sätt tecknen '' på varsin sida av strängen, t ex

```
$ echo xx'******'xx
```

Strängen xx******xx skrivs ut på standard output. Denna sträng får innehålla andra metatecken utom apostrof ' som är undantagen från denna regel. Tecknet \ har dessutom ofta speciell betydelse för olika kommandon, bl a för *echo*, varför det ska användas varsamt även inom ''.

Det finns också en tredje metod, citationstecknen " " sätts på varsin sida om strängen. Men dessa tecken kan inte ta hand om alla metatecken. Se kommandot *sh* i **Referenshandboken** för närmare detaljer.

5.1.7 Prompt

När en terminal används som standard input kommer shell att skriva ut en prompt innan den kan ta emot ett kommando. Shell ger som standardvärde prompten $. Denna kan ändras genom att variabeln **PS1** tilldelas ett nyttvärde. I nedanstående exempel visas prompten såsom den syns på bildskärmen. T ex

```
$ date
Fri May 03 10:43:13 PST 1985
$ PS1='hej på dej!'
hej på dej! date
Fri May 03 10:44:23 PST 1985
hej på dej!
.
```

```
hej på dej!
```

på skärmen kommer **hej på dej!** att skrivas ut istället för **$**. Prompten **#** indikerar att användaren för tillfället befinner sig i super-user mod.

5.1.8 Shell och login

Innan användaren loggar in kan ett allmänt systemmeddelande visas, ur filen **/etc/issue**, eventuellt följt av ett systemnamn. Efter inloggningen visas ytterligare ett meddelande (aktuellt för dagen) ur filen **/etc/motd**.

Därefter startas normalt en shell för att läsa och bearbeta de kommandon som skrivs på standard input, normalt terminalen. Om användarens bibliotek innehåller filen **.profile**, kommer shell att först läsa **.profile** eftersom denna fil förmodas innehålla diverse kommandon som är av betydelse i detta läge t ex bildskärmstyp, specialtangenter på tangentbordet. Först därefter kommer shell att gå till standard input och läsa kommandon.

5.1.9 Sammanfattning

Orden i en kommandosträng särskiljs med blanktecken.

& är kommandot för att en process skall bearbetas i bakgrunden. Detta tecken skrivs då sist på raden.

ls **> fil**

Utdata från *ls* omdirigeras från standard output till att skrivas på fil.

ls **>> fil**

Utdata från *ls* läggas till fil, utdata kommer att läggas till i slutet av filen dvs efter det tidigare innehållet i filen.

wc **< fil**

Kommandot *wc* får indata från filen fil istället för ifrån standard input.

ls **|** *wc* **-l**

Tecknet ö är en pipe, vilket innebär att utdata från *ls* är indata till *wc*. Resultatet av kommandoraden skrivs på standard output och innehåller information om antalet rader som ingår i aktuellt bibliotek.

ls **|** *fgrep* **old**

Resultatet av kommandoraden skrivs ut på standard output och innehåller information om vilka filnamn som innehåller strängen old.

ls | *fgrep* **old** | *wc* **-l**

Resultatet från kommandoraden skrivs ut på standard output och innehåller information om hur många filer i detta aktuella bibliotek som strängen old ingår i.

cc **pgm.c &**

Kompilerar programmet pgm.c i bakgrunden.

Wildcards

*	Matchar alla strängar inklusive nollsträngen.
?	Matchar ett tecken, vilket som helst.
[...]	Matchar alla inneslutna tecken.
[.-.]	Matchar alla tecken mellan och inklusive de angivna.
[!...]	Matchar alla tecken utom de inneslutna.
[!.-.]	Matchar alla tecken utom de som matchas av [.-.].

Observera! En punkt som första tecken i ett filnamn matchas aldrig av en wildcard.

5.2. Shell procedurer

Shell kan inte bara bearbeta enkla kommandon utan också läsa och bearbeta kommandosträngar som ligger i en fil. När kommandot *sh* används behöver användaren inte ha bearbetningstillstånd till filen. Exempel:

```
$ sh fil argument
```

sh startar en ny shell och anger att kommandon skall läsas från filen fil. Denna fil kallas för kommandoprocedur eller shellprocedur. Argumenten kan inkluderas i anropet, i dessa fall refererar argumenten till de sk positionsparametrarna, **$1**, **$2**,.., i filen, t ex antag att filen **drs** innehåller proceduren

```
who | fgrep $1
```

skriv på terminalen

```
$ sh drs putte
```

I kommandot i filen **drs** ersätts **$1** med första positionsparametern som i detta fall är putte. Exemplet ovan är likvärdigt med

```
$ who | fgrep putte
```

En fil kan i sig själv vara bearbetningsbar om 'rätt' åtkomstprivilegier anges. I D-NIX finns det tre olika privilegier: läsbar, skrivbar och bearbetningsbar. Varje fil kan ha en, två eller alla tre av dessa privilegier angivna samtidigt. Dessutom kan olika privilegier anges för ägaren, gruppen och speciellt för alla övriga användare. Privilegier som en fil har kan ändras med kommandot chmod. I exemplen nedan ändras privilegier för alla.

Om privilegiet för filen **drs** ändras till att vara bearbetningsbar

```
$ chmod +x drs
```

kommer uttrycket

```
$ drs putte
```

att vara likvärdigt med

```
$ sh drs putte
```

Denna funktion tillåter att program- och shellprocedurer används blandat. I båda fallen kommer en ny process att skapas för varje nytt kommando som skall bearbetas. Denna process avslutas normalt då kommandot eller shellproceduren är avslutad. Jämför kommandot *exec*, vilket istället gör att nuvarande shell-process byts ut mot en ny.

5.2.1 Flödeskontroll - for do done

En shellprocedur används ofta för att genomlöpa en loop av argument (**$1**,**$2**,**$3**,...) där de angivna kommandona exekveras för var och en av argumenten. Ett exempel på en sådan procedur är proceduren telefon som genomsöker filen /usr/lib/telnr, filen innehåller namn och telefonnummer. *cat* används nedan för att se filens innehåll.

```
$  cat /usr/lib/telnr
.
```

```
.....................
Eva Karlsson 675787
Putte Engby 124848
.....................
.
```

Texten i proceduren telefon är följande.

```
$ cat telefon
for i
do
      fgrep -i "$i" /usr/lib/telnr
done
```

Om filen telefon har exekveringsprivilegier satta kan telefonnummer visas som i exemplet nedan.

```
$ telefon putte
Putte Engby 124848
```

Kommandot telefon skriver ut alla rader i filen /usr/lib/telnr som innehåller ordet putte. Optionen **-i** till *fgrep* anger att stora och små bokstäver behandlas lika.

Den generella formen för en for-loop är:

```
for name in w1 w2 w3 w4 ......
      do kommandolista
      done
```

name måste bestå av en shellvariabel som sätts till respektive w1 w2 w3 w4 .... varje gång loppen med kommandolista och do exekveras. Om 'in w1 w2 w3 w4 .....' utesluts kommer loopen att exekveras en gång för varje positionsparameter, dvs shell antar att 'in w1 w2 w3.....' är 'in $1 $2 $3 ....' Kommandolista kan innehålla en eller flera kommandon separerade eller avslutade antingen med semikolon eller med newline. Orden do och done är reserverade ord och känns endast igen efter semikolon eller på en ny rad.

Ytterligare ett exempel på hur *for* kan användas är kommandot create,

```
$ cat create
for i
      do > $i ; done
```

Prova kommandot med:

```
$ create fil1 fil2
```

Kommandot create skapar för varje angivet argument, i det här fallet fil1 och fil2, två tomma filer. Om de fanns tidigare försvinner nu eventuellt gammalt innehåll.

Observera! Om done står på samma rad som kommandot måste ett semikolon föregå done.

5.2.2 Flödeskontroll - case esac

Case är en villkorssats som tar hand om de fall när flera olika händelser inträffar, exemplet nedan visar ett kommando som vi kallar append.

```
$ cat append
case $# in
      1) cat >> $1 ;;
      2) cat >> $2 < $1 ;;
      *) echo 'syntax: append (från) till' ;;
esac
```

Om kommandot append endast skrivs med ett argument, som t ex

```
$ append test.doc
```

inträffar punkt 1) i kommandot append, dvs positionsparametern $# kommer att ersättas med strängen 1, vilket medför att text från standard input kommer att läggas till sist i filen test.doc. I detta fall läses standard input från tangentbordet, och avslutas med tangenten **CTRL-D**. Om kommandot skrivs med två argument som t ex

```
$ append test.doc read.brv
```

inträffar punkt 2) i kommandot append, vilket innebär att innehållet i test.doc läggs till sist i filen read.brv. Om inget eller fler än två argument anges skrivs raden *) ut.

Den generella syntaxen för case är:

```
case ord in
      mönster1) kommandolista ;;
      mönster2) kommandolista ;;
      mönster3) kommandolista ;;
      .
esac
```

Varje kommandolista måste avslutas med två semikolon (;;).

Shell försöker att jämföra värdet av shellvariabeln ord med varje mönster, i den ordning som mönster förekommer. Om shell finner ett mönster som motsvarar ord, kommer kommandolistan att exekveras.

Observera! Det finns ingen kontroll på att bara ett mönster passar, utan det första mönster som case hittar kommer att exekveras. I exemplet nedan kommer aldrig kommandona efter den andra *) att utföras.

```
case $# in
      *) ... ;;
      *) ... ;;
esac
```

Ett annat användningsområde för case är att urskilja olika former av argument. Nedanstående exempel är ett fragment av ett *cc* kommando.

```
for i ; do
case $i in
      -[ocs]) ... ;;
      -*) echo 'okänd flagga '$i ;;
      *.c)  /lib/c0 $i ........ ;;
      *) echo 'oväntat argument '$i ;;
esac; done
```

Om ett kommando passar in på flera olika mönster, använd tecknet | till att separera de olika mönstren. [...] kan även användas. Kommandot *case* kan antingen skrivas som

```
case $i in
      -x | -y) ..... ;;
esac
```

eller som

```
case $i in
     -[xy]) ..... ;;
esac
```

De notationskonventioner för metatecken som nämnts tidigare gäller även för kommandot case:

```
case $i in
     \?) ....
esac
```

matchar tecknet ?.

5.2.3 Here dokument - << <<-

Tidigare använde vi filen telnr till att förse kommandot *fgrep* med data. Istället för att data ligger i en separat fil kan data innefattas i shellproceduren, detta kallas för ett 'here dokument', t ex

```
$ cat telefon
     for i
          do fgrep "$i" <<+
          ...
          Eva Karlsson 675787
          Putte Engby 124848
          ...
          +
          done
```

Shell kommer i ovanstående exempel att ta all text som står mellan <<+ och + som standard input för *fgrep*. Notera att + måste stå i början av en rad. Istället för + kan ett valfritt ord användas för att markera början och slutet av here-dokumentet.

Innan parametrarna i dokumentet är åtkomliga för *fgrep* kommer de att substitueras, vilket visas i nedanstående exempel. Detta kan förhindras genom att låta något tecken i begränsningsordet föregås av \.

Exempel på substituering i ett kommando i filen edg:

```
$ cat edg

ed $3 <<+
g/$1/s//$2/g
w
+
```

Om innehållet i den exekverbara strängen edg är enligt ovanstående så är kommandot

```
$ edg string1 string2 fil
```

likvärdigt med kommandot

```
$ ed fil <<+

g/string1/s//string2/g
w
+
```

som ändrar alla förekomster av string1 i fil till string2.

5.2.4 Shellvariabler

I shell kan strängvariabler tilldelas ett värde. Namnen på strängvariablerna skall alltid börja med en bokstav, resterande tecken kan bestå av bokstäver, siffror och understrykningstecken. En variabel kan tilldelas ett värde på tre olika sätt, se exempel

```
$ anv=kurt csmn=m000 dir=mh000
```

Variablerna anv, csmn och dir tilldelas respektive värde. En variabel kan sättas till nollsträngen på följande sätt. Notera att variabeln fortfarande är definierad i shell.

```
$ null=
```

För att helt ta bort en variabeldefinition används kommandot *unset.*

```
$ unset csmn
```

När en variabel skall substitueras med sitt värde skall tecknet $ anges före variabelnamnet, på följande sätt

```
$ echo $anv
```

Variabeln anv kommer att substitueras mot värdet av denna variabel dvs kurt och kommandot *echo* visar värdet på bildskärmen.

För att underlätta skrivandet kan förkortningar användas för strängar som förekommer ofta, t ex

```
$ b=/usr/kalle/bin
$ mv pgm $b
```

Kommandot *mv* kommer att flytta filen **pgm** till biblioteket **$b** d v s /usr/kalle/bin.

En generell beteckning för variabelsubstitution är

```
echo ${anv}
```

I detta enkla fall är ovanstående likvärdigt med:

```
echo $anv
```

Beteckningen med { } måste användas när ett variabelnamn följs direkt av en bokstav eller en siffra, vilken annars skulle tagits som en del av variabelnamnet. Till exempel

```
$ tmp=/tmp/ps
$ ps -a > ${tmp}fil
```

Utdata från *ps* kommer att skrivas till filen /tmp/psfil.

Om det istället hade stått

```
$ ps -a > $tmpfil
```

hade varibeln tmpfil substituerats med sitt värde.

Alla variabler som räknas upp här nedan sätts då shell startas utom $?, denna parameter sätts först efter exekvering av ett kommando.

$*	Hela parametersträngen med alla parametrar utom $0
$0	Kommandot själv
$1 etc	Positions-parametrarna i kommandosträngen. Jämför shell-kommandot *shift* som skif-

tar innehållet i positionsparametrarna ett eller flera steg. *shift* beskrivs tillsammans med kommandot *while*.

$# Ger antalet positionsparametrar i decimal form. Se t ex exemplet 'append' för shell-kommandot *case*.

$@ Samma som $*, men behandlas annorlunda inom citationstecken "...".

$? Det sist exekverade kommandots utgångsstatus (returnerad kod), i decimal form. Den returnerade utgångsstatusen från de flesta kommandon är om kommandot lyckas noll och om det misslyckas icke-noll. Se vidare shell-kommandona if och while.

$$ Anger processnumret för det shell som är aktuellt, i decimal form. Eftersom varje process har sitt unika nummer används $$ ofta till att skapa temporära filer, exempelvis

```
ps -a > /tmp/ps$$
...
rm /tmp/ps$$
```

$! Anger processnumret på den sista process som exekverades i bakgrunden, i decimal form.

$- Visar de shell-optioner som gäller vid det aktuella tillfället. Dessa kan vara satta då shell startades eller med kommandot *set*.

En del variabler har en speciell betydelse som endast används vid speciella tillfällen.

$HOME Standardargumentet för kommandot *cd*. Ett filnamn som anges utan /, antas ligga i det biblioteket som är det aktuella biblioteket. Det aktuella biblioteket ändras med hjälp av cd kommandot. Till exempel

```
$ cd /usr/bin/kurt
```

ändrar det aktuella biblioteket till /usr/bin/kurt. Kommandot *cd* kan anges utan argument, vilket är ekvivalent med att skriva

```
$ cd $HOME
```

Variabeln **$HOME** sätts automatiskt vid inloggning, men kan ändras vid behov.

$PATH Innehåller en lista på de bibliotek som innehåller kommandon. Varje gång ett kommando exekveras av shell utan att ett fullstän-

digt kommandofilnamn som börjar med / anges, söks denna lista igenom efter en exekverbar fil. Om **$PATH** inte är satt kommer som default följande bibliotek att genomsökas i följande ordning: aktuella biblioteket, /bin och /usr/bin. I **$PATH** skrivs biblioteken in åtskilda av : (kolon), exempel

```
PATH=:/usr/kurt/bin:/bin:/usr/bin
```

Strängen ovan specificerar att det aktuella biblioteket (nollsträngen innan första :), /usr/kurt/bin, /bin och /usr/bin ska genomsökas i denna ordning. Detta medför att användaren kan komma åt sina egna kommandon oberoende av det aktuella biblioteket. Om namnet på kommandot innehåller / kommer inte den just beskrivna sökningen av biblioteken att utföras, ett enda försök kommer att göras att exekvera kommandot.

$CDPATH Innehåller en lista på de bibliotek som ska avsökas då kommandot *cd* exekveras för att leta efter det bibliotek som anges om biblioteket anges utan inledande /. Formen är likadan, med kolon, som för variabeln **$PATH**. Om **$CDPATH** inte är definierad söks bara i det aktuella biblioteket.

$PS1 Promptsträngen i första shell, default $.

$PS2 Shell prompt när mer indata behövs, default '>'.

$IFS En sträng med alla tecken som shell anser som fältavgränsare mellan ord.

$MAIL Denna variabel anger den fil dit kommandot mail sänder meddelanden till användaren. När shell startas i interaktiv mod, t ex vid inlogging eller då sh-kommandot ges utan argument, och med jämna mellanrum därutöver enligt variabeln **$MAILCHECK** kontrolleras den specificerade filen innan promten skrivs ut. Om filen har ändrats sedan sist, skrivs följande ut **You have mail,** därefter är shell redo att ta emot ett kommando. Variabeln **$MAIL** sätts automatiskt vid inlogging enligt användarnamnet enligt exemplet nedan (namnet är kurt).

```
MAIL=/usr/spool/mail/kurt
```

$MAILCHECK	Värdet anger tiden i sekunder mellan kontrollerna av filen **$MAIL** för att informera användaren om att meddelande har kommit.
$MAILPATH	Detta är en kolon-separerad lista med meddelandefilnamn. Här kan till skillnad från i **$MAIL** flera filer specificeras och dessutom kan textsträngar anges vilka skrivs ut istället för texten **You have mail** då ett meddelande kommer. Olika texter kan anges för olika filer genom att filnamnet följs av tecknet % och motsvarande textsträng.

5.2.5 Testkommando

Kommandot *test* returnerar status noll om ett angivet villkor är uppfyllt och icke-noll om det inte är uppfyllt. Villkoret kan t ex vara om en fil existerar eller ej såsom i följande exempel:

```
if test -f fil
      then echo "Filen existerar"
      else echo "Filen finns inte"
fi
```

Generellt beräknar test ett värde som returneras som utgångsstatus för kommandot. Vanligen görs tester på shellvariabler. Några av de mest frekventa testargumenten återges här nedan.

test "s"	sant om argumentet s inte är nollsträngen I detta fall bör argumentet inneslutas av " " eftersom test ger ett felmeddelande om argumentet helt saknas. Citationstecknen krävs för att test ska kunna skilja på ett null-argument och om argumentet saknas.
test -f fil	sant om fil existerar som vanlig fil.
test -r fil	sant om fil existerar och är läsbar
test -w fil	sant om fil existerar och är skrivbar.
test -d fil	sant om fil existerar och är ett bibliotek.
test n1 -eq n2	sant om n1 och n2 är heltal och lika.

Ett annat exempel är en del ur en shell-procedur där kommandot testar om ett filnamn slutar med .bak'

```
if test $1 != `basename $1 .bak`
      then echo "Filnamnet slutar med .bak"
      else echo "Filnamnet slutar inte med .bak"
fi
```

En fullständig beskrivning av *test* finns i **Referenshandboken**.

5.2.6 Flödeskontroll - while (until) do done, shift

I shell finns det en while loop, en until loop och en if then else fi sats, hur dessa fungerar beror på vilket utgångsstatus de kommandon har som skall behandlas. Den generella formen för while loopen är:

```
while kommandolista1
      do kommandolista2
      done
```

while testar på värdet på utgångsstatusen för det sista enkla kommandot i kommandolista1, som exekveras varje gång som loopen genomlöps medan kommandolista2 bara exekveras om utgångsstatus lika med noll har returnerats från kommandolista1. Skulle utgångsstatusen vara skild från noll avslutas while-kommandot. Exempel:

```
while test $1
      do ........
            shift
      done
```

är likvärdigt med

```
for i
      do .......
      done
```

Shift är ett shellkommando som ändrar namnen på positionsparametrarna $2, $3, ... till $1, $2, ..., vilket medför att $1 försvinner.

Ytterligare en användning av while eller until loopen är att den väntar på att en extern händelse skall inträffa, först därefter exekveras den specificerade kommandolistan.

En until loop avslutas på ett annat sätt än while loop, t ex enligt följande exempel i filen testf.

```
$ cat testf
until test -f fil1
      do sleep 300
      done
kommandon
```

Loopen kommer att var 5:e min testa om filen fil1 har skapats och först därefter exekveras kommandona efter until-loopen. (fil1 kommer troligtvis att skapas av någon annan process).

Det är möjligt att bryta sig ur en while-, until- eller for-loop även utan att villkoret är uppfyllt genom kommandot *break*. På liknande sätt kan nästa varv i en while- eller for-loop påbörjas utan att resten av kommandona i kommandolistan genomlöps genom kommandot continue.

5.2.7 Flödeskontroll - if then elif else fi

I shell finns det också en generell villkorssats, if.

```
if kommandolista1
then kommandolista2
else kommandolista3
fi
```

if testar utgångsstatus från på det sista enkla kommandot i kommandolista1. Om detta är noll exekveras kommandolista2. Om det är skilt från noll exekveras istället kommandolista3.

if kan användas ihop med kommandot test för kontroll av om en fil existerar.

```
if test -f fil2
then behandla fil2
else gör något annat
fi
```

Existerar fil2 kommer den för filen specificerade processen att utföras, i annat fall kommer det som står efter else att utföras.

Ett exempel på hur if, case och for kan användas ges senare.

En multipeltest av if kan se ut på följande sätt:

```
if ...
      then
            ...
      else if ...
            then
            ...
            else if ...
                  ...
            fi
      fi
fi
```

För att undvika denna ganska klumpiga konstruktion kan en utvidgning av if användas.

```
if ...
then ...
elif ...
then ...
elif ...
...
fi
```

Som ett exempel på detta kan kommandot mytouch skapas. mytouch kan användas till att ändra sista modifieringstiden för i-noden för en eller flera filer. Denna tid ses med följande kommando:

```
$ l -c filnamn
```

Kommandot ligger i filen mytouch som kan ses med *cat*:

```
$ cat mytouch
flag=
trap "rm -f junk$$ ; exit" 1 2 3 15
for i
      do case $i in
            -c)   flag=N ;;
            *)    if test -f $i
                  then ln $i junk$$ ; rm junk$$
                  elif test $flag
                  then > $i
                  else echo fil ö'$iö' existerar ej
                  fi ;;
      esac
done
```

I detta exempel används flaggan **-c** till att tvinga filer att skapas om de inte redan existerar. Ett felmeddelande skrivs annars ut om filen inte existerar. Om argumentet **-c** hittas kommer shell att sätta shellvariabeln flag till en icke-noll sträng. Kommandona

```
ln ...
rm ..
```

länkar sig till filen och tar därefter bort den, detta medför att i-nodens modifieringstid uppdateras för filen. Däremot inte den modifieringstid

som gäller för filinnehållet, det som ses med kommandot *l* utan optionen **-c**.

Sekvensen

```
if kommando1
then kommando2
fi
```

kan skrivas enklare

```
kommando1 && kommando2
```

Jämför detta med

```
kommando1 || kommando2
```

Kommando2 utförs i detta fall (med | |) bara om bearbetningen av kommando1 har misslyckats, dvs om utgångsstatus från kommando1 är skilt från noll.

5.2.8 Gruppering av kommandon

En grupperingen av kommandona kan ske på två olika sätt, antingen med

```
kommando-lista;
```

eller

```
(kommando-lista)
```

I det första fallet utförs bara kommandolistan, medan den utförs som en separat process i det andra fallet. Exempel:

```
(cd xdir ; rm junk)
```

rm junk bearbetas i biblioteket **xdir** utan att det aktuella biblioteket ändras i det shell som anropats. Kommandolistan används nämligen av en ny shell som avslutas efter *rm*-kommandot. Grupperingen

```
cd xdir ; rm junk
```

utför samma sak som i exemplet ovan men det shell som anropats lämnas kvar i biblioteket **xdir**.

Det går även att ge namn åt en kommando-lista. Därvid definieras en namngiven funktion inom shell. Denna funktion kan sedan användas på samma sätt som vanliga kommandon, med den skillnaden att kommandolistan exekveras inom nuvarande shell istället för i en ny process. Argument kan ges, vilka blir temporära positionsparametrar $1, $2, ... Funktioner definieras som i följande exempel, där det är väsentligt att { } omger listan och att listan avslutas med ; eller ny-rad.

```
namn1() { kommandolista ; }
```

5.2.9 Felsökning i shellprocedurer

I shell finns det två olika verktyg för debuggning. Det första anropas i en procedur med

```
$ set -v
```

Shell kommer att skriva ut kommandoraderna exakt som de läses innan varje rad exekveras. Med detta verktyg kan isolerade syntaxfel letas

upp. En procedur kan anropas utan att procedurens funktion i övrigt påverkas:

```
set -vn kommando parametrar
```

Som kommando skall en shellprocedur anges.

Optionen **-v** kan användas tillsammans med optionen **-n** för att förhindra bearbetning av de kommandon som kommer efter *set*. Då listas enbart de lästa raderna ur shellproceduren.

Observera! Om *set* **-n** skrivs på en terminal förblir denna obrukbar tills ett end-of-file ges.

Det andra sättet att utföra en debuggning på är göra 'trace' under exekveringen:

```
$ set -x
```

Kommandona kommer att skrivas ut allt eftersom de används. Kommandot set kan användas utan optioner, varvid *set* listar alla definierade shellvariabler, shell- och set-optioner och positionsparametrar.

```
$ set
```

De optioner som är satta vid det aktuella tillfället kan fås fram med

```
$ echo $-
```

Dessa och flera andra optioner beskrivs under kommandot *set* i **Referenshandboken**.

5.3. Parametertolkning

En shellvariabel kan tilldelas ett värde vid en variabelsubstitution eller tilldelas ett värde innan en shellprocess anropas. Variabelns värde kan ändras med namn=värde, men då ändras variabelns värde enbart i den shell som exekverar. Enbart om variabeln exporteras (med kommandot export eller genom att kommandot *set* **-a** givits) kommer dess nya värde att vara definierat i anropade kommandon. Alternativt kan värdet ändras i ett anropat kommando genom att namn=värde anges i kommandoraden före kommandot. Då ändras variabelvärdet bara i den nya process som startas upp och inte i den shell, varifrån kommandot ges.

```
anv=kurt kommando
```

Variabeln anv kommer att ha värdet kurt i det kommando som utförs. Om kommandot *set* **-k** har givits kan argument av typ namn=värde införas var som helst i argumentlistan. Dessa shellvariabler kallas ibland för tangentbordsparametrar. De återstående argumenten, om det finns några, är tillgängliga som positionsparametrarna $1,$2,... .

Med kommandot *set* kan positionsparametrarnas värde ändras inne i en process, exempel:

```
$ set -- *
```

I detta exempel används en 'wildcard' (*) som utvecklas till alla filnamn i det aktuella biblioteket. Positionsparametern $1 kommer att tilldelas det första filnamnet i det aktuella biblioteket, $2 det andra osv.

Observera! Det första argumentet, --, försäkrar korrekt behandling även av filnamn som börjar på -, eftersom det hindrar set-kommandot från att tolka eventuella strängar som börjar på - som optioner. Jämför kommandot *set* i **Referenshandboken**.

5.3.1 Parameteröverföring

I ett anrop av en shell procedur kan både positions- och tangentbordsparametrar anges i kommandoraden. Shellparametrar i övrigt kan göras tillgängliga för en anropad process genom att det i förväg specificeras att dessa parametrar skall exporteras, exempelvis:

```
$ export anv ccs
```

Variablerna anv och ccs är märkta för export.

Vid anrop av en shell process kommer alla exporterade variabler att kopieras och kan därvid modifiera den shell som anropas. Allmänt gäller det att variabler som ändras inom en process aldrig kan återföras till den anropande shell-processen. En process kan aldrig modifiera den shell som startat den nya processen.

Med kommandot *set* **-a** anges att alla shellvariabler som skapas eller ändras efter detta kommando, automatiskt markeras för export.

Om variabler deklareras endast för läsning, read-only, kan den anropade processen hindras från att ändra dessa variabler. Detta kommando har samma syntax som *export*.

```
readonly namn ...
```

Observera! Detta kommando måste, liksom export, specificeras innan den nya shell processen anropas.

5.3.2 Parameterersättning

Om en shellparameter inte är satt, ersätts (substitueras) den med nullsträngen, t ex om variabeln ddd inte har tilldelats något värde kommer någon av följande kommandon inte att skriva ut något alls, utom en radframmatning.

```
echo $ddd
echo ${ddd}
```

En defaultsträng kan väljas istället för 'ingenting' om en variabel är odefinierad, om följande format används.

```
$ echo ${ddd-TOM}
```

Detta kommando skriver ut värdet på ddd om den har något, i annat fall skrivs strängen 'TOM' ut. I defaultsträngen är de vanliga konventionerna tillåtna,

```
$ echo ${ddd-'*'}
```

skriver ut * om variabeln ddd inte har något värde. På samma sätt kommer

```
$ echo $äddd-$1å
```

att skriva ut ddd om den har något värde, i annat fall skrivs värdet på positionsparametern $1 ut, om den är satt till något värde. En variabel kan tilldelas ett defaultvärde, om den inte har något värde:

```
$ echo ${ddd=.}
```

vilket ger samma resultat som:

```
if test "$ddd"
      then echo $ddd
      else ddd=. ; echo $ddd
fi
```

Med denna variabelsubstitution sätts ddd till . (punkt) om variabeln inte är satt sedan tidigare. Positionsparametrarna kan däremot inte ändras på detta sätt.

Med nedanstående exempel:

```
$ echo ${ddd?message}
```

skrivs värdet på variabeln ddd ut om den har något värde, i annat fall kommer shell att skriva ut texten 'message' och bearbetningen av proceduren avbryts. Om ingen text anges efter frågetecknet, kommer ett standard-meddelande att skrivas ut: **parameter null or not set.**

En shell procedur som måste ha en del startparametrar satta, kan inledas av test-kommandon, t ex enligt följande:

```
: ${user?saknas} ${acct?saknas} ${bin?saknas}
........
```

: är ett kommando i shell och det gör ingenting när väl dess argument har tilldelas värden. Om någon av variablerna user, acct eller bin inte är satta kommer shell att skriva ut textsträngen för den första som inte definierats och överge proceduren.

Ytterligare varianter på olika sätt att ersätta shellvariabler med värden beskrivs i *sh* i **Referenshandboken.**

5.3.3 Kommandoersättning

Standard output med data från ett kommando kan substitueras på samma sätt som en parameter. Hela strängen som står mellan tecknen '....' tar shell som ett kommando som skall exekveras, varefter strängen ersätts med utdata från kommandot. Exempelvis listar kommandot *pwd* på standard output det fullständiga namnet på det aktuella biblioteket. Om standard output för detta kommando är /usr/kurt/bin så är kommandot

```
$ ddd='pwd'
```

likvärdigt med

```
$ ddd=/usr/kurt/bin
```

Ett annat exempel är:

```
$ ls 'echo "$1"'
```

som är ekvivalent med

```
$ ls $1
```

Kommandosubstitution kan användas överallt där parametersubstitution kan användas, inklusive i here-dokument och behandlingen av resultaten är i båda fallen den samma. Detta medför att kommandon kan användas i shell procedurer för att ändra strängar. Ett exempel är kommandot *basename* som tar bort suffixet från en sträng.

```
$ basename main.c .c
```

Kommandot *basename* skriver ut strängen main genom att ändelsen '.c' tas bort. basename tar bort eventuellt inledande del biblioteksnamn fram till sista snedstrecket (/). Hur detta kan användas för substitution visas i följande lilla utdrag ur en shellprocedur. Variabeln A antas vara ett filnamn, från vilket suffixet .c skall tas bort. Variabeln B erhåller filnamnet utan suffix.

```
case $A in
     ...
     *.c) B='basename $A .c'
     ...
esac
```

Andra exempel visas nedan:

```
$ for i in 'ls -t' ; do ..... ; done
```

Variabeln i sätts till filnamnen i tidsordning, den senast ändrade används först, enligt **-t** optionen för kommandot *ls*.

```
$ set 'date' ; echo $3 $2 $6, Klockan $4
```

Skriver ut t ex: 20 Mar 1987, Klockan 13:59:59 genom att kommanot *set* sätter positionsparametrarna till de strängar som kommandot *date* ger.

5.3.4 Villkor för ersättning - '....' \ "...."

Shell är en makroprocess som utför parametersubstitution, kommandosubstitution och skapar filnamn med hjälp av wildcards, som argument till kommandona. I detta kapitel kommer det att diskuteras i vilken ordning dessa beräkningar utförs och vilken effekt olika beteckningsmekanismer har.

Initialt analyseras kommandona enligt de grammatiska regler som listas kortfattat i avsnitt 6 av detta kapitel. Innan ett kommando exekveras utförs dessutom följande substitutioner i kommandosträngen, varvid olika uttryck ersätts av motsvarande strängar.

- Parameter-substitution, t ex $user.
- Kommando-substitution, t ex 'pwd'.

I dessa fall kan endast en beräkning göras, om inte kommandot *eval* används. T ex om värdet på variabeln x är $y kommer följande att skriva ut $y på bildskärmen och inte värdet av variabeln y. Notera då att apostrofer måste användas vid definitionen av x för att $-tecknet inte ska tolkas som en substitution vid definitionen av x.

```
$ x='$y'
$ echo $x
```

kommer att skriva ut $y.

- Tolkning av blanktecken.

Resultaten från de ovan gjorda substitutionerna delas in i ord som inte innehåller några blanktecken (blank interpretation). De tecken som anses vara blanktecken är definierade i shellvariabeln **$IFS**. Åtminstone mellanslag och tab-tecken och ny-rad ingår där. Nollsträngen betraktas inte som ett ord om den inte är citerad, t ex

```
$ echo '' (två stycken spostrofer)
```

kommer att som första argument ge nollsträngen till kommandot *echo* medan

```
$ echo $null
```

inte kommer att tilldela *echo* något argument alls, om variabeln null inte är satt eller innehåller nollsträngen. Om man vill vara säker på att $null ska uppfattas som nollsträngen kan "...." användas, vilket inte hindrar att substitutionen utförs.

```
echo "$null"
```

- Skapande av filnamn med hjälp av wildcards.

Varje ord genomsöks efter tecknen *, ? och [...] och en alfabetisk lista skapas över filnamnen för att ersätta orden. Var och ett av dessa filnamn blir ett separat argument.

Den just beskrivna ersättningarna förekommer också på den lista av ord som tillhör en for-loop. I case-satserna kan endast de två första ty-

perna av substitutionerna utföras, och wildcards är lokala inom case-satsen.

För att hindra denna ersättning av ett uttryck med sitt motsvarande sträng-värde kan, förutom de tidigare beskrivna metoderna \ och '.....', även en tredje mekanism användas, "..... " (citationstecken). Till skillnad mot de två övriga kan fortfarande parameter-och kommandosubstitution göras, men den förhindrar att filnamn skapas med hjälp av wildcards och tolkning av blanktecken. Följande tecken har en speciell betydelse innanför "....." och måste föregås av \.

$	parameter-substitution
‘	kommando-substitution
"	avslutar en citeringssträng
\	citerar speciella tecken $ ’ " \

Exempel:

```
echo "$x"
```

ger värdet av variabeln x till echo. På samma sätt

```
echo "$*"
```

ger positionsparametrarna som argument till echo. Detta är ekvivalent med

```
echo "$1 $2 ..."
```

Beteckningen $@ har samma betydelse som $*, utom när den är inom "....".

```
echo "$@"
```

beräknar inte positionsparametrarna innan de skickas till echo, utan kommandot får $1 $2 etc som argument, dvs det är ekvivalent med nedanstående.

```
echo "$1" "$2"
```

I följande tabell visas hur olika metatecknen i shell avkodas inom de olika citationsmetoderna apostrof ('....') och citationstecken ("....") samt vid kommandosubstitution (‘....‘).

	Metatecken					
	\	$	*	‘	"	’
Apostrof ’:	nej	nej	nej	nej	nej	avsl.
Citationstecken ":	ja	ja	nej	ja	avsl	nej
Kommandosubstitution ‘:	ja	nej	nej	avsl.	nej	nej

avsl.	=	avslutar
ja	=	tolkas
nej	=	tolkas inte

Notera vid egna tester att kommandot *echo*, liksom flera andra kommandon, internt tolkar vissa specialtecken, t ex \ i positionsparametrarna. Denna tolkning sker då inte av shell. Tabellen ovan visar bara om shell tolkar tecknen.

Notera även att vid kommandosubstitution den nya shell som startas kan tolka metatecken som anges. Bl a $ och * tolkas alltså inte av den shell som substituerar utdata till kommandosträngen. Exempelvis så kommer * i nedanstående exempel att tolkas som filerna i /usr/kalle och inte i det nuvarande aktuella biblioteket.

```
echo '(cd /usr/kalle ; ls *)'
```

I de fall där mer än en tolkning skall göras på en sträng kan det i shell ingående kommandot eval användas. Till exempel, om variabeln X har värdet $y, som i sin tur har värdet pqrrrr, medför detta att följande sekvens skriver ut värdet av pqrrrr.

```
y=pqrrrr
X='$y'
eval echo $X
```

Generellt sett utvärderar kommandot *eval* sina argument precis som alla andra kommandon och resultaten behandlas som indata till shell. Indata läses och kommandot exekveras, exempel

```
wg='eval who | fgrep'
$wg kurt
```

är ekvivalent med

```
who | fgrep kurt
```

I detta exempel behövs *eval* därför att det annars inte finns någon tolkning av metatecken, t ex |, som följer på en substitution.

5.4. Fel som upptäcks av shell

Hur ett fel behandlas som upptäcks av shell beror på vilket fel det är och om shell används interaktivt eller inte. Om in- eller utdata till ett shell är kopplat till en terminal är detta shell ett interaktivt shell. Ett shell som anropas med tillvalet **-i** satt är också ett interaktivt shell även om det inte är kopplat till en terminal.

En exekvering kan misslyckas om något av följande fall inträffar:

- Omdirigeringen av in-och utdata har misslyckats, t ex en fil existerar inte eller kan inte skapas.
- Kommandot existerar inte eller kan inte exekveras.
- Kommandot avslutas på ett felaktigt sätt t ex bussfel eller minnesfel. (Se nedan för en lista på D-NIX systemsignaler.)
- Kommandot avslutas normalt men utgångsstatusen är skild från noll.

I alla de ovan beskrivna fallen kommer shell att gå vidare och exekvera nästa kommando. Ett felmeddelande kommer att skrivas ut av shell för alla utom det sista.

Tillvalet **-e** ser till att shell avslutas direkt om utgångsstatus från ett kommando är skilt från noll. **-e** har ingen effekt om shell är interaktiv.

Dessutom kan nedanstående fel inträffa,

- Syntaxfel t ex 'if...then...done', där 'done' borde vara fi.
- En signal t ex interrupt från en annan process eller från tangentbordet. Shell väntar på att kommandot skall avslutas och avslutas eller återvänder till terminalen. Kommandot *trap* används för att fånga signaler och göra något annat.
- Något fel på ett kommando som ingår i shell, t ex *cd*.

I dessa tre fall kommer shell att gå ur kommando proceduren. Ett interaktivt shell återvänder till terminalen för att läsa ett nytt kommando.

D-NIX signaler:

1. Hangup
2. Interrupt (DEL från tangentbordet)
3. *Uthopp (quit, CTRL-\ från tangentbordet)
4. *Ej tillåten instruktion
5. *Trace trap
6. *IOT instruktion
7. *EMT instruktion
8. *Floating point exception

9. Kill (kan inte fångas eller ignoreras)
10. *Bussfel
11. *Segmentöverträdelse
12. *Felaktigt argument till system anrop
13. Skriver till pipe utan att någon läser det.
14. Alarm klocka
15. Programavslutning (från kill)

De signaler som är märkta med asterisk (*) orsakar en minnesdump om de inte fångas av kommandot *trap*. Emellertid ignorerar shell signalen UTHOPP (quit) som är den enda externa signal (från tangentbordet) som kan ge upphov till minnes dump. De signaler som är intressanta för shell är 1, 2, 3, 14 och 15.

En process kan fånga alla signaler utom signal 9, för att hindra att processen avbryts. Detta görs med kommandot *trap*.

Signaler kan sändas till bakgrundsprogram med kommandot kill. Till förgrundsprocesser kan signalerna Interrupt(2) och Quit(3) sändas genom kontrolltangenter från tangentbordet. Koden för tangenterna (normalt DEL och CTRL-\) kan emellertid ändras, vilket också görs av många program.

5.4.1 Hanteringen av fel

En shellprocedur avslutas normalt om ett interrupt genereras från terminalen. Men om det skulle behövas kan kommandot *trap* användas för att ge möjlighet att 'städa upp', som t ex att ta bort temporära filer. Exempel:

```
trap "rm /tmp/ps$$ ; exit" 2
```

I exemplet sätts en *trap* för signal 2 (interrupt från terminalen). Om denna signal mottages, exekveras kommandot

```
rm /tmp/ps$$ ; exit
```

Notera att '....' inte kan användas eftersom $$ skall avkodas av shell till processnumret.

exit är ett annat kommando som ingår i shell och som kan avsluta exekveringen av en shellprocedur. Om inte *exit* kommandot anges i trapinstruktionen skulle *trap* kommandot, efter det att *trap* utförts, återvända till den position där shellprogrammet befann sig innan det blev avbrutet.

D-NIX signalerna kan behandlas på ett av följande tre sätt. De kan ignoreras, de skickas då aldrig till processen. De kan bli fångade, i detta fall måste processen bestämma vilken handling som skall utföras när signalen mottages. Till sist kan tas om hand av systemet, varvid de ger upphov till en avslutning av processen utan att någon ytterligare handling måste utföras. *trap* kommandot ignoreras om en signal ignoreras redan vid inträdet t ex genom att ett kommando exekveras i bakgrunden.

Hur trap används visas i nedanstående exempel som är en modifierad version av proceduren mytouch från tidigare av kapitlet. Filen junk$$ skall rensas bort om kommandot blir avbrutet.

```
flag=
trap "rm -f junk$$ ; exit" 1 2 3 15
for i
do case $i in
      -c)   flag=N ;;
      *)    if test -f $i
                  then ln $i junk$$ ; rm junk$$
            elif test $flag
                  then > $i
            else echo fil \'$i\' existerar ej
            fi ;;
      esac
done
```

Kommandot *trap* uppträder innan den temporära filen skapas, på grund av att det är möjligt att processen avbryts direkt efter skapandet av filen, innan fler kommandon hinner utföras.

Signalen 0 används till att indikera ett kommandon som skall exekveras vid utträdet ur nuvarande shell.

En procedur kan själv välja att ignorera signaler genom att nollsträngen specificeras som ett argument till *trap* kommandot.

```
trap '' 1 2 3 15
```

Detta ger till resultat att signalerna HANGUP, INTERRUPT, UTTRÄDE (QUIT) och KILL ignoreras av både proceduren och de kommandona som anropas från proceduren. Kommandot *nohup* fungerar på detta sätt.

trap återställs genom att följande skrivs. Signalerna 2 och 3 återfår i detta exempel sina standardvärden.

```
trap 2 3
```

En lista på de för tillfället aktiva *trap* värdena och motsvarande kommandon kan fås genom *trap* utan parametrar.

```
trap
```

Ett trap-kommando behöver inte avslutas med *exit*. Utan *exit* fortsätter exekveringen av proceduren där den avbröts, efter att trap-kommandot utförts. Exemplet nedan gör att signalen tas om hand, men ingen åtgärd utförs utan prceduren fortsätter som om ingenting hänt. Kommandot

```
: utför ingenting då signalen 2 kommer.
trap : 2
```

Proceduren nedan, som vi kallar scan, är ett exempel på en användning av *trap* där proceduren fortsätter efter trap-kommandot. Scan tar varje bibliotek i det aktuella biblioteket, visar dess namn och exekverar därefter varje kommando som skrivs på terminalen med det angivna biblioteket som aktuellt bibliotek, tills ett end-of-file (CTRL-D) eller interrupt (DEL) mottages från terminalen. Vid end-of-file byter proceduren till nästa bibliotek. Ett interrupt ignoreras under exekveringen av kommandot men avbryter proceduren om scan står och väntar på att operatören ska ange nästa kommando.

```
ddd='pwd'
for i in *
do if test -d $ddd/$i
      then cd $ddd/$i
           while echo "$i:"
                 trap exit 2
                 read kommando
           do trap : 2 ; eval $kommando ; done
      fi
done
```

Här används även shell-kommandot read, vilket ingår i shell och läser en rad i taget från standard input och placerar resultatet i en variabel, i detta fall i variabeln 'kommando'. Kommandot returnerar en utgångsstatus som är skild från noll om det läser ett filslut, varvid proceduren går ur while-loopen. Om ett avbrott (signal 2) genereras under read-kommandot, utförs trap-kommandot *exit*, medan avbrott under exekvering av det inlästa kommandot ignoreras.

5.5. Bearbetning av shellkommandon

För varje kommando som startas upp och som inte ingår i shell, startas en ny process upp med systemanropet *fork*. Innan kommandot utförs, skapar *fork* en 'barn'-process genom att kopiera 'fader'-processens öppna filer och signalstatus.

Även kommandon i filer med samma namn som interna shell-kommandon kan bearbetas genom shell-kommandot . (en punkt) följt av ett mellanslag och kommandofilens namn.

I en del fall behöver inte en ny process startas, då används istället det i shell ingående kommandot *exec*. Detta kommando byter bara ut shell mot ett annat kommando. Om det används direkt från tangentbordet för att bearbeta ett kommando kommer användaren att loggas ut efter att kommandot utförts, eftersom den ursprungliga shell ersattes av kommandot. En enkel version av kommandot *nohup* skulle kunna vara en shellprocedur med följande utseende.

```
trap '' 1 2 3 15
exec $*
```

trap förhindrar att de kommandon som skapas senare känner av de specificerade signalerna och *exec* ersätter den shell som utför *nohup* med det specificerade kommandot.

Det mesta om omdiregering av in-/utdata har beskrivits tidigare. Nedan är en sammanfattande tabell.

In- och utdata beräknas från vänster till höger precis som de förekommer i kommandot. Detta är av betydelse då en filbeskrivning (file-descriptor) dupliceras, dvs flera filbeskrivningar används för åtkomst till samma fil. Omdirigering med > är samma som 1> för standard output och omdirigering med < är samma som 0< för standard input.

> word	Standard output (filbeskrivning 1) skickas till filen word, som skapas om den inte redan existerar.
>> word	Standard output (filbeskrivning 1) skickas till filen word. Om filen existerar läggs utdata till sist i filen, annars skapas filen.
< word	Standard input (filbeskrivning 0) tas från filen word.
<<word	Standard input (filbeskrivning 0) läses från eller följande rader tills en rad med enbart strängen
<<-word	word påträffas eller tills filslut. Detta kallas 'here-dokument'. word kan vara en valfri sträng. Om något tecken i word är citerat, t ex med \, kommer shell inte att tolka metatecken i here-dokumentet, annars kommer parameter- och kommandosubstitution att utföras. \nyrad ignoreras alltid. Tecknet

	\ måste alltid används till att citera tecknen \, $, ‘ och även första bokstaven i word där strängen word i dokumentet inte ska vara avslutningstecken. Alla tab-tecken i strängen word och i dokumentet ignoreras om ett minustecken anges (<<-word).
<&*nr*	Filbeskrivning 0 (standard input) dupliceras med filbeskrivning *nr*, varvid standard input läses från den fil som associerats med filbeskrivmimg *nr*.
>&*nr*	Filbeskrivning 1 (standard output) dupliceras med filbeskrivning *nr*, varvid standard output sänds till den fil som associerats med filbeskrivning *nr*.
<&-	Standard input (filbeskrivning 0)stängs.
>&-	Standard output (filbeskrivning 1)stängs.

Omdirigeringarna som beskrivits ovan kan föregås av en siffra, vilket medför att den filbeskrivning som skapas kommer att specificeras av denna siffra istället för av standardvärdena 0 (för 0<) och 1 för 1>). Som exempel kan "standard error output", som har filbeskrivning 2, sändas till en fil genom:

```
... 2> filnamn
```

Till standard error output sänder många kommandon sina felmeddelanden. Ett kommando som har utdata riktad mot en fil kan även sända felmeddelanden till samma fil genom:

```
... >filnamn 2>&1
```

Filbeskrivning 2 duplicerar här filbeskrivning 1, varvid utdata och felmeddelanden blandas, på samma sätt som normalt sker vid utskrift till terminalen. Notera att standard output (>) måste specificeras innan dupliceringen (2>&1) sker i kommandot.

Nya filer med andra filbeskrivningar än 0, 1 och 2 kan öppnas och/eller skapas genom att kommandot *exec* används utan att någon process anges, dvs med enbart omdirigeringar. Dessa kommer då att gälla nuvarande shell. Till exempel kan en fil med filbeskrivning 5 öppnas, från vilken kommandot *read* kan läsa strängar till shellvariabler. På detta sätt kan *read* användas för att omväxlande läsa från olika filer.

```
exec 5< annanfil
read 0<&5 onestr twostr threestr
echo 'Från filen :' $onestr : $twostr : $threestr :
read fourstr
echo 'Från tangentbordet :' $fourstr
```

I detta exempel läses en rad från filen annanfil och shellvariablerna onestr och twostr tilldelas värdet av de två första textsträngarna i raden medan threestr innehåller resten av raden. Därefter används read igen men läser denna gång från terminalen.

Miljön för ett kommando som körs i bakgrunden (startas med &) påverkas genom att standardvärdet för standard input till ett sådant kom-

mando är den tomma filen **/dev/null**. Detta förhindrar att shell och kommandot, som körs parallellt, läser samma indata. Om så inte vore fallet skulle snabbt kaos uppstå, t ex skulle kommandot

```
ed fil &
```

tillåta att både editorn och shell läste samma indata vid samma tidpunkt. För utdata ändras inget automatiskt utan användaren måste omdirigera standard output om bakgrundsprocessen sänder data den vägen och dessa inte får hamna på terminalen.

Den andra modifieringen som utförs på ett kommando som körs i bakgrunden, är att signalerna UTHOPP (QUIT) och INTERRUPT 'stängs av' så att de ignoreras av kommandot. Detta medför att dessa signaler kan användas från terminalen till andra kommandon, utan att det påverkar kommandon som körs i bakgrunden.

Observera! *trap* kommandot har ingen effekt på en ignorerad signal.

5.5.1 Anrop av shell

Följande optioner tolkas av shell när det anropas. Dessutom finns flera shell-optioner som kan sättas och tas bort med kommandot *set*. Nedanstående optioner kan inte ändras med *set*.

Om det första tecknet i argument noll (kommandonamnet) är ett minustecken, kommer kommandon först att läsas från filerna **/etc/profile** och **$HOME/.profile**. Därefter läses de enligt nedan. Detta gäller t ex den shell som startas vid inloggning till systemet.

Om inte **-c** eller **-s** optionen väljs kommer första argumentet på kommandoraden att tolkas som en fil med en shell-procedur. Denna procedur utförs därvid med övriga argument som positionsparametrar.

-V	Skriver ut versionsnumret för kommandot *sh*.
-c ***string***	Om optionen **-c** är satt kommer kommandona att läsas från strängen *string* på kommandoraden istället för från en fil.
-s	Om optionen **-s** är satt eller om inga argument återstår, kommer kommandona att läsas från standard input. Med **-s** används övriga argument som positionsparametrar i shell. Utdata från shell kommer normalt att skrivas till filbeskrivare 2, d v s till standard error output.
-i	Om **-i** optionen är satt eller om in-/utdata från shell är kopplad till en terminal, är shell interaktiv. I detta fall ignoreras signalen TERMINATE (för att inte kill 0 skall avsluta ett interaktivt shell) och signalen INTERRUPT fångas men ignoreras normalt (interrupt kan utföras i kommandot *wait*). Signalen UTHOPP (QUIT) ignoreras alltid.

-r	Med optionen **-r** blir startas shell som en begränsad shell, där användaren hindras att utföra vissa operationer.

5.5.2 Begränsad shell - rsh

Shell kan startas upp som en begränsad shell med en av följande metoder:

- Shell startas med optionen **-r**.
- Shell startas med kommandot *rsh*, länkat till *sh*.
- Variabeln **SHELL** är definierad och innehåller bokstaven r.

En begränsad (restricted) shell tillåter allt som en vanlig shell utom följande operationer. Dessa kan inte utföras:

- Byte av aktuellt bibliotek (*cd*).
- Ändring av variabeln **PATH**.
- Kommandon med namn som innehåller /.
- Omdirigering av utdata (> eller >>).

Detta begränsar användaren till kommandon enligt den fördefinierade **PATH** och i det bibliotek som är aktuellt då den begränsade shell startas. Användaren kan inte heller använda omdirigering för att skriva till några filer. Om den begränsade shell startas vid inloggning, exekveras däremot först **/etc/profile** och **$HOME/.profile** med fulla rättigheter och kan till exempel sätta upp **PATH** och flytta till rätt bibliotek innan användaren får ge egna kommandon.

Kommandon i kommandofiler som startas exekveras alltid med fulla rättigheter om inte shell-optionen **-r** anges.

5.6. Syntax och konventioner

5.6.1 Kommandosyntax

post:	indata och utdata namn=värde
enkelt kommando:	post enkla kommandon
kommandon:	enkla kommandon (kommandolista) kommandolista for namn do kommandolista done for namn in ord ... do kommandolista done while kommandolista do kommandolista done until kommandolista do kommandolista done case ord in case-del esac if kommandolista then kommandolista else-del fi
pipeline:	kommando pipeline | kommando
andor:	pipeline andor && pipeline andor || pipeline
kommandolista:	andor kommandolista ; kommandolista & kommandolista ; andor kommandolista & andor
indata-utdata:	> fil < fil >> fil <<ord
fil:	ord &siffra &-
case-del:	mönster) kommandolista ;;
mönster:	ord mönster | ord
else-del:	elif kommandolista then kommandolista else kommandolista tomt

tomt:	
ord:	en sekvens av icke blanka tecken
namn:	en sekvens av bokstäver, siffror och understrykningar som börjar med en bokstav
siffra:	0 1 2 3 4 5 6 7 8 9

5.6.2 Metatecken och reserverade ord

För de symboler som är svenska specialtecken hänvisas till kapitel 1 för en komplett lista över vilka svenska tecken som skall användas istället för motsvarande internationella tecken.

a) syntax

|	pipe symbol
&&	'and' symbol
||	'or' symbol
;	kommandoseparator
;;	case begränsare
&	bakgrundskommando
()	kommando gruppering
<	omdirigering av indata
<<	indata från ett here dokument
>	skapande av utdata
>>	utdata läggs till
=	tilldelning av värde

b) mönster

*	matchar alla teckensträngar
?	matchar ett ensamt tecken
[...]	matchar alla tecken som står innanför
[!...]	matchar alla tecken utom de som anges

c) substitution

${...}	substituerar en shell variabel
`...`	substituerar utdata från kommando

d) citering

\	citerar nästa tecken dvs tecknet behandlas inte som metatecken.
'...'	citerar de inneslutna tecknen utom '
"..."	citerar de inneslutna tecknen utom $ ` \ "

e) reserverade ord

if then else elif fi
case in esac
for while until do done
{}
:

6

6. Elektronisk post

6. Elektronisk post

Att skicka meddelanden till varandra via dator, elektronisk post, är ett ganska nytt begrepp som kan ersätta gamla vanliga metoder som att skicka brev, telex, ringa telefonsamtal, olika typer av komihåg system och en hel del annat. Den typ av meddelandeförmedling som finns i D-NIX brukar kallas brevlådesystem. Varje användare som har en egen användaridentitet på systemet har en egen brevlådefil där meddelanden från andra användare samlas.

Här beskrivs nu en del av de funktioner som finns med i detta meddelandesystem:

- Du skriver in dina meddelanden på ditt tangentbord och kan därför lättare ändra på saker om du skulle skriva fel.
- Du kan skicka identiska kopior på ditt meddelande till ett stort antal personer samtidigt.
- Det går mycket fortare att skicka ett brev med elektronisk post än det går att skicka det med den vanliga posten, vare sig det gäller internpost eller med postverket. Ett meddelande via elektronisk post når mottagaren nästan samtidigt som du skickar det.
- Ett meddelande via elektronisk post stör inte mottagaren i samma utsträckning som ett telefonsamtal. Du kan läsa ett mottaget meddelande när du har tid och lust.
- Brevet kan arkiveras på ett smidigt sätt för att sedan, lika smidigt, kunna plockas fram när det behövs igen.
- Du kan sända meddelanden till någon i samma hus eller till någon på andra sidan jordklotet, genom att använda något världsomspännande nätverk.
- Du behöver inga frimärken, kuvert eller papper och du behöver inte springa iväg till några postkontor.

Den största nackdelen med denna typ av meddelande förmedling är att mottagaren måste vara inloggad på sitt datorsystem för att få reda på att det finns ett meddelande att läsa.

Kommandot som används i UNIX-världen för att skicka meddelanden och även för att ta emot och läsa dem heter *mail*.

Det finns flera sätt att skapa, läsa och göra sig av med meddelanden.

6.1. Att sända meddelanden

Grundkommandot för att sända ett meddelande med mail är:

```
mail lognamn
```

Där lognamn är mottagarens användar-ID på UNIX-systemet. Detta loginnamn kan antingen vara användar-ID om mottagaren finns på ditt system (t ex **mats**) eller ett systemnamn och en användar-ID om mottagaren finns på ett annat UNIX-system som kan kommunicera med ditt system (t ex **sys!mats**).

Om vi antar att mottagaren finns på ditt system skriver du *mail* följt av en användar-ID och **RETURN**. Sedan kan du börja skriva in ditt meddelande på följande rad. Det finns ingen begränsning för hur långt ditt meddelande får vara. När du skrivit färdigt meddelandet, sänder du iväg det genom att antingen skriva en punkt (.) följd av **RETURN** eller genom att trycka på **CTRL-D**. Det måste i båda fallen göras först på en ny rad.

Nedan följer ett exempel på hur det hela kan ha sett ut på skärmen.

```
$ mail stina
Mitt möte med Per Persson i morgon<Retur>
har blivit flyttat till 12:00 så vi<Retur>
kan tyvärr inte äta lunch som vi kom<Retur>
överens om. Kanske vi kan ta lunchen<Retur>
på fredag istället?<Retur>
.
$
```

När sedan Stina loggar in på sin terminal, eller om hon redan är inloggad får hon följande meddelande om att hon har post:

```
You have mail
```

Hur Stina skall göra för att läsa inkommna meddelanden beskrivs under rubriken Att ta emot meddelanden.

Ett bra sätt att pröva och öva *mail* är att skicka ett meddelande till sig själv. Att skicka mail till sig själv kan också fungera som ett bra komihågsystem, du kan till exempel skicka följande meddelande på kvällen innan du går hem (vi antar att ditt eget loginnamn är mig):

```
$ mail mig
Ta reda på varför Olle inte har hört av<Retur>
sig angående samtalet med Hans Persson.<Retur>
.
$
```

När du loggar in nästa dag får du upp meddelandet om att du har post. Du kan sedan läsa meddelandet för att påminnas om att kolla upp varför Olle inte har hört av sig.

6.1.1 Att sända ett meddelande till flera personer samtidigt

Du kan skicka ett meddelande till ett antal olika personer samtidigt genom att skriva deras loginnamn på kommandoraden efter *mail* kommandot. Se exemplet nedan.

```
$ mail benny tomas per ola sten
Glöm inte bort vår interna badminton
turnering på torsdag klockan 7.
Glöm inte heller att ta med 50 kronor var
att efter tävlingen utdela till vinnaren.
MVH Nisse
.
$
```

benny, tomas, per, ola och sten kommer nu snart att få upp meddelandet om att de har post på skärmen, om de är inloggade. Annars får de upp meddelandet så fort de loggar in.

6.1.2 Att skicka post till andra system

Hitills har du bara skickat meddelanden till användare på samma system som dig själv, men det går att nå andra användare också. Du kan skicka meddelanden till alla system som på något sätt är ihopkopplat med ditt. Det kan vara system på andra avdelningar i samma hus eller andra företag eller lokalkontor i samma land. Du kan också skicka meddelanden till användare i andra länder bara det finns någon typ av kommunikation mellan era system.

För att skicka meddelanden till andra system måste du ange systemnamnet efter mail-kommandot. Om det andra systemet heter **sys3** och användaren som du vill sända meddelandet till har loginnamn **mats** gör du så här:

```
$ mail sys3!mats
```

Efter detta kommando skriver du ditt meddelande som vanligt. Observera utropstecknet (!) mellan systemnamnet och loginnamnet.

Innan man kan använda mail på detta sätt måste man ha reda på systemnamnet på den maskin som användaren man vill skicka meddelandet till kör på. Lättaste sättet är då att fråga användaren om detta. Om han nu inte vet vad systemet heter kan du be honom använda följande kommando på sitt system.

```
uname -n
```

kommandot kommer att svara med namnet på systemet, som i följande exempel:

```
$ uname -n
sys3
$
```

Nu vet du vad den andre användarens system heter, men du vet fortfarande inte om ditt system och hans kan kommunicera med varandra. Det lättaste sättet att kontrollera detta är att skicka ett testmeddelande till den andre användaren. Om meddelandet kommer fram så kan de två systemen kommunicera med varandra. I annat fall är det troligt att de inte kan det.

Det kan finnas möjligheter att gå runt ett sådant problem. Låt säga att ditt system heter **a** och användaren **per** som du vill sända ett meddelande till finns på system **c**. System **a** och **c** har ingen direktkontakt med varandra, däremot har båda kontakt med ett annat system som heter **b**. Vad man då kan göra är följande:

```
$ mail b!c!per
```

Vad man gör är alltså att skicka meddelandet till system **b** som sedan skickar det vidare till system **c** och användare **per**.

Det kanske också finns ett system **d** som du har kontakt med och som har kontakt med system **c**. Det kan du då också använda, det kan vara bra om system **b** inte fungerar någon gång då du vill skicka ett meddelande.

Då det inte finns någon kontroll på om ett meddelande kommer fram kan det vara bra om mottagaren bekräftar att han mottagit meddelandet genom att t ex skicka ett kort svar. Detta är speciellt viktigt när många system ingår i adressekvensen och man inte vet om något av dessa inte fungerar just för tillfället.

6.2. Att ta emot meddelanden

Med mail-kommandot kan du inte bara sända meddelanden du kan också läsa meddelanden du fått från andra användare eller kanske från dig själv som en påminnelse om någonting.

Om du är inloggad och någon har skickat ett meddelande till dig kommer du efter en stund få upp följande meddelande på skärmen:

```
You have mail
```

Det betyder att ett eller flera meddelanden ligger och väntar på att bli lästa. Meddelanden finns i biblioteket **/usr/mail/loginamn**, där loginnamn är ditt eget användar-ID. Detta bibliotek brukar man kalla postlåda. För att hämta meddelanden från din postlåda och få upp dem på skärmen skriver du *mail* kommandot utan några argument.

```
mail
```

Meddelandena visas nu ett i taget med början på det senast mottagna meddelandet. Det kan se ut som följer:

```
$ mail
From stina Tue Mar 21 09:33 MET 1989
Jag kan tyvärr inte tacka ja till lunchen
på fredag då jag har ett viktigt möte.
Tack ändå,
Stina

?
```

Första raden, det så kallade huvudet, innehåller information om meddelandet: Sändarens loginnamnm, vilket datum och vid vilken tid meddelandet skickades. Följande rader ända ned till raden med det ensamma frågetecknet (?) är själva meddelandet.

Om du får upp långt meddelande på skärmen är det inte säkert att du hinner läsa allt på en gång. Då kan du stoppa utskriften på skärmen med hjälp av **CTRL-S**. Detta fryser skärmen så att du kan läsa i lugn och ro, när du läst färdigt trycker du bara på **CTRL-Q** så fotsätter utskriften på skärmen.

Efter varje meddelande som skrivs ut, skrivs ett frågetecken (?) ut. Mail väntar på ett svar från dig. Det finns många olika svar, optioner, att välja på, du kan t ex lämna meddelandet i postlådan medan du läser nästa. Du kan också ta bort det helt eller kanske spara undan det någonstans för att senare kunna använda det som referens t ex. För att få reda på vilka optioner som finns behöver du bara skriva ett frågetecken vid frågetecknet och trycka på **RETUR**.

För att få se nästa meddelande utan att ta bort föregående meddelande behöver du bara trycka **RETUR** efter frågetecknet.

```
?
```

Nuvarande meddelande blir kvar i din postlåda och nästa meddelande visas på skärmen. Om du har läst alla meddelanden som fanns i din brevlåda så kommer prompten ($) upp.

För att ta bort ett meddelande från postlådan skriver du ett **d** efter frågetecknet:

```
?d
```

Meddelandet är borttaget och om det finns några meddelanden kvar i postlådan visas, i så fall, nästa meddelande. Om så inte är fallet kommer prompten ($) upp som en bekräftelse på att du läst samtliga meddelanden.

För att spara meddelandet i en fil kallad **sparmail**, för att senare kunna ta fram det igen, skall du göra som följer, filen kommer att hamna i ditt hemmabibliotek:

```
? s sparmail
```

Om det redan finns en fil som heter **sparmail** skrivs den över i annat fall skapas en ny fil med angivet namn. Du kan sedan använda t ex *ls* kommandot för att verifiera att det finns en ny fil med namnet **sparmail**. (*ls* skriver ut innehållet i nuvarande bibliotek på skärmen.)

Du kan även spara meddelandet under ett annat filnamn i ett annat bibliotek om du så önskar genom att specificera sökvägen, till exempel:

```
? s projekt/meddel/meddel1
```

Filen här kallad **meddel1** kommer att sparas i bibliloteket **meddel** som är ett underbibliotek till **projekt** som i sin tur är ett underbibliotek till ditt hemma bibliotek. (För mer information om hur man använder sökvägar refereras till kapitlet om Shell.

För att avsluta läsandet av meddelanden skriver du ett **q** efter frågetecknet:

```
? q
```

De meddelanden som du inte läst förblir kvar i din postlåda tills du använder mail-kommandot nästa gång.

6.3. Om du skulle göra fel

Om du gör ett fel när du skriver in mottagarens användar-ID, kan *mail* kommandot inte skicka ditt meddelande. Istället kommer du få upp ett meddelande på skärmen som talar om att det inte har gått bra och att ditt meddelande sparats under filnamnet **dead.letter**.

Som ett exempel du vill skicka ett meddelande till en användare med loginnamn peter på ett system som kallas sys30. Ditt meddelande är "Mötet hålls i morgon klockan 2 istället för idag". Du upptäcker inte att du skrivit loginnamnet petre istället för det korrekta peter och sänder iväg meddelandet.

```
$ mail petre
Mötet hålls i morgon klockan 2 istället för idag.
.

mail:Can't send to petre
Mail saved in dead.letter
$
```

Meddelandet du tänkte skicka till peter finns nu i filen **dead.letter**. Om du vill kan du gå in och ändra i den med hjälp av en editor, t ex *dmacs*. När du sedan vill skicka iväg meddelandet gör du som följer:

```
$ mail peter < dead.letter
```

peter kommer nu få det meddelande som finns i dead.letter.

6.4. Interna kommandon i mail

mail utan argument skriver ut meddelanden till användaren, ett i sänder i ordning sist inkommet, först utskrivet. För varje meddelande ställs en fråga med tecknet ? och en rad läses från standard input för ett kommando som anger vad som skall göras med meddelandet. Följande interna kommandon finns i *mail*:

<new-line>	Fortsätt med nästa meddelande
+ or n	Samma som <new-line>
d	Ta bort meddelandet och fortsätt med nästa meddelande
d #	Ta bort meddelande nr #. Gå inte vidare till nästa meddelande
dq	Ta bort meddelande och avsluta *mail*
h	Visa ett fönster med brevhuvud runt nuvarande meddelande
h #	Visa brevhuvud för meddelande nr #
h a	Visa brevhuvuden för alla meddelanden i användarens *mailfil*
h d	Visa brevhuvuden för de meddelanden som valts att tas bort
p	Visameddelandet igen.
-	Gå tillbaka till föregående meddelande.
a	Visa meddelande som ankom under det att du körde *mail*
#	Visameddelande nr #
r [användare]	Svara avsändaren och annan/andra användare, ta sedan bort meddelandet
s [filer]	Spara meddelandet i angivna filer (mbox är standard).
y	Samma som tillvalet s
u [#]	Ta tillbaka borttaget meddelande nummer # (senast lästa är standard)
w [filer]	Spara meddelandet, utan brevhuvud, i angivna filer (mbox är standard).
m [mottagare]	Sänd meddelandet till angivna mottagare (du själv är standard).
q	Lägg tillbaka ej borttagna meddelanden i mail-filen och avsluta.
EOF (CTRL -D)	Samma som q.

x	Lägg tillbaka alla meddelanden i mail-filen oförändrade och avsluta.
!command	Gå in i en ny temporär shell för att utföra ett kommando.
*** eller ?**	Visa en kommandosammanfattning.

6.5. Några exempel

Att skicka meddelanden:

Det enklaste, att skicka ett meddelande till dig själv:

```
$ mail migsjälv
Mitt loginnamn på maskinen är migsjälv,
punkt (.) eller <ctrl-d> ensam på enrad skickar meddelandet.
.
$
```

Att skicka till någon annan på samma system som dig:

```
$ mail annan
Jag skickar härmed ett meddelande
till dig som har loginnamn annan.
Kan du svara mig så fort du får
tid?
.
$
```

Skicka ett meddelande till ett antal olika användare på samma system:

```
$ mail anv1 anv2 anv3
Skickar samtidigt mail till 3 st
användare som har loginnamnen
anv1, anv2, anv3. De kommer alla att
få samma meddelande, dvs detta.
MVH mig
.
$
```

Skicka ett meddelande till någon på ett annat system:

```
$ mail ejeget!någon
Ett meddelande till en användare på
systmet ejeget med loginnamn någon
.
$
```

Vad någon skall göra om du vill skicka ett meddelande till honom och varken han eller du vet vad hans system heter:

```
$ uname -n
ejeget
$
```

Om du inte har direktkontakt med ett system men har kontakt med ett annat system som i sin tur har kontakt med systemet du vill skicka ett meddelande till kan du göra så här:

```
$ mail b!c!per
Du skickar meddelandet till system b
som du har kontakt med som i sin tur
skickar meddelandet till per på system
c som du inte har direktkontakt med.
```

Om du skulle skriva fel loginnamn:

```
$ mail fle

Den jag skickar till har loginamn fel
och inte fle, man får då följande
felmeddelande
.

mail:Can't send to fle
Mail saved in dead.letter

$
```

För att skicka iväg meddelandet i dead.letter till loginnamn fel:

```
mail fel < dead.letter
```

Att ta emot meddelanden:

När du får upp detta meddelande på skärmen betyder det att du har ett eller flera meddelanden som väntar på att bli lästa:

```
$ you have mail
```

För att läsa dem skall du bara skriva *mail* följt av **RETUR**:

```
$ mail

Du får då upp det senast skickade
meddelandet på skärmen. Efter meddelandet
finns ett frågetecken (?) efter det kan
du skriva ett kommando som talar om vad
mail skall göra härnäst.

?
```

Om du enbart skulle trycka **RETUR**:

```
?

Du får upp näst senaste skickade
meddelande, dvs nästa meddelande i
brevlådan på skärmen om du enbart
trycker return. Föregående meddelande
finns fortfarande kvar i brevlådan.

?
```

Om du skriver **d** följt av **RETUR**:

```
? d

d följt av return tar bort meddelandet
före frågetecknet (?) ur brevlådan och
du kan inte få fram det igen.
```

Om du skriver **s** följt av ett filnamn och **RETUR**:

```
? s sparmail


Detta gör att meddelandet ovanför
frågetecknet sparas i nuvarande bibliotek
under namnet sparmail
```

```
? s projekt/meddel/meddel1

Detta gör att meddelandet ovanför
frågetecknet får namnet meddel1 och
sparas i biblioteket meddel som är
ett underbibliotek till projekt som i
sin tur är ett underbibliotek till
det nuvarande biblioteket.
```

Om du skriver **q** följt av **RETUR** avlutas *mail* och inga fler meddelanden visas. Du kan sedan återigen skriva *mail* om du vill se de meddelanden som finns kvar i brevlådan.

```
? q
$
```

7

7. Unix Kermit

7. Unix Kermit

C-Kermit är en helt ny implementation av Kermit. Den är skriven i moduler i programspråket C för att lätt kunna flyttas mellan olika datorer. Protokollets tillståndstabell är beskriven i wart, en lex-liknande förprocessor till C. De systemberoende funktionerna har isolerats i separatkompilerade moduler så att programmet lätt ska kunna flyttas mellan olika Unix datorer, men även för andra datorer med C-kompilatorer.

Dokumentationen skrevs ursprungligen av Frank da Cruz, Columbia University, och Herm Fischer, Litton Data Systems. Programmet utvecklades vid Columbia University av Frank da Cruz, Bill Catchings, Jeff Damens och Herm Fischer, med hjälp från många andra.

Funktioner i Unix Kermit:

Lokal användning:		Ja
Fjärranvändning:	Ja	
Login script:		Ja (som UUCP)
Överföring av textfiler:		Ja
Överföring av binära filer:	Ja	
Wildcard vid sändning:		Ja
Möjlighet att avbryta filöverföring:	Ja	
Omdöpning av filer med samma namn:		Ja
Möjlighet till time out:		Ja
Prefix för åttonde biten:		Ja
Prefix för upprepade tecken:		Ja
Alternerande blockkontroll:	Ja	
Terminalemulering:		Ja
Kommunikationsparametrar kan ställas:	Ja	
Överför BREAK:		Ja
Understödjer uppringande modem:	Ja	
Kommunikation med IBM stordatorer:		Ja
Transaktions-loggning:		Ja
Sessions-loggning:		Ja
Debug-loggning:	Ja	
Paket-loggning:	Ja	
Fungera som server:		Ja
Avancerade server-funktioner:		Ja
Lokal filhantering:		Ja
Kommando- och initieringsfiler:	Ja	
Fillåsning för UUCP och fleranvändare	Ja	
Långa paket:		Ja
Glidande fönster:		Nej
Filattribut:		Nej
Makron för kommandon:	Nej	
Rå filöverföring:		Nej

Alla siffror i detta avsnitt om Kermit är angivna med decimal notation om inte annat anges.

Kermit kan användas både som ett direktkommando, enligt traditionell Unix-stil med interaktiva kommandon. När den används som direktkommando får användaren tillgång till en delmängd av Kermits funktioner. Används däremot interaktiva kommandon, får användaren tillgång till alla funktioner.

På system som har uppringande modem kan Kermit använda kommandofiler och login script. Detta ger möjlighet att efterlikna de funktioner för filöverföring som finns i uucp mellan olika typer av datorer, även med möjlighet att sända och ta emot filer utan operatör, t ex på natten.

7.1. Unix filsystem

För en noggrann beskrivning av ditt filsystem bör du titta i handboken som beskriver din version av Unix. För att använda Kermit är det värt att känna till en del detaljer. Unixfiler har normalt namn som består av gemena bokstäver. Biblioteken är ordnade i en trädstruktur. Nivåer mellan biblioteken åtskiljs av tecknet "/". T ex

```
/usr/foo/bar
```

beskriver sökvägen till filen bar i biblioteket /usr/foo. Wildcard eller metatecken gör det möjligt att beskriva flera filer i taget. En "*" matchar alla strängar och ett "?" matchar ett tecken, vilket som helst.

När Kermit startas med filer specificerade på en kommandorad i Unix kommer shell att expandera metatecknen. Här blir fler möjligheter tillgängliga då shell hanterar fler kombinationer än vad Kermit gör. T ex

```
kermit -s ~/ck[uvm]*.{upd,bwr}]
```

expanderas av shell till en lista av alla filer i användarens hembibliotek (~/) som startar med bokstäverna "ck" följt av en bokstav, u, v eller m, följt av en eller flera tecken, följt av en punkt samt en av strängarna "upd" eller "bwr". Kermit själv expanderar endast metatecknen "*" och "?".

Unix filer är uppbyggda som en textström av 8-bitars tecken. En textfil består av 7-bitars ASCII-tecken med den högsta biten nollställd, och med raderna separerade med Unix newline tecken, d v s linefeed (LF, ASCII 10 decimalt). Detta skiljer sig från textfiler på de flesta andra datorer, där raderna åtskiljs av carriage-return/linefeed (CRLF, ASCII 13 följt av ASCII 10). Binära filer har ofta data där den högsta biten är satt. Därför behandlas dessa på ett speciellt sätt.

Vid filöverföring konverterar Kermit automatiskt mellan filnamn med versaler och gemena samt mellan CRLF och LF som radslutsmarkering, om den inte får ett kommando som säger något annat. När binära filer överförs måste programmet ges ett kommando för att inte utföra LF/CRLF konverteringen (**-i** på en kommandorad eller **set file type binary** interaktivt, se nedan).

7.2. Filöverföring

Om Kermit är i lokal mod kommer bildskärmen (stdout) att kontinuerligt uppdateras med information om hur filöverföringen går. En punkt skrivs ut för vart fjärde datapaket, medan andra typer av paket visas med följande tecken:

I	Utbyte av parameterinformation
R	Initiera mottagning
S	Initiera sändning
F	Filhuvud
G	Generellt "server" kommando
C	"Remote" värd kommando
N	Negativt godkännande (NAK)
E	Kritiskt fel
T	En timeout har skett
Q	Markerar att ett felaktigt eller ett ej godkänt paket har tagits emot
%	Markerar att ett paket har sänts om

Du kan skriva vissa "avbrotts" kommandon under filöverföringen:

CTRL-F:	Avbryt aktuell filöverföring och ta nästa (om den finns).
CTRL-B:	Avbryt hela filöverföringen och avsluta hela transaktionen.
CTRL-R:	Sänd om aktuellt paket
CTRL-A:	Visa en statusrapport över den aktuella transaktionen.

Dessa avbrottstecken skiljer sig från de som används i andra Kermit implementationer. Detta har gjorts för att undvika konflikter med de tecken som används för avbrott i Unix shell. I System V implementationen av Unix måste avbrottskommandon föregås av 'connect' escapesekvensen (normalt \). **CTRL-F** och **CTRL-B** fungerar bara vid överföringen av datapaket (D), och kan därför inte användas för att avbryta filöverföringar som inte hunnit så långt.

Varning: Om **CTRL-F** eller **CTRL-B** används för att avbryta en inkommande fil som har samma namn som en redan existerande fil och "file warning" inte är aktiverad, kommer den befintliga filen att försvinna.

Nödavbrott: När Unix Kermit körs i fjärrmod och du har startat en protokolloperation (sända eller ta emot en fil, server kommandot *wait* mm) kan du inte få kontroll över terminalen innan protokolloperationen har slutförts (färdigutförd eller avbruten p g a time out). Du kan t ex inte stoppa överföringen med Unix vanliga avbrottssekvens, eftersom terminalen är i "rå mod". Om du ändå behöver avbryta av någon anledning, kan du skriva

två **CTRL-C** direkt till Unix Kermit programmet (*connect* först om det är nödvändigt):

```
CTRL-C CTRL-C
```

Detta gör att programmet avslutas och terminalen återställs till dess vanliga funktion.

7.3. Kermit använt som direktkommando

Syntaxen för Kermit som direktkommando har ändrats från tidigare versioner av Unix Kermit för att följa de regler som specificeras i "proposed Syntax Standard for Unix System Commands". Reglerna som används är:

- Kommandonamnen måste vara 2 till 9 tecken långa.
- Kommandonamnen måste bestå av enbart gemena bokstäver samt siffror.
- Tillval ska anges med en enda bokstav.
- Tillval inleds med "-" (minustecken).
- Tillval utan argument kan föras samman bakom ett minustecken.
- Tillvalsargument kan inte vara frivilliga.
- Argumenten följer omedelbart efter tillval, separerade med ett blanktecken.
- Ordningsföljden på tillvalen spelar ingen roll.
- "-" med blanktecken före och efter indikerar standard input.

En grupp av tillval bakom ett minustecken kan avslutas med ett tillval som kräver argument.

Följande notation används vid kommandobeskrivningen:

fn	Unix filspecifikation, kan innehålla wildcardtecknen "*" eller "?".
fn1	Unix filspecifikation som ej kan innehålla "*" eller "?".
rfn	En filspecifikation för det andra systemet (remote) i dess egen syntax. Specificerar en eller flera filer.
rfn1	En filspecifikation för det andra systemet (remote), men endast en fil.
n	Ett decimalt nummer mellan 0 och 94.
c	Ett decimalt nummer mellan 0 och 127 som representerar ASCII värdet av ett tecken.
cc	Ett decimalt nummer mellan 0 och 31 eller exakt 127 som representerar ASCII värdet av ett kontroll-tecken.
[ ]	Frivilliga tillval.
{x,y,z}	Alternativa val anges inom { }.

Kommandotillvalen för Kermit kan specificera något som ska utföras eller en parameterinställning. Om Kermit startas från en kommandorad och inget ska utföras, svarar Kermit med en prompt och startar den interaktiva dialogen. Tillval som specificerar vad som ska utföras beskriver antingen protokolltransaktioner eller terminaluppkopplingar.

-s *fn* Sänder den eller de specificerade filerna. Om *fn* innehåller wildcard-tecken kommer Unix shell att expandera dessa till en lista. Om *fn* är "-" kommer Kermit att sända från standard input, som måste komma från en fil:

```
kermit -s - < foo.bar
```

eller en parallell process:

```
ls -l | grep kristin | kermit -s -
```

Du kan inte använda denna funktion för att sända tecken från terminalen. Om du vill sända en fil med namnet "-" kan du skriva den med pathname, t ex:

```
kermit -s ./-
```

-r Ta emot en eller flera filer. Väntar passivt på att filerna ska komma.

-k Ta emot (passivt) en eller flera filer och sänd dem till standard output. Detta tillval kan användas på flera sätt:

```
kermit -k
```

Visar den mottagna filen på bildskärmen; används endast i "lokal mod".

```
kermit -k > fn1
```

Sänder den mottagna filen eller filerna till den namngivna filen, *fn1*. Om fler än en fil tas emot, slås alla samman till en fil, *fn1*.

```
kermit -k | kommando
```

Skickar all data som tas emot till det namngivna kommandot, t ex

```
kermit -k | sort > sorted.stuff
```

-a *fn1* Om du har specificerat en filöverföring kan du ange ett alternativt namn för en fil med **-a** tillvalet, t ex:

```
kermit -s foo -a bar
```

sänder filen foo och talar om för mottagaren att filens namn är bar. Om mer än en fil tas emot eller sänds, berörs bara den första filen av **-a** tillvalet:

```
kermit -ra baz
```

lagrar den först mottagna filen under namnet baz.

-x Startar Kermit som server. Kan användas i både local och "remote" mod.

Innan vi fortsätter ska vi säga några ord om användningen av kermit i "remote" och lokal mod. Kermit är lokal om den körs på en arbetsstation som du använder direkt, eller om den körs i ett fleranvändarsystem och överför filer över en extern kommunikationslinje inte över den terminal där du är inloggad. Kermit är "remote" om den körs på ett fleranvändarsystem och överför filer via den kommunikationslinje (normalt **/dev/tty**), där du har loggat in från din arbetsstation.

Följande kommandon ställer Kermit:s mod:

-l *dev* Line - Specificerar en terminallinje för filöverföring och terminalanslutning, t ex

```
kermit -l /dev/tty05
```

När en yttre linje används kan du behöva flera tillval för att få igång kommunikationen med det andra (remote) systemet:

-b n Baud - Specificerar överföringshastigheten för linjen specificerad med **-l**, t ex

```
kermit -l /dev/tty05 -b 9600
```

Detta tillval ska alltid användas tillsammans med **-l** tillvalet, eftersom den yttre linjen kan vara ställd till vilken hastighet som helst.

-p Paritet - e, o, m, s, n (even, odd, mark, space eller none). Om pariteten är annan än none, kommer prefix att användas för att överföra 8 bitars binära data, under förutsättning att den andra (remote) Kermit klarar detta. Standard paritet är none.

-t Specificerar halv duplex, linjevändning med XON som handskakningstecken.

Följande kommandon kan användas med en Kermit i lokal mod, antingen som standard eller p g a att **-l** tillvalet använts.

-g *rfn* Begär att en "remote" server ska skicka namngiven fil eller filer. *rfn* är en filspecifikation i det andra systemets egen syntax. Om *fn* innehåller några speciella shelltecken, t ex "*" måste det citeras, t ex

```
kermit -g x\*.\?
```

eller

```
kermit -g "profile exec"
```

-f Sänd ett *finish* kommando till det andra systemet (remote server).

-c Koppla upp en terminalanslutning över den specificerade kommunikationslinjen (eller standardlinjen) innan någon protokolltrans-

aktion görs. Gå tillbaka till det lokala systemet genom att skriva escape (normalt CTRL-\, följt av bokstaven **c**).

-n Som **-c**, men efter att en protokolltransaktion gjorts. **-c** och **-n** kan användas i samma kommando. Användningen av **-n** och **-c** beskrivs nedan.

På ett fleranvändarsystem ska **-l** och **-b** tillvalen användas tillsammans med **-r**, **-k** eller **-s** tillvalen om Kermit på den andra datorn används i "remote" mod.

Flera andra kommandoradstillval finns tillgängliga:

-i Specificerar att filer ska skickas eller tas emot "som de är" utan konverteringar. Detta tillval är nödvändig vid överföring av binära filer. Den kan också användas för att öka effektiviteten i Unix-till-Unix överföring av textfiler genom att ta bort CRLF/LF konverteringen. Detta gör att en blandning av text- och binärfiler kan överföras.

-w Skrivskydda - Undvik kollision mellan filnamn på mottagna filer.

-e *n* Utökad paketlängd - Specificerar att Kermit kan ta emot paket med upp till *n* teckens längd. *n* kan vara från 10 till ett stort nummer, t ex 1000, beroende på systemet. Som grundvärde är den maximala paketlängden 90 tecken. Paket längre än 94 kommer bara att användas om den andra Kermit har "long packet" protokollutbyggnad.

-q Quiet - Undertryck uppdatering av skärmen under filöverföring, t ex för att köra filöverföringen i bakgrunden.

-d Debug - Lagra information för felsökning i filen **debug.log** i aktuellt bibliotek. Använd detta tillval om du tror att programmet uppför sig felaktigt. Visa sedan loggen för den som ansvarar för din Kermit.

-h Hjälp - Visar en kort sammanfattning av kommandorads tillvalen.

Kommandoraden får inte innehålla mer än en tillval för protokolltransaktion.

Filer sänds med sina egna namn, förutom att gemena bokstäver görs versala, biblioteksdelen av namnen tas bort, vissa tecken som "~" och "#" byts mot "X" och om filnamnet börjar med en punkt sätt in ett "X" framför punkten. Inkommande filer lagras under sina egna namn, förutom att versala bokstäver görs gemena och om **-w** specificerats läggs ett

generationsnummer till efter namnet om den har samma namn som en befintlig fil som annars skulle skrivits över. Om **-a** tillvalen använts används samma regler på dess argument. Alla transformationer av filnamn vid filöverföringen visas.

Under överföringen kodas filerna på följande sätt:

- Kontrolltecken konverteras till vanliga tecken med hjälp av prefix.
- Sekvenser med upprepade tecken ersätts med repetitionstal, under förutsättning att den andra Kermit kan hantera kompression av repeterade tecken.
- Om paritet används vid överföringen konverteras 8-bitars data med en speciell prefixkod. Detta under förutsättning att den andra Kermit kan hantera prefixkoden för 8 bitars data.
- Konvertering mellan Unix newline och CR/LF sekvenser görs om inte **-l** tillvalet specificerats.

Exempel på kommandorader:

```
kermit -l /dev/tty05 -b 1200 -cn -r
```

Detta kommando kopplar upp dig mot det system som är anslutet till tty05 med överföringshastigheten 1200 baud. Du loggar in på det andra systemet och kör Kermit med send kommando. Efter att du gått tillbaka till ditt eget system väntar Kermit på att en eller flera filer kommer. När filöverföringen är klar går du över till det andra systemet och loggar ut.

```
kermit -l /dev/tty05 -b 1200 -cntp m -r -a foo
```

Detta kommando arbetar som det föregående exemplet, med skillnaden att kommunikationen sker med halv duplex och pariteten satt till mark. Den första filen som tas emot lagras under namnet foo.

```
kermit -l /dev/tty05 -b 9600 -c ö tek
```

Detta exempel använder Kermit för att koppla samman din terminal med det system som är anslutet till tty05. Kermit använder sig inte av någon speciell terminalemulering. I detta exempel skickas Kermits standard I/O via en "pipe" genom en (hypotetisk) terminalemulator kallad tek, vilken filtrerar all utdata till din terminal.

```
kermit -l /dev/tty05 -b 9600 -nf
```

Detta kommando kan användas för att stänga en annan server och sedan koppla upp sig mot det systemet för att logga ut eller starta ett annat program. **-n** tillvalet startas efter **-f** (**-c** skulle ha startats före).

```
-l /dev/tty05 -b 9600 -qg foo.\* &
```

Detta kommando gör att Kermit startas i bakgrunden. Programmet hämtar en grupp av filer från en "remote server" (notera citeringen av "*" tecknet). Inget kommer att skrivas ut på bildskärmen och inte heller tangentbordet kommer att kännas av. Detta gör det möjligt att utföra annat arbete under tiden filöverföringen sker i bakgrunden.

```
kermit -l /dev/ttyi6 -b 9600 -g foo.\* > foo.log < | /dev/null &
```

Detta kommando liknar det föregående, men med den skillnaden att filöverföringens utskrift skickas till filen **foo.log**. Även standard input har omdirigerats så att inte Kermit skall känna av avbrottskommandon.

```
kermit -iwx
```

Detta kommando startar Kermit som en server. Filerna sänds utan NL - CR/NL konvertering; **-i** tillvalet är nödvändig för överföring av binärfiler och användbar vid Unix-till-Unix överföringar. Inkommande filer som har samma namn som befintliga filer ges nya namn.

```
kermit -l /dev/tty05 -b 9600
```

Detta kommando väljer kommunikationslinje och överföringshastighet. Eftersom ingen aktivitet specificeras kommer Kermit att skriva ut prompten och övergå i interaktiv mod. Alla parametervärden som ställs på denna rad gäller tills du går ur eller ändrar på dem.

```
kermit
```

Detta kommando startar Kermit interaktivt med alla parametrar satta till standardvärden.

Nedanstående exempel visar hur Unix Kermit kan användas för att skicka ett helt bibliotekskträd från en Unix-maskin till en annan med hjälp av tar som Kermits standard input respektive output. I den dator från vilken filer skall sändas, i detta fall remote, kan följande kommando användas:

```
tar cf - /usr/fdc | kermit -is -
```

Kommandot *tar* skickar nu biblioteket **/usr/fdc** samt alla dess filer och underbibliotek till standard output istället för till magnetmedia. *kermit* tar emot som standard input och lagrar det på en binär fil. På det mottagande systemet, i detta fall det lokala, kan följande kommando användas:

```
kermit -il /dev/tty05 -b 9600 -k | tar xf -
```

Kermit tar emot *tar* arkivet och skickar det via standard output till ett annat tar kommando som lagrar filerna ur bibliotekskträdet som sändes.

Ett sista exempel visar hur Unix filkompression kan användas för att öka filöverföringshastigheten:

```
compress file | kermit -is - (sändare)
kermit -ik | uncompress (mottagare)
```

Utgångsstatus vid terminering

Då kermit avslutas returneras en utgångsstatus med värdet noll om inget allvarligt fel uppstått. Vid fel returneras värdet ett. Om bakgrundskörning har använts (med "&" eller startkommando), som utnyttjat sig av en kommandofil med interaktiva kommandon (via omdirigering eller 'take'-filer), kommer ett kommando som inte går att utföra att orsaka ett avbrott med utgångsstatus 1.

7.4. Interaktiv användning

Kermit's interaktiva kommandoprompt är **C-Kermit>**. Som svar på denna prompt kan du ange alla giltiga kommandon. Kermit utför detta kommando och frågar dig sedan efter ytterligare kommandon. Detta upprepas sedan tills du avslutar programmet.

Alla kommandon inleds med ett nyckelord, ett engelskt verb, t ex *send*. Nyckelorden kan förkortas till ett eller flera tecken, så länge ordet är unikt och skiljer sig från andra nyckelord. Några av de vanligaste nyckelorden (t ex *send, receive* och *connect*) har speciella förkortningar som inte är unika (s för send, r för receive och c för connect).

Vissa tecken har en speciell funktion vid interaktiva kommandon:

?	Frågetecken. När det används i ett kommando kommer en förklaring av vad kommandot kan göra eller vad som förväntas av användaren. Beroende på sammanhang kan det vara en kort mening, en meny över nyckelorden eller en fillista.
ESC	Escape eller Altmode tangenten - Begär att Kermit ska komplettera nyckelordet, filnamnet eller sätta in standard-värdet. Ett pip (BELL) kommer att höras om begäran inte kan genomföras.
DEL	Delete eller Rubout tangenten - Ta bort närmast föregående tecken. Det går även att använda BS (Backspace, CTRL-H) för detta.
^W	CTRL-W - Ta bort det sista ordet på kommandoraden.
^U	CTRL-U - Ta bort hela kommandot.
^R	CTRL-R - Visa det aktuella kommandot igen.
SP	Space - Avskiljer fälten (nyckelord, filnamn och siffror) i ett kommando. HT (Horizontal Tab) kan också användas.
CR	Carriage Return - Startar utförandet av kommandot.
LF	(Linefeed) eller FF (formfeed) kan också användas.
****	Backslash - Gör det möjligt att använda ovanstående tecken i kommandot. För att verkligen skriva in tecknet \ som ett tecken skriver du två backslash i rad (\\). En backslash i slutet på en kommandorad gör att nästa rad tolkas som fortsättningen på den tidigare raden. Detta är bra ur läsbarhets-

synpunkt i kommandofiler, specielt i "script" kommandot.

Du kan använda redigeringstecknen (DEL, ^W, etc) flera gånger i rad, t ex för att radera allt tillbaka till prompten. Ingenting kommer att göras innan kommandot startats med carriage return, linefeed eller formfeed. Om du skulle göra fel får du ett felmeddelande och en ny prompt - använd gärna "?" och ESC för att lära dig använda kommandona. Ett viktigt kommando är *help* som du bör använda första gången du kör Kermit.

Kermit tar även emot interaktiva kommandon från filer. När du startar Kermit i interaktiv mod, söker programmet först efter filen **.kermrc** i ditt hembibliotek eller det aktuella biblioteket (först kontrolleras hembiblioteket). Om filen finns utförs de kommandon som finns lagrade i filen. Dessa måste vara lagrade som interaktiva kommandon och inte på kommandoradsformat. Kommandot take kan också användas när som helst vid interaktiv användning. Det går att anropa en kommandofil inifrån en annan i flera steg om det skulle behövas.

Här följer en kort beskrivning av interaktiva kommandon:

Kommando	**Funktion**
!	Utför ett Unix shell kommando.
bye	Avsluta och logga ut en remote kermit server.
close	Stäng en loggfil.
connect	Koppla upp en terminalanslutning till en remote server.
cwd	Byt aktuellt bibliotek.
dial	Slå ett telefonnummer.
directory	Visa innehållet i ett bibliotek.
echo	Visa argumentet exakt.
exit	Avsluta programmet och stäng alla öppna loggfiler.
finish	Avsluta en remote kermit server utan att logga ut.
get	Hämta filer från en remote Kermit server.
help	Visa hjälpmeddelande för ett angivet kommando.
log	Öppna en loggfil - debugging, packet, session, transaction.
quit	Samma som *exit*.
receive	Vänta passivt på att filer ska komma.

remote	Skickar filhanteringskommandon till en remote Kermit server.
script	Utför en login script med ett remote system.
send	Skicka filer.
server	Starta systemet som server.
set	Ändra parametrar.
show	Visa aktuellt värde på parametrar.
space	Visa utrymme på massminne.
statistics	Visa statistik på de senaste transaktionerna.
take	Utför kommandon lagrade i en fil.

Parametrarna för *set* kommandot är:

Parameter	**Funktion**
block-check	Nivå av felupptäckt i paket.
delay	Väntetid före sändning av det första paketet.
duplex	Specificerar vilken sida som ska eka under connect kommandot.
end-of-packet	Sluttecken för utgående paket.
escape-character	Prefix-tecken för "escape commandon" under *connect*.
file	Ändra vissa filparametrar.
flow-control	Flödeskontroll vid full-duplex på kommunikationslinjen.
handshake	Styrtecken för halv-duplex kommunikation.
incomplete	Hantering av ej komplett mottagna filer.
line	Namn på enhet för kommunikationslinjen.
modem-dialer	Typ av modemuppringare på kommunikationslinjen.
parity	Kommunikationslinjens teckenparitet.
prompt	Ändra Kermit:s prompt.
receive	Parametrar för inkommande paket.
retry	Gräns för antal omförsök.
send	Parametrar för utgående paket.
speed	Överföringshastighet på kommunikationslinjen.
terminal	Terminalparametrar.

Parametrarna för *remote* kommandot är:

Parameter

	Funktion
cwd	Ändra aktuellt bibliotek, remote.
delete	Ta bort filer, remote.
directory	Visa filnamn, remote.
help	Begär hjälptexter från remote server.
host	Skicka ett kommando till en remote värd i dess eget kommandospråk.
space	Visa utrymme på massminne, remote.
type	Visa en fil, remote, på din skärm.
who	Visa vilka som är inloggade eller information om en användare.

De flesta av dessa kommandon beskrivs i Kermit Users Guide. Vissa speciella aspekter på Unix Kermit kommandon beskrivs nedan.

send

SYNTAX

```
send fn
send fn1 rfn1
```

Skicka en eller flera filer specificerade i *fn* till en annan Kermit som körs som server eller som har fått kommandot *receive*. Filerna skickas under sina egna namn (enligt beskrivningen ovan, eller under de namn som specificeras med set file names kommandot). Om den andra formen används, d v s *fn1* är en enda fil, kan *rfn1* specificera namnet som den ska sändas under. *send* kommandot kan förkortas till *s* trots att inte s är en unik förkortning av kommandonamnet.

Metatecknen "*" och "?" kan användas i *fn*. Om "?" används, måste det föregås av prefixtecknet \ för att undvika att det tolkas som hjälpkommando. "*" matchar alla strängar, "?" matchar ett tecken vilket som helst. Andra metatecken, t ex "[a-z]og" som namn på filgrupper kan inte användas vid interaktiva kommandon. De kan däremot användas från en Unix kommandorad. Om *fn* innehåller en "*" eller ett "?" finns det en gräns för hur många filer som kan matchas. Gränsen är systemberoende. Om du får meddelandet **Too many files match** måste du göra ett smalare urval av filer. Om *fn* t ex ser ut på följande sätt:

```
/usr/longname/anotherlongname/*
```

kommer Kermit:s strängutrymme att fyllas ut snabbt. Gör då en *cwd* (se nedan) till biblioteket och skicka kommandot igen.

Observera! Kermit skickar bara från aktuellt eller specificerat bibliotek. Den söker inte genom trädstrukturer. Om biblioteket innehåller underbibliotek kommer dessa att hoppas över. På samma sätt skapar Kermit aldrig bibliotek när det tar emot filer. Om du behöver göra detta kan du köra *tar* genom Kermit som visades ovan i kommandoradsexemplen. En annan möjlighet är att använda *cpio*.

Observera! Kermit hoppar inte över "osynliga" filer som matchar filspecifikationen. Normalt hanterar Unix-system filer som börjar med en punkt (t ex **.profile** och **.kermrc**) som "osynliga". På samma sätt hanteras "temporär" filer som börjar med "#".

receive

SYNTAX

```
receive
receive fn1
```

Vänta passivt på att filer ska komma från den andre Kermit. Denna måste ges send kommandot. *receive* kommandot fungerar inte tillsammans med en server (använd *get* vid dessa tillfällen). Om *fn1* specificerats kommer den först mottagna filen att lagras under detta namn. *receive* kommandot kan förkortas till *r*.

get

SYNTAX

```
get rfn
get
```

Begär att en remote Kermit ska skicka den eller de filer som anges. Filspecifikationen (eller listan) kan innehålla mellanslag trots att dessa normalt används som fältavskiljare i Kermit. Kommandot kan även användas för att ge den inkommande filen ett nytt namn; skriv *get* ensamt på en rad och Kermit kommer att fråga dig efter filnamnet, både på remote systemet och det lokala systemet. Exempel:

```
C-Kermit> get
Remote file specification: profile exec
Local name to store it under: profile.exec
```

På samma sätt som *receive* kommer bara den första filen som tas emot att ges ett nytt namn. Om flera filer kommer, lagras de under sina egna namn om det är möjligt. Om ett "?" skall finnas med i filspecifikationen till remote systemet, måste prefixtecknet \ användas. I annat fall tolkas "?" som en förfrågan om hjälp.

Om du har startat ett flera raders *get* kommando kan du avbryta från dess lågnivåpromptar genom att svara med **RETURN** på prompten, t ex:

```
C-Kermit>get
Remote file specification: foo
Local name to store it under: (return)
(cancelled)
C-Kermit>
```

server

SYNTAX

```
server
```

server kommandot aktiverar Kermit i server mod på den aktuella kommunikationslinjen. I fortsättningen måste alla kommandon komma som korrekt formaterade Kermit-paket från det andra Kermit systemet. En Unix Kermit server kan tolka följande inkommande kommandon:

Kommando	Server funktion
get	Skicka filer.
send	Ta emot filer.
bye	Försöker logga ut sig själv.
finish	Gå tillbaka till den nivå varifrån den startades.
remote directory	Skicka bibliotekslista.
remote delete	Ta bort filer.
remote cwd	Ändra aktuellt bibliotek.
remote type	Skicka filer till din bildskärm.
remote space	Rapportera utrymme på massminnet.
remote who	Visa vilka som är inloggade.
remote host	Utför ett Unix shell kommando.
remote help	Visa dessa kommandon.

Observera! Unix Kermit kan inte alltid utföra *bye* kommandot. Den försöker utföra detta genom att ge systemanropet *kill()*, men detta fungerar inte alltid på alla system p g a den komplicerade processstruktur som kan skapas under Unix.

Om Kermit server är ansluten till en yttre linje (d v s den är i lokal mod) kan terminalen användas för annat arbete om du satt parametern *set file display off*. Normalt förväntar sig programmet att terminalen används för att kontrollera filöverföring, eller för statuskontroller eller avbrottskommandon. För att få Kermit att arbeta i bakgrunden i interaktiv mode krävs ett systemspecifikt kommando. Den vanligaste metoden är att starta programmet i kommandoradsmod med tillvalet **-q** och med tecknet **&** i slutet av raden.

När en Unix Kermit server ges kommandot *remote host* använder det sig av den shell som startades vid login till remote systemet. T ex Bourne shell eller Berkely C-shell.

remote, bye och finish

Kermit kan själv begära tjänster från en remote Kermit server. Förutom kommandona *send* och *get*, finns även följande kommandon:

remote cwd [directory]	Om biblioteksspecifikationen tas med, kommer du att få en fråga efter ett lösenord. Lösenordet kommer inte att skrivas ut på skärmen under inmatning.
remote delete rfn	Ta bort fil eller filer (remote)
remote directory [rfn]	Biblioteksutskrift av filer, remote.

remote host command	Kommando i värddatorns eget kommandospråk.
remote space	Utrymme på massminne, remote.
remote type [rfn]	Visa fil eller filer på bildskärmen, remote.
remote who [user]	Visa vilka som är inloggade.
remote help	Visa vilka funktioner som finns tillgängliga, remote.
bye och finish	När du är uppkopplad till en remote Kermit server avslutar dessa kommandon remote servern, finish återgår till Kermit eller kommandonivån i systemet. bye loggar dessutom ut sig från systemet.

log, close

SYNTAX

```
log {debugging, packets, session, transactions} [fnl]
```

Kermits aktiviteter kan loggas på flera sätt. Kommandot *log* öppnar en loggfil och kommandot *close* stänger den. Dessutom stängs alla öppna loggar när kommandona *exit* och *quit* används. Loggfilen kan namnges, men om namnet utelämnas ges den namn enligt standard nedan.

```
log debugging
```

Detta kommando ger en mycket omfattande log av allt som sker i Kermit. Det kan användas av den Kermit ansvarige för att hitta misstänkta fel i Kermit programmet. När detta kommando angetts blir filöverföringen mycket långsam. Standard filnamn: **debug.log**.

```
log packets
```

Detta kommando ger en logg över alla paket som skickas eller tas emot på kommunikationsporten. Loggen är avsedd för den Kermit ansvarige och är användbar då problem i överföringen kan bero på någon av de två Kermit programmen. Standard filnamn: **packet.log**.

```
log session
```

Denna logg kommer att innehålla allt som visas på din skärm under connect kommandot, utom lokala meddelanden eller när du arbetar med lokala escape kommandon. Standard filnamn: **session.log**.

```
log transactions
```

Denna logg håller reda på alla filer som skickats eller tagits emot under den tid som tranaktionsloggen varit aktiv. Den innehåller tidsmarkeringer, statistik, filnamnstransformationer och alla fel som har uppstått. Transaktionsloggen ger dig möjlighet att göra långa filöverföringar i bakgrunden utan att du missar viktig information. Standard filnamn: **transaction.log**.

Kommandot close stänger en log, t ex *close debug*.

Observera! Debug och Transaction logar är tillval som kan väljas vid kompileringen; Kermit kan kompileras utan dessa loggar. Om detta görs kommer programmet att bli snabbare, ta mindre plats på skivminnet och dessa kommandon kommer heller inte att finnas tillgängliga.

Lokala filhanteringskommandon

Unix Kermit tillåter viss lokal filhantering med hjälp av interaktiva kommandon:

```
directory [fn]
```

Visar en lista över namn, moder, storlekar och datum för filer som matchar *fn* (standard är "*"). Motsvarar *ls -l*.

```
cwd [biblioteksnamn]
```

Ändrar Kermits aktuella bibliotek till det angivna, eller till hembiblioteket om inget namn ges. Kommandot påverkar bara Kermit processen samt processer som den senare skapar.

```
space
```

Visar information om utrymme på massminnet för det aktuella biblioteket och den aktuella enheten.

```
!kommando
```

kommando kommer att köras under Unix shell. Om inget kommando anges startas en interaktiv shell. När du går ur shellen, med CTRL-D eller *exit*, kommer du tillbaka till Kermits kommandonivå. Använd "!" kommandot för att sköta filhantering m m som inte har egna funktioner i Kermit. Vissa begränsningar finns dock:

- Kermit försöker att använda din normala shell (login).
- Minst ett mellanslag måste skilja ! från shellkommandot.
- Ett *cd* kommando kommer inte att fungera använd istället Kermit kommandot *cwd*.

set

Då Kermit är avsett att användas även när olika datorsystem ska kommunicera är det ofta nödvändigt att instruera programmet vilka parametrar som ska gälla vid kommunikationen. Detta sker med *set* kommandot. Med *show* kommandot kan du på bildskärmen visa hur parametrarna är satta. Här följer en kort beskrivning av vilka parametrar som finns tillgängliga i denna version av Kermit:

```
set block-check {1, 2, 3}
```

Parametern bestämmer vilken typ av feldetektering som ska användas. *1* är en 6-bitars kontrollsumma som är "vikt" för att alla bitar i varje tecken ska ingå. *2* är en 2 tecken lång 12-bitars kontrollsumma. *3* är en 3 tecken lång 16-bitars CRC-summa (cyclic redundancy check). Rent generellt kan sägas att en högre siffra ger en bättre felupptäckt och möjlighet till felkorrektion, men samtidigt kräver de större beräkningar. Typ 1 är den mest använda och understödjs av alla Kermit implementationer och är fullt tillräcklig i de flesta fall. Typ 2 och 3 har sina största fördelar då binärfiler ska överföras på dåliga linjer.

```
set delay n
```

Väntetid i sekunder innan det första paketet ska skickas efter ett send kommando. Används i remote mod för att ge dig tid att hoppa tillbaka till din lokala Kermit och ge *receive* kommandot. Normalt är väntetiden fem sekunder.

```
set duplex {full, half}
```

Används tillsammans med kommandot connect. Specificerar vilken sida som ska eka tecknen. *full* betyder att den andra sidan ekar och *half* betyder att Kermit själv ska eka de tecknen som skrivs.

```
set escape-character cc
```

Används tillsammans med connect för att få kontakt med Kermit. Tecknet fungerar som prefix för escape kommandon. T ex för att stänga en uppkoppling och återgå till Kermit eller till Unix kommando nivå. Normalt är detta tecken CTRL-\ (ASCII-tecknet 28 decimalt). Escape tecknet används i System V Kermit även som prefix för avbrottskommandon under filöverföring.

```
set file {display, names, type, warning}
```

Fastställer vissa parametrar för filer:

```
set display  {on, off}
```

Normalt *on*. I lokal mod visas filöverföringen på bildskärmen (stdout) och tangentbordet känns av (stdin) för avbrott. När *off* används (**-q** på kommandorad), görs inget av detta. Detta kan användas för filöverföring i bakgrunden och påverkas inte av att annat arbete fortgår parallellt på terminalen.

```
set names {converted, literal}
```

Normalt *converted*, vilket innebär att utgående filnamn har pathnamnet borttaget, gemena bokstäver omvandlade till versaler, tecknet ~ och extra punkter omvandlade till X och ett extra X i början på filnamn som börjar med en punkt. Inkommande filnamn får versaler omvandlade till gemener. *literal* innebär att ingen omvandling görs. Det gör att alla pathnamn som finns i ankommande filer måste finnas på den mottagande datorn och vara skrivbara. När *literal* används ska avsändaren inte använda pathnamn i filspecifikationen om inte samma path finns på det mottagande systemet.

```
set type {binary, text} [{7, 8}]
```

Normalt *text*, vilket innebär att en omvandling mellan Unix NL och CR/LF görs. Detta behövs för kommunikation med Kermit som arbetar på datorer som inte har Unix. *binary* innebär att filöverföringen sker utan omvandling. *binary* (**-i** på kommandorad) krävs för överföring av binära filer och är att föredra även vid överföring mellan olika Unix-system då överföringstiden minskar.

Om den avslutande parameter används beskriver den bytestorleken vid överföringen. Standardvärdet är *8*. Om du specificerar *7*, kommer den högsta biten att tas bort från varje byte som sänds eller tas emot. Detta kan vara bra vid överföring av textfiler där den högsta biten är satt för att beskriva attribut eller liknande vid lagringen (t ex WordStar och andra liknande program).

```
set warning {on, off}
```

Normalt *off*, vilket innebär att inkommande filer skriver över redan befintliga filer med samma namn. Om den sätts *on* (**-w** på kommandorad) kontrollerar Kermit alla inkommande filer och döper om dem om de skulle skriva över någon fil. Det nya namnet får formen foo~n, där foo

var det ursprungliga namnet och n är ett generationsnummer. Om filen foo redan finns kommer den nya filen att heta foo~1. Och om foo och foo~1 finns blir det nya namnet foo~2 osv. Om det nya namnet blir längre än det maximalt tillåtna för filnamn raderas tecken från slutet först, t ex thelongestname på ett system med 14 tillåtna tecken skulle bli thelongestn~1.

Varning: Om CTRL-F eller CTRL-B används för att avbryta en inkommande fil och en fil med samma namn redan fanns samtidigt som *file warning* är avslagen, kommer den tidigare filen att raderas.

```
set flow-control {none, xon/xoff}
```

Normalt *xon/xoff* för flödeskontroll vid full duplex. Parametern ska sättas till *none* om det andra systemet inte kan hantera XON/XOFF, eller om du har gett ett *set handshake* kommando. Om den sätts till XON/XOFF skall handshake sättas till *none*. Detta gäller både vid filöverföring och vid terminalanslutning.

Varning: Detta kommando har ingen funktion på vissa Unix-system där *kermit* sätter kommunikationslinjen i "rawmode", och där detta utesluter flödeskontroll.

```
set incomplete {discard, keep}
```

Hantering av filer som inte kommit fram i sin helhet. Om en inkommande fil avbryts eller ett fel uppstår vid överföringen, kommer den mottagna delen normalt inte att sparas. Om du ger kommandot set incomplete keep kommer dessa delar att sparas.

```
set handshake {xon, xoff, cr, lf, bell, esc, none}
```

Normalt *none*. I annat fall utförs handskakning för att vända sändningsriktningen vid halv duplex. Det innebär att Unix Kermit inte svarar på ett paket innan det indikerade handskakningstecknet mottagits eller väntetiden har gått ut. Handskakningen används bara vid filöverföring. Om du sätter handskakning till något annat än *none* ska du sätta *flow* till *none*.

```
set line {enhetsnamn}
```

Enhetsnamnet för kommunikationslinjen som används för filöverföringen och terminalanslutningen, t ex **/dev/tty05**. Om du specificerar ett enhetsnamn kommer Kermit att arbeta i lokal mod. Kom då ihåg att utföra de *set* kommandon som behövs, t ex *set speed*. Om du utelämnar enhetsnamnet kommer Kermit att återgå till standard-värdena. Om du specificerar **/dev/tty** kommer Kermit att ställa sig i remote mod. När Unix Kermit går in i lokal mod försöker den att synkronisera sig med andra program (t ex *uucp*) som använder sig av yttre kommunikationslinjer så att inte de två programmen försöker att använda sig av samma linje samtidigt. Innan den försöker att låsa en specifik linje kommer den att stänga och låsa upp tidigare använda linjer. Metoden som används för låsning av linjer är **uucp lock file**, som kommer att beskrivas senare.

```
set modem-dialer {direct, hayes, racalvadic, ventel ..}
```

Typ av modemuppringare på kommunikationslinjen. *direct* används om modemet ej kan ringa upp eller att linjen behöver känna en carrier innan den kan öppnas. I detta fall kommer kommandot *set line* att vänta

på ett inkommande samtal. *Hayes* och *Ventel* innebär att påföljande set line kommando (eller -l på kommandoraden) förbereder för nästa *dial* kommando för Hayes och Ventel modem. Stöd för nya modemuppringare läggs till kontinuerligt, så ge kommandot *set modem ?* så får du se en lista över vilka modem som understöds i din version av Kermit. Se beskrivningen av *dial* kommandot.

```
set parity {even, odd, mark, space, none}
```

Bestämmer vilken paritet som ska användas för paket och terminalanslutning, normalt *none*. Om annat än *none* specificeras använder sig Kermit av prefix för överföring av 8-bitars binära data. Denna funktion kan bara användas om det andra systemet klarar av det. Om inte, kan inte 8-bitars data överföras.

```
set prompt {sträng}
```

Den angivna strängen kommer att ersätta **C-Kermit>** som programmets prompt. Om strängen utelämnas kommer systemet att återgå till C-Kermit>. Om strängen är innesluten av ", kommer dessa att tas bort och mellanslag före eller efter prompten att behållas.

```
send parameter
```

Bestäm parametrar vid sändning av paket. Dessa kommer att användas bara för det första paketet som sänds, då den andra Kermit kan ändra dessa under utbytet av protokollparametrar (om inte annat anges nedan).

```
end-of-packet cc
```

Specificerar det kontrolltecken som behövs för att den andra Kermit ska känna igen slutet på ett paket. Kermit skickar detta tecken i slutet av varje paket. Normalt är det 13 (cariage return), som de flesta Kermit använder sig av. En del använder sig dock inte av något kontrolltecken alls, medan andra använder sig av t ex linefeed (10).

```
packet-length n
```

Bestämmer maximala paketlängden som får användas, normalt 90. Kortare paketlängder kan vara användbart på dåliga linjer eller vid kommunikation med system med små buffertar. Ett kortare paket ger sämre överföringseffektivitet men mindre risk för felaktiga överföringar och kortare tid vid omsändning av felaktiga paket. Detta kommando går före det värde som den andra Kermit begär under protokollinitieringen, under förutsättning att den andra Kermit inte begär en kortare paketlängd.

```
pad-character cc
```

Bestämmer vilket tecken som skall sändas före varje paket. Normalt sänds inget tecken. Detta tecken behövs ibland för att kommunicera med långsamma halv duplex system som måste ställas i "transparent" eller "no echo" mod med hjälp av något kontrolltecken.

```
padding n
```

Hur många utfyllnadstecken som ska användas, normalt 0.

```
start-of-packet cc
```

Prefixtecknet för ett Kermit paket är CTRL-A (01H). Enda anledningen att ändra detta tecken är om någon utrustning mellan de två Kermit

programmen inte kan överföra CTRL-A eller om en utrustning ekar allt som kommer in. I det senare fallet kan mottagaren av detta eko ändra prefix för utgående paket så det blir skilt från prefix för inkommande paket varvid ekade paket kan ignoreras. I så fall måste även den andra Kermit göra motsvarande prefix-ändring för dess inkommande paket. I dagsläget kan Unix Kermit bara ändra prefix för utgående paket.

```
timeout n
```

Ange det antal sekunder som du vill att den andra Kermit ska vänta på ett paket innan den gör "time out" och begär omsändning av paketet.

```
receive parameter
```

Bestäm parametrar som den andra Kermit ska använda när den sänder paket.

```
end-of-packet cc
```

Begär att den andra Kermit ska avsluta sina paket med det specificerade tecknet.

```
packet-length n
```

Bestämmer maximala paketlängden som den andra Kermit ska sända, normalt 90. Om du specificerar en längd av 95 eller större kommer denna att användas om den andra Kermit har stöd för långa paket. I detta fall beror den maximala paketlängden på det andra systemet, men det finns ingen anledning att använda paket längre än 1000 tecken. Kommandot *show parameters* visar Kermits aktuella samt maximala paketlängder.

```
pad-character cc
```

Normalt behöver inte Kermit ha utfyllnadstecken före de inkommande paketen. Men detta kommando ger Kermit möjlighet att specificera att det andra systemet ska använda ett visst tecken för utfyllnad, *cc*. Normalt är *cc* NUL (00H).

```
padding n
```

Hur många utfyllnadstecken som ska begäras, normalt inget.

```
start-of-packet cc
```

Ändra inledningstecknet som Kermit väntar på vid inkommande paket till det som den andra Kermit sänder.

```
timeout n
```

Normalt sätter varje Kermit sin timeout till det värde som motparten automatiskt anger. Detta kommando gör det möjligt att ändra på denna och specificera ett annat värde. Om du specificerar 0 kommer ingen timeout att ske och Unix Kermit kommer att vänta i evighet på ett förväntat paket.

```
set speed {0,110,150,300,600,1200,1800,2400,4800,9600}
```

Överföringshastigheten för den yttre kommunikationslinjen. Kommandot kan inte användas för att ändra hastigheten till din lokala terminal. Många system är uppkopplade så att detta kommando måste ges sedan du använt set line kommandot innan du kan använda linjen. *set baud* är en synonym till *set speed*.

```
terminal
```

Används för att specificera terminalparametrar. För tillfället är *bytesize* den enda parametern och den kan sättas till 7 eller 8. Standardvärdet är 7.

show

SYNTAX

```
show {setparametrar, versions}
```

Kommandot *show* visar värdena på alla set parametrar som beskrivs ovan. Om du anger *show versions* kommer Kermit att svara med versionsnummer och datum för dess interna moduler. *show versions* kan användas för att bestämma vilken Kermit du har innan du rapporterar problem till den Kermit ansvarige.

statistics

Kommandot *statistics* visar information om den senast utförda överföringen, även I/O till fil och kommunikationslinje. Kommandot visar även om någon kodning gjorts av informationen som sänts eller tagits emot, t ex prefix av 8-bits data eller komprimering av repeterade tecken.

take, echo

SYNTAX

```
take fnl
echo [text to be echoed]
```

Kommandot *take* talar om för Kermit att den ska utföra kermit-kommandon som finns i en namngiven fil. Filen kan innehålla interaktiva kommandon, även *take*. Kommandofiler kan nästlas till en viss systemberoende gräns, men de får då inte innehålla text som skall sändas till det andra systemet under *connect* kommandot. Detta betyder att en kommandofil som ser ut på detta sätt:

```
set line /dev/tty17
set speed 9600
connect
login myuserid
mypassword
etc
```

inte kommer att sända **login myserid** eller någon av de följande texterna till det andra systemet. För att utföra en dialog enligt ovan bör kommandot *script* användas, beskrivs nedan.

Kommandot *%* är användbart för att skriva kommentarer i en takekommandofil. Kommandot kan bara användas i början på en rad.

Kommandot *echo* kan användas i kommandofiler för att skriva ut meddelanden till användaren m m. *echo* kommandot skall inte förväxlas med Unix *echo* kommando som kan användas för att visa vilka metatecken som kommer att expanderas. Kermits *echo* kommando skriver endast ut textargumentet på terminalen. Argumentet kan innehålla oktala sekvenser på formen \ooo där o är en oktal siffra (0-7) och det kan finnas en, två eller tre siffror. Dessa specificerar ett ASCII-tecken, t ex \012 (eller \12) för newline. För att skriva tecknet \ måste det skrivas två gånger, då Kermits kommandotolkare tar bort det första.

Kommandofilerna använder exakt samma syntax som de interaktiva kommandona.

Observera! Detta innebär att alla specialtecken som frågetecken och \ måste "citeras" med \ (backslash) även i kommandofilerna, precis på samma sätt som i interaktiv mod. Långa rader kan brytas genom att avsluta dem med "/".

Kommandofiler kan användas istället för makrokommandon eftersom makron inte har implementerats i denna version av Kermit. T ex om du ofta kommunicerar med ett system kallat "B" som är anslutet till tty05 med överföringshastigheten 4800 baud kan du skapa följande fil och kalla den för **be**:

```
set line /dev/tty05
set speed 4800
echo Connecting to system B...
connect
```

och sedan utföra dessa kommandon genom att skriva *take be* (eller *t be* eftersom inga andra kommandon börjar med "t") varje gång du vill koppla upp dig mot system B.

Vid uppkoppling mot IBM stordatorer behövs ett antal set kommandon. Dessa kan samlas i en take fil som kan se ut på följande sätt:

```
set speed 1200
set parity mark
set handshake xon
set flow-control none
set duplex half
```

Observera att det inte finns något kommando för att återställa alla dessa inställningar och ställa Kermit i samma läge som när det startades. För att göra detta måste du antingen starta om programmet eller skapa en kommandofil som utför de nödvändiga *set* kommandona:

```
% Sample Kermit command file to restore settings
%
set parity none
set handshake none
set flow-control xon/xoff
set duplex full
```

Varje gång Kermit startas i interaktiv mod utförs automatiskt kommandot take på din .kermrc-fil. Filen **.kermrc** ska därför innehålla de *set* kommandon eller andra kommandon som du alltid vill utföra då kermit startas. Du kanske vill ha en markering om inkommande filer har samma namn som redan existerande, skriv då:

```
set file warning on
```

i din **.kermrc** fil. I vissa icke-Unix system som använder Kermit har initialiseringsfilen ett annat namn, t ex **kermit.ini**.

Fel som uppstår under utförandet av kommandofiler (t ex en misslyckad uppringning) avbryter utförandet av den aktuella kommandofilen och går tillbaka till föregående nivå (kommandofil, interaktiv mod eller Unix shell). Om Kermit körs i bakgrunden kommer ett fel att avbryta körningen helt.

I Unix kan du också använda omdirigering av en fil för att få Kermit att köra kommandon från filen:

```
kermit < cmdfile
```

eller du kan använda en 'pipe' från en annan process:

```
cmdprocess | kermit
```

connect

Kommandot *connect* kopplar samman din terminal med den andra datorn som om den vore en lokal terminal till datorn. Kopplingen sker genom den enhet som angavs i det senaste set line kommandot eller genom standard-enheten om du har en arbetsstation eller PC. Alla tecken du skriver sänds via kommunikationslinjen och alla tecken som tas emot visas på din bildskärm. De aktuella parametervärdena för hastighet, paritet, duplex och flödeskontroll används, och 7-bitars överföring används om du inte get kommandot *set terminal bytesize 8*. Om du har använt ett *log session* kommando kommer allt som visas på bildskärmen dessutom att sparas i din sessions logg. Det gör det möjligt att hämta filer på system som inte har något Kermit program tillgängligt.

För att komma tillbaka till ditt eget system måste du skriva escape tecknet, dvs CTRL-\ om du inte ändrat detta tecken med *set escape* kommandot. Detta skall direkt följas av ett enbokstavs kommando, t ex "c" för "close connection". Bokstavskommandona är följande:

c	Koppla ner uppkopplingen
b	Skicka en BREAK signal
0	(noll) skicka en NULL
s	Statusrapport över uppkopplingen
h	Koppla ned linjen (Lägg på luren)
^	Skicka en CTRL-\ (det du har definierat som escape tecken kan skrivas in om du skriver den två gånger i rad)

Gemena bokstäver och motsvarande kontrolltecken för dessa kommandon kan också användas. Ett mellanslag efter escape tecknet ignoreras. Alla andra bokstäver genererar en ton (BELL) i terminalen.

Kommandot *connect* visar bara alla inkommande tecken på bildskärmen. Det förutsätts att de styrtecken som sänds av värddatorn hanteras av din terminal eller PC. Om du behöver använda terminalemulering kan connect kommandot användas med kermit som direktkommando (**-c** eller **-n**) och omdirigeras via en pipe genom ett terminalemuleringsfilter, t ex:

```
kermit -l /dev/acu -b 1200 -c | tek
```

dial

SYNTAX

```
dial telefonnummer-sträng
```

Detta kommando styr uppringande modem. Innan det ges skall *set line* och *set speed* kommandon ha getts för att identifiera den typ av modem som ska användas vid uppringningen. I *dial* kommandot anger du telefonnumret och Kermit skickar det till modemet i rätt format. Kermit tar sedan hand om de kommandon som returneras från modemet och talar om för dig om uppringningen gått bra. Telefonnummer-strängen får in-

nehålla kommandon till modemet, t ex kommatecken (,) för paus med Hayes-modem, eller "&" för Ventel-modemets vänta på kopplingston eller % för paus med Ventel-modem.

När detta skrivs finns stöd för följande modemtyper:

- Cermetek Info-Mate 212A
- DEC DF03-AC
- DEC DF100 Series
- DEC DF200 Series
- General DataComm 212A/ED
- Hayes Smartmodem 1200
- Racal Vadic
- US Robotics 212A
- Ventel

Stöd för ytterligare modem läggs hela tiden till i programmet, för att se vilka modem som understöds kan du ge kommandot *set modem ?*.

Den enhet som används för uppringningen är den som valdes med det senaste *set line* kommandot. *dial* kommandot låser sedan denna linje (se avsnittet om linjelåsning nedan) och utför kommandot. Om du behöver ringa upp och därefter gå tillbaka till shell måste uppringningen ske på ett enhetsnamn (t ex **/dev/tty05**) och sedan gå till shell från Kermit på en länkad enhet som är fristående från den linje som används för uppringningen (t ex **/dev/tty05**). Detta är samma teknik som används av *uucp*.

Eftersom uppringande modem har strikta krav på att nonchalera CD signalen som de flesta Unix-system väntar sig är kommandosekvensen för uppringning betydligt striktare än Kermits andra kommandon.

Exempel 1:

```
kermit -l /dev/cul0 -b 1200

C-Kermit>set modem-dialer hayes
C-Kermit>dial 9,5551212
Connected!
C-Kermit>connect

diverse kommandon

C-Kermit> ...
C-Kermit>quit
```

detta kopplar ner modemet och frigör linjen.

Exempel 2:

```
kermit

C-Kermit>set modem-dialer hayes
C-Kermit>set line /dev/tty05
C-Kermit>dial 9,5551212%
Connected!
C-Kermit> ...
```

Exempel 3:

```
kermit

C-Kermit>take min-ringfil
Connected!
```

Filen min-ringfil:

```
set modem hayes
set line /dev/tty05
dial 5551212
connect
```

Rent generellt kräver Kermit att modemet ger CD signalen när en uppringning håller på och att signalen tas bort när en uppringning är avslutad eller när linjen går ner. Om det finns en omkopplare för att tvinga på en CD signal skall denna inte användas. Kermit kräver också (på de flesta system) att modemet följer datorns DTR signal. Om det finns en omkopplare för att simulera DTR i modemet skall denna normalt inte användas. Om detta inte är fallet kan modemet inte frigöra linjen vid slutet av uppkopplingen eller när avbrott kommer från Kermit.

För Hayes modem finns det två viktiga switchinställningar, #1 och #6. #1 ska sättas så att DTR bara används när linjen är öppen. #6 ska sättas upp så att carrier-detect sker korrekt. Switcharna #2 och #4 kan vara i vilken position som helst. (Det betyder annars: #2 = koder som engelsk text eller siffror, #4 = Hayes ekar modemkommandon).

Om du vill avbryta ett *dial* kommando som håller på, skriv ett CTRL-C för att återgå till kommandonivån.

script

SYNTAX

```
script expect send [expect send] ...
```

expect har syntaxen:

```
expect[-send-expect[-send-expect[...]]]
```

Detta kommando ger möjlighet att logga in på ett remote system och/eller starta program eller liknande efter inloggning på systemet.

Login script möjligheten fungerar på ungefär samma sätt som det Unix använder för *uucp* och **L.sys** filen. En login script är en sekvens i följande form:

```
expect send [expect send] ...
```

där *expect* är en prompt eller en fråga som förväntas från remote systemet. send är den sträng (namn, nummer m m) som ska skickas som svar. *send* kan även vara strängen EOT för att skicka CTRL-D eller BREAK för att skicka avbrottsignalen. Tecken i *send* kan även föregås av "~" för att skicka speciella tecken. Dessa är:

~b	Backspace
~s	Mellanslag
~q	"?" (fångas av Kermits kommandotolk)
~n	Radframmatning
~r	Return
~t	Tab
~'	Enkel apostrof
~~	Tilde-tecken
~"	Citationstecken
~c	Lägg inte till Return eller svaret
~o[o[o]]	Ett tecken angivet med oktalt värde
~d	1/3 sekunds fördröjning under sändning
~w[d[d]]	vänta specificerat intervall under expect, avbryt sedan (time out)

Precis som med vissa uucp system avslutas sända strängar av *~r* om *~c* inte anges.

Endast de sista 7 tecknen i varje *expect* jämförs. En null expect, d v s *~0* eller två bindestreck, ger en kort fördröjning innan man fortsätter till nästa *send* sekvens. En null expect blir alltid uppfylld.

Precis som i *uucp* kommer scripten att misslyckas om inte strängen som angivits i *expect* kommer från remotesystemet. Om du tror att en sträng kan utebli kan du använda villkorliga sekvenser. Dessa uttrycks på följande sätt:

```
expect-send-expect[-send-expect[...]]
```

där sekvenser inledda med ett bindestreck följs om föregående expect misslyckas. Time out tiden för expect kan specificeras med *~w*. Ett *~w* utan argument väntar 15 sekunder.

Expect/send transaktioner kan felsökas genom att logga transaktionerna. Loggen sparar allt informationsutbyte, både förväntat och verkligt. Även utförandet av script kommandon kommer att loggas i transaktionsloggen om denna är aktiverad.

Observera att tecknet \ måste dubbleras i Kermit precis som i alla interaktiva kommandon. En rad kan avslutas med ett \ för att delas på flera rader.

Exempel 1:

Med ett modem ska en värddator med Unix anropas. En loginpromt, förväntas (som slutart med ..gin:) och om detta inte kommer, sänd en null sträng avslutad med *~r*. Vissa Unix-system kräver antingen EOT eller BREAK istället för null strängen, detta är beroende av det inloggningsprogram som används. Efter att användar-id och lösenord angivits ska prompten "?" besvaras med "x". Därefter förväntas prompten "$" från shell (sänd en return om den inte kommer). Därefter görs *cd* till bibliote-

ket **kermit** och programmet *wermit* körs. När wermit är startat, avslutas script-kommandot och man kopplas upp via connect-kommandot.

```
set modem-dialer hayes
set line /dev/tty05
set baud 1200
dial 9,5551212
script gin:--gin:--gin: smith ssword: mysecret -q \
x $--$ cd-skermit $ wermit
connect
```

Observera! Då kommandon kan köras i bakgrunden kan de även köras automatiskt vid vissa tider. En typisk shell-script (C-shell) som kan köras av *cron* kan se ut på följande sätt:

```
#
#keep trying to dial and log onto remote host and
#exchange files. Wait 10 minutes before retrying if
#dial or script fail.
#

cd someplace
while ( 1 )
      kermit < /tonight.cmd >> nightly.log &
      if ( ! $status ) break
sleep 600
end
```

Filen tonight.cmd kan ha två take-kommandon i sig, t ex en för att utföra kommandon för att ställa modem, linje, överföringshastighet och ringa upp samt en annan för att hämta/sända kommandon till remote servern. Den sista raden i tonight.cmd ska vara kommandot *bye* eller *quit.*

help

SYNTAX

```
help keyword
help {set, remote} keyword
```

Korta hjälpmeddelanden finns hela tiden tillgängliga i den interaktiva kommandonivån genom att skriva ett "?". En lite längre hjälp kan fås med kommandot *help*. Kommandot utan argument ger en kort sammanfattning av hur kommandon anges och hur man får ytterligare hjälp. *help* följt av något av Kermit kommandona på den högsta nivån, t ex *send*, ger information om kommandot. Kommandon som *set* och *remote* har ytterligare en hjälpnivå. *help* kan alltså användas på följande sätt: *help*, *help set* eller *help set parity*, där var och en ger ytterligare information.

exit, quit

Dessa två kommandon är identiska. Båda kan användas för följande:

- Få terminalen att återgå till normalinställning.
- Släppa kommunikationslinjer som har satts av *set line* fria.
- Stänga alla öppna loggfiler.
- Släppa fritt lås som satts av uucp &keller liknande på kommunikationslinjen.
- "lägga på luren" om kommunikationslinjen understödjer DTR-signalen.

När du lämnat Kermit kommer du att vara i samma aktuella bibliotek som när du startade programmet. Kommandot *exit* ges underförstått varje gång Kermit avslutas normalt, t ex efter det att direktkommandot Kermit utförts eller efter vissa avbrott.

7.5. UUCP Låsfiler

Unix har inget standardiserat sätt att få ensam åtkomst till en yttre kommunikationslinje. När du ger kommandot *set line* till Unix Kermit skulle du normalt få tillgång till denna kommunikationslinje även om någon annan process använder den. Den metod som används av de flesta Unixsystem för att hantera denna situation kallas Uucp låsfiler. *uucp*, Unix-to-Unix Copy, skapar en låsfil i sitt bibliotek, **/usr/spool/locks** med namn enligt **LCK..namn**, där *namn* är enhetsnamnet, t ex **tty07**.

Unix Kermit använder Uucp låsfiler för att undvika konflikter med *uucp*, *tip* och andra program som använder denna metod. När du försöker få tillgång till en yttre linje med kommandot *set line* eller med **-l** från kommandoraden, kommer Kermit att undersöka Uucp biblioteket för att se om det finns någon låsfil för denna enhet. Om du ger kommandot *set line /dev/tty06* kommer Kermit att söka efter filen

```
/usr/spool/locks/LCK..tty06
```

Om den hittar denna fil kommer ett felmeddelande och en biblioteksutskrift för filen att visas där du kan se vem som använder linjen, t ex

```
-r--r--r-- 1 fdc 4 May 7 13:02 /usr/spool/locks/LCK..tty06
```

I detta fall skulle du kontakta användaren **fdc** för att få veta när linjen blir ledig.

Denna konvention kräver att biblioteket **uucp** kan läsas och skrivas av alla användare. Om detta inte är möjligt kommer programmet att ge en varning om detta, men du får ändå fortsätta på egen risk.

Om Kermit inte hittar någon låsfil, kommer den att skapas. Därefter kan ingen annan användare använda linjen från ett program som använder metoden med Uucp låsfiler innan du har frigjort linjen. Om Kermit inte kunde skapa låsfilen får du ett varningsmeddelande, men får fortsätta på egen risk. När Kermit avslutas kommer låsfilen att tas bort.

Även om låsbiblioteket är läs- och skrivbart beror funktionen av låsmekanismen på om alla användare har samma namn för alla enheter. Om en enhet har mer än ett namn kan mekanismen sättas ur funktion genom att använda en synonym.

Om ett program som använder denna låsmetod skulle avbrytas på ett okontrollerat sätt, t ex dödat via ett shell kommando, kommer låsfilen att finnas kvar i uucp biblioteket. Detta markerar då helt felaktigt att linjen är låst. Om du äger låsfilen kan du själv ta bort den, annars kanske du måste gå till systemadministratören och be att få den bortplockad. En annan möjlighet är att programmet *uudemon.cleanup* regelbundet körs på systemet, som tar bort låsfiler efter en viss tid. Detta är vanligt på system med *uucp* och programmet körs då med jämna mellanrum via *crontab*.

Låsning behövs inte, och används inte heller, om kommunikationen sker via användarens login terminallinje.

Det kan se ut som linjelåsningen är fylld av faror. Den används i Unix Kermit enbart därför att många andra kommunikationsprogram under Unix använder sig av metoden. Det är önskvärt att ha ensamrätt till en kommunikationslinje, men samtidigt är det inte bra att en linje kan bli låst p g a att en gammal låsfil ligger kvar, eller att uucp biblioteket har felaktiga privilegier satta.

7.6. Kermit under System V Unix

Kermit kan avbrytas på kommandonivå eller under filöverföring genom att skriva CTRL-C. Programmet kommer då att utföra den normala exit funktionen och återställa terminalen. Om en protokolltransaktion höll på kommer ett felpaket att sändas till det andra Kermit programmet så att den kan avsluta på ett korrekt sätt.

Kermit kan startas i bakgrunden (& i en kommandorad i shell). Om en bakgrundsprocess "dödas" måste användaren manuellt ta bort alla låsfiler och modemet kan behöva återställas. Detta beror på att Kermit inte kan fånga signalen från systemanropet *kill ()*.

När ett systemkommando utförs, kan Kermit ofta fås att återgå till kommandonivån genom att skriva ett CTRL-C. Med System V ersätts den normala avbrottsfunktionen (oftast DEL tangenten) med CTRL-C. När Kermit upptäcker ett CTRL-C avbryter den på normalt sätt och tar bort låsfiler, återställer kommunikationslinjen, modem och terminal.

CTRL-C, CTRL-Z och CTRL-\ förlorar sin normala funktion vid direktkommunikation i *connect* kommandot och även under filöverföring när den styrande terminallinjen används för paket I/O.

Om du kör Kermit i "quiet"-mod som förgrundsprogram och avbryter programmet med ett avbrottskommando som CTRL-C, kommer terminalparametrarna inte att återställas korrekt. "quiet"-mod bör därför endast användas då Kermit körs i bakgrunden, eftersom programmet då ignorerar eventuella avbrottssignaler.

Om Kermit körs i bakgrunden kommer avbrottsignalerna (CTRL-C och System V **quit** signal) att ignoreras. Detta förhindrar att en avbrottsignal till ett förgrundsarbete stoppar en Kermit session i bakgrunden.

7.7. Begränsningar och kända fel i Kermit

Editerings tecken

Programmets interaktiva kommandoavbrott-, radera- och avbrottstecken är CTRL-C, Delete (eller Backspace) och CTRL-U. Det finns för närvarande ingen möjlighet att ändra på dessa.

Flödeskontroll

Kermit försöker att använda XON/XOFF under protokollanvändning, men samtidigt sätts kommunikationslinjen i "raw mod". På många system betyder "raw mod" att flödeskontrollen kopplas bort, så även om du ger kommandot *set flow xon/xoff* kommer ingen flödeskontroll att göras. Detta är systemberoende.

Höga överföringshastigheter

Det finns ingen möjlighet att specificera hastigheter högre än 9600 baud. Det flesta Unix system har inte ens symboler för detta, och även om de har det har programmet ingen möjlighet att kontrollera om en specifik linje understödjer dessa hastigheter.

Modemstyrning

Om en anslutning görs över en kommunikationslinje (istället för på den egna terminallinjen och linjen kräver modemsignaler t ex data terminal ready och carrier detect), kommer kommunikationen att avbrytas då användaren återgår till shell. I dessa fall bör de interaktiva kommandona användas och shellkommandon via pipes bör undvikas (se även *set modemdialer* och *dial* kommandona).

Login script

Login script implementationen följer idag konventionerna för Unix uucp:s **Systems** fil, istället för Kermits vanliga login script hantering.

Kommunikationslinjer för att ringa in resp. ut

Kermit kräver utgående linje för set line kommandot samt -l tillvalet. De flesta system har linjer uppsatta som ingående linjer, till vilka uppringande samtal kopplas samt några utgående linjer. Om en linje måste delas mellan in och utgående trafik finns det flera möjligheter. Dessa ligger helt utanför Kermit.

Kermit i lokala datanät (LAN)

Kermit kan användas med hastigheter upp till 9600 baud på lokala datanät under förutsättning att nätverksbuffertarna är tillräckligt stora att hanter Kermit paket.

När datorer är anslutna till Lokala Datanät via ett asynkront terminalgränssnitt ska anslutningen konfigureras med XON/XOFF flödeskontroll mellan dator och nätverksgränssnitt. Dessa signaler bör ej skickas transparent. Detta kan minska riskerna att Kermit fyller datanätets buffertar och förhindrar även att datanätet fyller Kermits buffertar.

Om inte nätets maskinvara kan hantera 100 tecken i taget och det inte går att använda flödeskontroll kan Kermits *set send/receive packet-length* användas för att korta av paketen.

Återställning av terminal efter avbrott eller kill

Om Kermit har stoppats på ett onormalt sätt kan det hända att terminalen inte återställs. (T ex. om Kermit stoppats med D-NIX-kommandot *kill*). Det kan då vara nödvändigt att göra detta manuellt med CTRL-J *stty sane* CTRL-J. Detta återställer normalt terminalen så att den blir användbar.

Remote host kommandon kan råka ut för time-out

När kommandot *remote host* används för att via en Kermit server starta Unix kommandon (som t ex *make*) kan dessa ta så lång tid att en time-out blir resultatet.

XOFF låsningar

När du återgår till Kermit efter en transaktion eller om du har avslutat serverfunktionen, kan det vara nödvändigt att skriva CTRL-Q för att ta bort eventuella XOFF låsningar.

8

8. D-NIX kommandon - sammandrag

8. D-NIX kommandon - sammandrag

Nedan är tillgängliga kommandon listade i alfabetisk ordning. I **Referenshandboken** ges detaljerade beskrivningar av de individuella kommandona i samma ordning. De flesta kommandona finns i filer som har samma namn. Några av dessa kommandon är interna kommandon i kommandoavkodaren shell, som också innehåller andra kommandon ej medtagna här. Se kommandobeskrivningen för kommandot *sh*.

accept	Tillåter LP-anrop för skrivare (LP).
badblk	Ersätter dåliga sektorer.
banner	Skapar huvudtitlar, försättstitlar (posters).
basename	Tar bort biblioteksnamn och/eller suffix ur pathname.
bootpar	Ändrar bootparametrar i NVRAM.
bup	Backup av valda filer.
cancel	Återkallar utskriftsbegäran på LP-skrivare.
cat	Visar filers innehåll.
cd	Byter till annat bibliotek.
chgrp	Ändrar gruppidentitet för filer.
chmod	Ändrar åtkomsttillstånd för filer.
chown	Ändrar ägare till filer.
cmp	Jämför två filer.
copy	Kopierar grupper av filer.
cp	Kopierar filer eller bibliotek.
cpio	Kopierar filarkiv, till och från.
cron	Tidsstyrning av processer.
crontab	Hanterar en användares crontab-fil.
cu	Ansluter terminal till externt system.
cut	Klipper ut selekterade fält på varje rad i en fil.
date	Skriver eller sätter datum och tid.
dd	Kopierar och omvandlar (konverterar) en fil.
devnm	Skriver ut namn på fysiska enheter.
df	Anger ledigt utrymme på skivminne.
dirname	Ger biblioteksdelen av ett pathname.
disable	Deaktiverar LP-skrivare (LP)
dmacs	D-MACS skärmorienterad texteditor.
DIAB1130	Se *machid*

DIAB 1320	Se *machid*
DIAB1420	Se *machid*
DIAB2420	Se *machid*
DS90-31	Se *machid*
DS90-41	Se *machid*
DS90-30S	Se *machid*
DS90-30	Se *machid*
DS90-11	Se *machid*
DS90-10E	Se *machid*
DS90-21	Se *machid*
du	Information om hur skivutrymmet används,uttryckt i kbyte.
echo	Ekar det som skrivs som argument.
ed	Texteditor. (radorienterad).
enable	Aktiverar LP-skrivare. (LP).
env	Skapar/ändrar omgivningen (environment).
errdemon	Loggar felmeddelanden.
eval	Avkodar och utför kommandon.
exec	Exekverar ett kommando istället för pågående shell.
exit	Avsluta shell, eller logga ut om exit ges från login shell.
export	Markerar shellvariabler för export.
expr	Evaluerar argument som uttryck.
false	Ger utgångsstatus skilt från noll.
fgrep	Söker rader med mönster i en fil.
find	Söker filer.
format	Formaterar disketter.
fsck	Kontroll och reparation av filsystem.
fscl	Testar om filsystemet på en disk är rent.
fsize	Skapa/ta bort checksumma i filsystem.
getopt	Utvärderar tillval i shellprocedurer.
getopts	Utvärderar tillval i shellprocedurer.
getty	Bestämmer terminaltyp, mod, hastighet och linjetyp samt startar en login-procedur.
grep	Söker rader med mönster i filer.

init	Initierar kontrollprocesser. (Ej användarkommando).
ipcrm	Tar bort meddelandekö, semafor, eller delat minnessegment.
ipcs	Rapporterar status för kommunikation mellan processer.
kermit	Filöverföring med kermit protokoll.
kill	Avslutar processer.
l	Listar informationen om filerna och innehållet i biblioteken på långt format.
labelit	Skapar/visar etikett för diskvolym.
lc	Listar fil/biblioteksinformation kolumnvis.
line	Läser från stdin till stdout.
ln	Skapar en länk mellan filerna.
load	Visar processorns genomsnittsbelastning.
login	Inloggning till systemet. (exec login).
logname	Ger användarens login-namn.
lp	Skickar utskrifter till skrivare. (LP)
lpadmin	Konfigurerar LP kösystem.
lpd	Hanterar utskrifter till skrivare från printerspoolern *lpr*. (Lpr)
lpmove	Flyttar utskrifter mellan olika destinationer. (LP)
lppg	Delar upp en utskrift i sidor.
lpr	Spooler för skrivare. (Lpr)
lpsched	Startar/stoppar köhanterare och flyttar en utskrift. (LP)
lpshut	Avbryter LP köhanterare lpsched. (LP)
lpstat	Skriver ut status-information för LP. (LP)
lpsubmit	Kontrollerar utskrift till skrivare.
ls	Listar information om filer och innehållet i bibliotek.
machid	Detta är inget kommando, men under denna rubrik beskrivs kommandon som visar vilken datortyp som programmet körs på. Kommandot returnerar **true** enbart på den datortyp kommandonamnet anger.
mail	Sänder meddelande till användare eller läser meddelanden.
mc68k	Se *machid*.

mc68020	Se *machid*.
mc68030	Se *machid*.
mesg	Accepterar/förbjuder kommandot *write* till egen terminalport.
mkcfig	Ändrar systemparametrar för D-NIX operativsystem-fil.
mkcont	Skapar fysiskt sammanhängande fil på skivminnet.
mkdir	Skapar nya bibliotek.
mkfs	Skapar ett D-NIX filsystem.
mknod	Skapar speciella filer eller pipes.
mknodm	Skapar specialfiler för spegel-skivminnen.
mksort	Skapar en sorteringsfil för kommandot sort.
mkuser	Lägger till användare.
mntchk	Kontrollerar en enhet och lägger till den (*mount*).
mount	Lägger till eller tar bort filsystem.
mv	Flyttar och /eller byter namn på filer och bibliotek.
mvdir	Flyttar ett bibliotek.
newgrp	Login till en ny grupp.
nice	Utför ett kommando med lägre prioritet.
nohup	Utför ett kommando med spärr mot hangup och avbrottssignaler.
ns32000	Se *machid*.
od	Skriver filer, oktal /hex/dec/ASCII.
passwd	Ändrar lösenord för *login*.
pdp11	Se *machid*.
powerfail	Total avstängning av systemet. (Ej användarkommando).
pr	Skriver ut filer formaterade.
print	Standard utskriftsprocedur, använder *pr*.
ps	Processtatus.
pwd	Anger vilket bibliotek som är aktuellt.
queue	Skriver en lista över de filer som finns i spoolerkön. (Lpr)
readonly	Förhindrar ändringar av shellvariabler.
red	Anropar begränsad *ed* (Text editor)

reject	Hindrar LP att ta emot utskrifter.
rinstall	Installerar nya D-NIX versioner.
rm	Tar bort filer, länkar, bibliotek.
rmail	Sänder meddelande till användare (begränsad).
rmdir	Tar bort tomma bibliotek.
rmuser	Tar bort en användare.
rsh	Anropar en begränsad shell kommandotolk.
scrfile	Tar bort filer och rensar motsvarande skivarean.
sed	Stream editor för editering av filer.
set	Listar eller ändrar shellvariabler och optioner.
setmnt	Uppdaterar filen **/etc/mnttab**. (Ej användarkommando)
setspeed	Ändrar temporärt terminalparametrar för seriella portar.
sh	Startar kommandotolken shell.
shutdown	Avslutar alla processer i systemet på ett korrekt sätt.
siv	Skärm-orienterad texteditor.
sleep	Gör en paus i bearbetningen.
sort	Sorterar och sammanlägger filer.
stty	Sätter/visar terminalparametrar för seriella portar.
su	Ändra användare eller bli super-user.
sync	Uppdaterar mountade filsystem.
tar	Sparar/återställer filer i bandarkiv och andra media.
tc	Övervakningsprogram för terminalkoncentrator
telinit	Ändra systemnivå i init-processen.
test	Provar villkor och returnerar utgångsstatus.
time	Exekveringstid för ett kommando skrivs ut.
times	Skriver ut användarens ackumulerade systemtid.
touch	Uppdaterar modifieringstid och användningstid på filer.
tr	Översätter tecken.

true	Ger utgångsstatus "sant" (noll).
tty	Ger terminalens namn.
type	Visar ett helt pathname för ett kommando.
u370	Se *machid*
u3b	Se *machid*
u3b10	Se *machid*
u3b2	Se *machid*
u3b5	Se *machid*
umask	Sätter *umask* för skapande av fil/bibliotek.
umount	Kopplar bort mountade filsystem.
uname	Skriver ut namnet på D-NIX systemet.
uniq	Rapporterar upprepade rader i en fil.
unset	Tar bort definitioner för shellvariabler.
vax	Se *machid*
wait	Väntar på att processer skall avslutas.
wall	Skriver ut meddelande till alla inloggade användare.
wc	Räknar ord, tecken och rader.
who	Vem är inloggad på systemet.
write	Skriver till annan användares terminal.