Linux – Friheden til at lære Unix

Version 1.8.20060212 – 2020-12-31

Peter Toft

og mange andre

Linux – Friheden til at lære UnixVersion 1.8.20060212 – 2020-12-31

af Peter Toft og og mange andre

I denne bog gennemgås Unix for begyndere, så brugeren kan begå sig på en Linuxmaskine. Desuden gennemgås netværksopsætning for en hjemmebruger.

Indholdsfortegnelse

Tabelliste

Forord

1. Lær Unix

I denne bog tager vi fat på unix-kundskaber. Der er ikke de store forudsætninger for at læse denne bog. Bogen er skabt ud fra "Linux – Friheden til at vælge" – version 5.0.

2. Linux-bøgerne

Bogen er en del af en serie, som kan findes på http://www.linuxbog.dk/

- *Linux – Friheden til at vælge installation* – Om at installere Linux.
- *Linux – Friheden til at lære Unix* – Om hvordan man bruger Linux' (og Unix') kommandolinjeværktøjer.
- *Linux – Friheden til at vælge grafisk brugergrænseflade* – Om alle de grafiske brugergrænseflader, der findes til Linux.
- *Linux – Friheden til at vælge programmer* – Om de programmer du kan få til Linux.
- *Linux – Friheden til systemadministration* – Om at administre sit eget linuxsystem.
- *Linux – Friheden til at programmere* – Programmering på Linux
- *Linux – Friheden til at programmere i C* – Om at programmere i sproget "C".
- *Linux – Friheden til at programmere i Java* – Om at programmere i sproget "Java".
- *Linux – Friheden til sikkerhed på internettet* – Om at sikre dit Linuxsystem mod indbrud fra internettet.
- *Linux – Friheden til egen webserver* – Om at sætte en webserver med databaser, CGI-programmer og andet godt op.
- *Linux – Friheden til at skrive dokumentation* – Om at skrive dokumentation (og andet) i SGML/DocBook, LaTeX eller andre formater.
- *Linux – Friheden til at vælge kontorprogrammer* – Kontorfunktioner på et Linux/KDE/OpenOffice.org-system.
- *Linux – Friheden til at vælge IT-løsning* – Om muligheder, fordele og ulemper ved at bruge Linux i sin IT-løsning.
- *Linux – Friheden til at vælge OpenOffice.org* – Om at bruge OpenOffice.org, både på Linux og på andre styresystemer.
- *Linux – Friheden til at vælge digital signatur* – Digital signatur på Linux.

3. Ophavsret

Denne bog er skrevet af Linux-brugere til Linux-brugere. Store dele af bogen er skrevet eller redigeret af enkelte forfattere, hvilket er nævnt i revisions-historien til bogen.

Bogen kan findes i opdateret form på http://www.linuxbog.dk/, mens prøve-udgaver kan findes på http://cvs.linuxbog.dk/.

Figur 1. ÅDL

Kommentarer, ris og ros og specielt fejl og mangler bedes sendt til linuxbog@sslug.dk (mailto:linuxbog@sslug.dk), men er du medlem af SSLUG kan du i stedet for med fordel skrive til sslug-bog@sslug.dk (mailto:sslug-bog@sslug.dk).

4. Om forfatterne og bogens historie

Bogen er redigeret af Peter Toft, mens den bygger på store dele af "Linux – Friheden til at vælge" fra 1998, skrevet af Peter Toft, Kenneth Geisshirt og Snebjørn Andersen. De sidste to har skrevet store dele af bogen.

5. Vi siger tak for hjælpen

Vi har haft stor glæde af mange SSLUG-medlemmers støtte, rettelser og forslag til forbedringer – bliv ved med dette. Specielt vil vi nævne:

- Jesper Norskov Kristensen
- Rasmus Ory Nielsen
- Anders Pedersen
- Frank Nørvig

- Mads Gige

Du kan i Appendiks C finde en liste over alle de revisioner, som bogen har været igennem.

Hvis du har ord du ikke forstår, så kan http://www.whatis.com være interessant. Her kan du slå mange computerord op dog kun på engelsk. I øvrigt kan bogens stikordsregister være interessant.

6. Typografi

Vi vil afslutte indledningen med at nævne den anvendte typografi.

- Navne på filer og kataloger vises som `foo.bar`
- Kommandoer, du udfører ved at skrive dem på en kommandolinje, vises som **help**
- Der er flere steder i bogen, hvor vi viser, hvad brugeren (som vi kalder "tyge") taster, og hvad Linux svarer. Det vil se ud som:

  ```
  [tyge@hven ~]$ Dette taster brugeren
  Dette svarer Linux.
  ```
- Der er tilsvarende flere steder i bogen, hvor vi viser, hvad systemadministratoren (brugeren "root") taster, og hvad Linux svarer. Det vil se ud som:

  ```
  hven# Dette taster systemadministratoren
  Dette svarer Linux.
  ```
  Det vigtige her er at kommandofortolkeren bruger nummertegnet (#) til at markere at man har systemadministratorrettigheder.

7. Eksemplerne

Hvis du mangler en kommando til et eksempel på dit system, så kan du i appendiks Appendiks B slå op, hvilken pakke kommandoen kommer i på forskellige systemer, så du kan bede din systemadministrator om at installere den.

Kapitel 1. Linux i teksttilstand

Linux er et unix-lignende styresystem. Unix' historie går tilbage til slutningen af 1960'erne, hvor en gruppe forskere ved AT&T's forskningslaboratorium eksperimenterede med computerens uanede muligheder. En af de grundlæggende ideer i Unix er at det skal være let at kombinere mange små programmer til større løsninger.

Dengang Unix var ungt var der ikke noget, der hed grafiske brugergrænseflader og mus. Brugeren havde udelukkende adgang til at styre computeren gennem en teksterminal. Med tiden udvikledes der meget effektive kommandofortolkere til både interaktiv brug og programmering af disse systemer. Denne udvikling fortsætter den dag idag (blandt andet ud fra en anerkendelse af at det er lettere at udtrykke sig præcist med ord end ved at pege, nikke og ryste på hovedet). Disse kommandofortolkere findes stort set alle i linux-udgaver, så du også kan styre dit linux-system med præcise og effektive kommandoer. Begynder du at bruge Linux seriøst, vil du forhåbentlig også finde ud af at kommandofortolkere er et effektivt redskab til at automatisere løsningen af rutineprægede opgaver.

Nu må du ikke tro, at det kun er af gammel vane, at unix-brugere taster deres kommandoer ind på en kommandolinje; faktisk er det muligt at udføre endda meget komplekse opgaver med meget lidt tastearbejde. Det skyldes, at der med Unix altid følger et hav af hjælpeprogrammer. Lad os give dig et eksempel – bare rolig, vi forventer ikke, at du allerede nu kan gennemskue, hvordan det fungerer, men vi vil bare vise dig, hvor lidt tastearbejde der skal til for at udføre store opgaver. Lad os antage, at du har en hjemmeside liggende på maskinen `www.hven.sslug.dk`. Du er en produktiv person med mange interesser, så din hjemmeside består af mange HTML-filer. En dag flytter du, og din hjemmeside skal skifte maskine – din nye maskine hedder `www.saltholm.sslug.dk`. Dit problem er, at du skal rette alle henvisninger på dine sider. I Unix (og dermed også Linux) kan det gøres ganske let ved at udføre kommandoen (du skal ikke gøre det):

```
[tyge@hven ~]$ find -iregex '.*html?' -print0 |\
xargs -0 sed -i 's/www\.hven\.sslug\.dk/www.saltholm.sslug.dk/gi'
```

Når du udfører kommandoen, bliver alle filer, som ender på "html" (bl.a. også "HTML" og "htm"), fundet (**find**). Du benytter et program ved navn **sed** til at foretage selve søg-og-erstat-proceduren. Dette eksempel er nok lidt for avanceret til vores bog her, men vi håber, at du nu kan se, hvor kraftfuld en enkel kommandolinje kan være i Unix.

1.1. Virtuelle konsoller

Når du vil styre Linux ved at skrive kommandoer i stedet for at være afhængig af en grafisk brugergrænseflade er det en mulighed at logge direkte ind med skærmen i ren teksttilstand. Hvis du sidder ved en linux-maskine der er i grafiktilstand, kan du typisk gå over til ren teksttilstand ved at taste Ctrl-Alt-F1. Det tastetryk bringer dig over til et helt andet skærmbillede, der er uafhængigt af det du så

før (et tryk på Ctrl-Alt-F7 eller Ctrl-Alt-F8 burde bringe dig tilbage igen). Over på ren-tekst-skærmbilledet burde der stå noget i stil med "login:" efterfulgt af en blinkende cursor (prøv ellers at taste Return eller Enter en enkelt gang for at få det frem). Her kan du ganske som ovre på det grafikbaserede skærmbillede logge ind på maskinen ved at indtaste dit brugernavn og din adgangskode, men når du logger ind på det tekstbaserede skærmbillede er det eneste du får en såkaldt "prompt" eller "kommandolinje". Afhængig af opsætningen af din konto vil den kunne se lidt forskellig ud, men dette er en mulighed:

```
[tyge@hven ~]$
```

Her kan du skrive kommandoer til systemet. I første omgang er den vigtigste kommando nok **logout**, som du bruger til at logge ud igen, så skærmbilledet er klart til en anden bruger (eller bare til en anden gang). Alternativt kan man i nogle tilfælde bruge kommandoen **exit** eller tastekombinationen Ctrl-D.

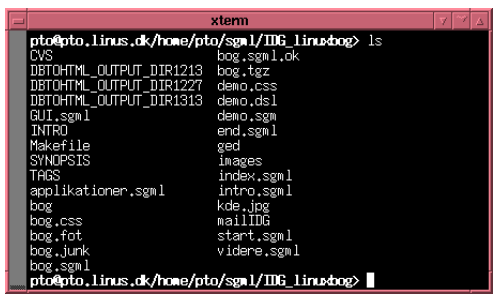
På et typisk linux-system vil Ctrl-Alt-F1, Ctrl-Alt-F2, og så videre indtil til Ctrl-Alt-F6 bringe dig til seks forskellig tekstbaserede skærmbilleder, mens Ctrl-Alt-F7 og nogle gange også Ctrl-Alt-F8 vil bringe dig til grafikbaserede skærmbilleder. Disse syv eller otte forskellige skærmbilleder kaldes systemets virtuelle konsoller. En fordel ved at have flere virtuelle konsoller er at forskellige brugere kan bruge den samme skærm uden at de behøver at lukke de programmer de har i gang når de bytter.

1.2. Terminalvinduer

Hvis du ikke er meget for kun at arbejde med tekstbaserede programmer (de er for eksempel ikke praktiske til at vise billeder) har du brug for at kunne få en kommandolinje frem inde i det grafikbaserede system (der hedder "X" eller "X Window System" på et almindeligt linux-system). Når du er logget ind gør du det ved at starte et terminalemuleringsprogram. Det mest udbredte er **xterm** men hvis du bruger KDE eller Gnome kan du med fordel benytte henholdsvis **konsole** eller **gnome-terminal** i stedet for, da de er lidt bedre integrerede i de to skrivebordssystemer.

Man får typisk startet et terminalemuleringsprogram ved at klikke på en ikon med en muslingeskal eller en sort skærm, eller ved at vælge et menupunkt der hedder noget i stil med "Konsole", "Terminal" eller "xterm".

Figur 1-1. Eksempel på xterm



Hvis du bruger **xterm** kan det være nyttigt at vide at man med et tryk på Ctrl-tasten og en af de tre museknapper får tre forskellige menuer til at justere opsætningen af terminalvinduet. Ctrl-venstre musetast vil man sjældent få brug for, men Ctrl-midterste musetast giver blandt andet mulighed for (de)aktivere vinduets rullebjælke og at nulstille terminalvinduet ("Reset") og Ctrl-højre musetast giver mulighed for at vælge skriftstørrelse.

Hvis du bruger **konsole** kan det være nyttigt at vide at du kan have flere forskellige aktive kommandofortolkere i samme vindue. Du kan starte en ekstra kommandofortolker med Ctrl-Alt-N, og du kan skifte mellem de kommandofortolkere der kører i vinduet med Shift-Pil-til-venstre og Shift-Pil-til-højre.

1.3. Kommandofortolkerne

En kommandofortolker er et program som brugeren kan bruge til at kommunikere med styresystemet. Den vil typisk kunne bruges til at starte andre programmer, til at styre hvordan data skal kanaliseres fra program til program og til at undersøge systemets tilstand.

Man kan se hvilke godkendte kommandofortolkere som findes på systemet i filen `/etc/shells`. Hvis man efterinstallerer en kommandofortolker på systemet, er det vigtigt at man husker at føje den fulde sti til den nye kommandofortolker til `/etc/shells`, da de brugere der vil bruge den kommandofortolker ellers kan få problemer.

På en SuSE-maskine kan `/etc/shells` se ud som følger – om de alle rent faktisk er installerede er ikke garanteret. Oftest følger alle med Linux-distributionerne.

```
[tyge@hven ~]$ cat /etc/shells
/bin/ash
/bin/bash
/bin/bash1
/bin/csh
/bin/false
/bin/ksh
/bin/sh
/bin/tcsh
/bin/true
/usr/bin/csh
/usr/bin/ksh
/usr/bin/passwd
/usr/bin/bash
/usr/bin/rbash
/usr/bin/tcsh
/usr/bin/zsh
```

I det følgende går vi nærmere ind i flere af de kommandofortolkere man typisk anvender. De mest anvendte kommandofortolkere er **bash** og **tcsh** (i den rækkefølge), mens det er sjældnere at folk bruger de ældre kommandofortolkere **csh** og **ksh**, eller den forholdsvis nye **zsh**.

Man kan afprøve en kommandofortolker ved simpelt hen at starte den som ethvert andet program:

```
[tyge@hven ~]$ zsh
hven%
```

Man kan (i princippet) altid stoppe en kommandofortolker med tastetrykket Ctrl-D, der signalerer at der ikke er flere data til programmet:

```
hven% <Ctrl-D>
[tyge@hven ~]$
```

Bemærk at de forskellige kommandofortolkeres kommandolinjer almindeligvis ser forskellige ud, så det er enkelt at se lige netop hvilken kommandofortolker man bruger. Men man kan selv justere kommandolinjens udseende, så den giver de oplysninger man har brug for.

Man vælger sin standard-kommandofortolker med programmet **chsh** (kort for "*ch*ange *sh*ell"):

```
[tyge@hven ~]$ chsh -s /bin/zsh
Changing login shell for tyge.
Password: din adgangskode skrives her
Shell changed.
```

(det kan godt ske at teksten er på dansk). Ændringen træder i kraft næste gang du logger ind på maskinen.

1.3.1. GNU Bourne Again Shell (bash)

Bash er den klart mest populære kommandofortolker i linux-verdenen. Det er ikke en tilfældighed, da den er forvalgt som kommandofortolker for nye brugere på de fleste linux-systemer, og da den er ret tæt på at have den syntaks som Unix-standarden specificerer for kommandofortolkere.

Ved login læses filen `~/.bash_profile` (eller `~/bash_login` eller `~/.profile`), og hver gang et terminalvindue startes vil filen `~/.bashrc` blive læst og udført. Endelig er det muligt at få filen `~/.bash_logout` udført når man stopper en kommandofortolker – her er det muligt at indlægge kommandoer som rydder op etc.

Bash udmærker sig dels ved at være kompatibel med den gamle unix-kommandofortolker, **sh**, men i høj grad på dens rigdom på både programmeringmuligheder og til en vis grad også på dens funktioner som interaktiv kommandofortolker.

Med pil-op/ned kan med genfinde de forrige kommandoer og man kan endda søge sig tilbage til en kommando man engang har udført i Bash ved at trykke **Ctrl-r**

STARTEN-AF-DEN-GAMLE-KOMMANDO. Med yderligere tryk på **Ctrl-r** vil man gense andre ældre kommandoer der også passer med starten af den kommando man har skrevet.

Med tabulator-tasten kan man få ekspanderet fil/katalog-navne svarende til den begyndelse af navnet man skriver. Man kan således med **cat /e<tabulator>/sh<tabulator>** få ekspanderet sig til **cat /etc/shells** uden at få skrevet ret meget.

Aliaser er nemme at sætte op i Bash. Opret filen `~/.alias` og indskriv aliaser efter skemaet `alias ALIAS-navn="kommando"`. Hvis man senere vil se hvilke aliaser man anvender kan **alias** vise dem alle.

```
alias ll="ls -al"
alias sa="ssh-add ~/.ssh/id_dsa"
```

Kommandoprompten kan sættes nøjagtig efter egen smag efter et stort udbud af muligheder. I denne bog har vi valgt at få vist brugernavn (\u), maskinnavn (\h) og nuværende katalog (\w), men man kan nemt vælge om som vist nedenfor. Udseende af prompten ændres direkte ved at sætte variablen `PS1`. Her ændrer vi til at vise brugernavn (\u) og tidspunktet (\T).

```
[tyge@hven ~]$ echo $PS1
[\u@\h:\w]
[tyge@hven ~]$ export PS1="\u (\T):"
tyge 10:01:12 :
```

Tip: Vil du have farver på din promt, så læs http://www-106.ibm.com/developerworks/linux/library/l-tip-prompt/.

1.3.1.1. Fiduser til mere effektiv brug af Bash

Du skal også lege lidt med tasterne pil-op og pil-ned, som løber igennem de gamle kommandoer igen. Ctrl-A og Ctrl-E bruges til at gå til starten og slutningen af en linje.

Kommando- og filudvidelsesfunktionerne er utroligt rare, hvis man vil slippe for at skrive lange program- og filnavne igen og igen. Når man har skrevet starten på et program- eller filnavn kan man ved at trykke på tabulatortasten få kommandofortolkeren til at gætte hvad man mener. Hvis der kun er én mulighed fylder kommandofortolkeren resten af navnet på. Er der flere muligheder, kan du trykke endnu en gang på tabulatortasten for at få vist alle mulighederne.

Hvis du skriver

```
[tyge@hven ~]$ tou<TAB> sikke_et_langt_filnavn
```

skulle det gerne blive til

```
[tyge@hven ~]$ touch sikke_et_langt_filnavn
```

idet der ikke er andre kommandoer, der begynder med 'tou'

```
[tyge@hven ~]$ rm sik<TAB>
```

Det fungerer i øvrigt ved, at kommandofortolkeren i første "ord" leder efter en kommando eller et program den kender gennem systemvariablen `PATH` eller i den sti, som angives foran selve kommandoen. Efter kommandoen ledes der efter filnavne i det angivne bibliotek.

1.3.2. Csh

En af de gamle kommandofortolkere som i praksis findes på alle unix-systemer er **csh** – "the Berkeley UNIX C shell". Tcsh skal ses som en naturlig videreudvikling af Csh, og er i praksis bagudkompatibel med Csh. Til alle praktiske formål er Tcsh at foretrække fremfor Csh, hvorfor vi ikke skal beskæftige os videre med Csh.

1.3.3. Tcsh

Tcsh er en af de meget populære kommandofortolkere i unix-verdenen, da den giver stort set alle de samme muligheder som bash (omend med en anden syntaks), er bagudkompatibel med den aldrende csh-kommandofortolker, og har en lidt mere raffineret kommando- og filnavnsekspansionsfunktion, som man som bruger selv kan indstille så det kun er de relevante filer der bliver foreslået. Et eksempel:

```
[tyge@hven ~]$ ls
linuxbog-unix.ps
linuxbog-unix-html.tar.gz
[tyge@hven ~]$ gv l<tab>
```

bliver til

```
linuxbog-unix.ps
[tyge@hven ~]$ gv linuxbog-unix.ps
```

hvis brugeren (eller systemadministratoren) har indstillet **tcsh** så den går ud fra at programmet **gv** kun bruger filer der ender på "`.ps`". Lige netop denne indstilling laves med linjen:

```
complete gv       'p/*/f:*.{ps,pdf,eps}/'
```

(filnavne der ender på "`.pdf`" og "`.eps`" kan også bruges). Som bruger føjer man den til filen "`~/.tcshrc`", mens man som systemadministrator kan føje den til "`/etc/csh.cshrc`", hvorved alle der bruger Tcsh vil få fornøjelse af det.

Selvom der er masser af andre filer, så vil **tcsh** filtrere filerne intelligent, da den ved at første argument efter **gv** kun kan være en Postscript-fil – evt. i et af underkatalogerne. Tcsh kan sættes op til at genkende alle programmers filtyper. I bogens eksempler på www.linuxbog.dk/unix/eksempler/shells/tcsh (http://www.linuxbog.dk/unix/eksempler/shells/tcsh) kan findes `dot.complete.tcsh`, som viser hvordan mange programmer er tilpasset.

I eksemplet er vist hvordan man ikke behøver at skrive hele filnavnet. Skulle der være flere filer som passer med starten af det man skriver, vil alle muligheder blive vist og man må fylde lidt flere bogstaver på før kommandofortolkeren automatisk kan gætte resten. Tricket med at trykke tabulator kan anvendes hele tiden.

Skulle man få brug for at udføre en af de forrige kommandoer igen, da trykker man blot på pil op (eller ned) for at gå igennem de forrige mange kommandoer. Det er i praksis en funktion man anvender meget. Hvis man skal til starten eller slutningen på den kommando man er ved at skrive bruger man henholdsvis Ctrl-A og Ctrl-E.

Hvis man vil lave genveje til kommandoer med aliaser kan man sætte dem ind i filen "`~/.tcshrc`", der bliver læst når Tcsh starter. Skemaet er `alias navn "kommando"`. Hvis man senere vil se hvilke aliaser man har defineret bruger man kommandoen **alias**.

Hvis **tcsh** startes som login-kommandofortolker vil den (udover nogle systemopsætningsfiler) læse filerne "`~/.tcshrc`", "`~/.history`", "`~/.login`" og "`~/.cshdirs`". Hvis **tcsh** ikke startes som login-kommandofortolker vil den (udover nogle systemopsætningsfiler) kun læse "`~/.tcshrc`". Det er altså i filen "`~/.tcshrc`" man som bruger kan vælge sin personlige opsætning af Tcsh.

Systemvariable sættes i Tcsh med **setenv VARIABELNAVN VÆRDI**. Bemærk at der bare er et mellemrum mellem variablens navn og den værdi den skal tildeles. Man kan se om en variabel er defineret ved at se på indholdet af `${?VARIABELNAVN}`:

```
[tyge@hven ~]$ echo ${?PRINTER}
0
```

1 betyder at variablen er defineret og 0 at den ikke er defineret.

Til sammenligning kan vi så prøve at tildele variablen `PRINTER` navnet på vores foretrukne printer:

```
[tyge@hven ~]$ setenv PRINTER lp1
[tyge@hven ~]$ echo ${PRINTER}
lp1
[tyge@hven ~]$ env | grep PRINTER
printer=lp1
```

Tcsh har i øvrigt et lille irritationsmoment i forbindelse med at der bliver installeret nye programmer. De instanser af Tcsh der blev startet før et program blev installeret, kan først finde det, når man har kørt

kommandoen **rehash** i dem.

På www.linuxbog.dk/unix/eksempler/shells/tcsh (http://www.linuxbog.dk/unix/eksempler/shells/tcsh) kan findes et forslag til følgende `~/.tcshrc`. På samme URL kan findes et forslag til `~/.login`.

Eksempel 1-1. Eksempel på en .tcshrc-fil

```
#!/bin/tcsh
# User .tcshrc file (/bin/tcsh initialization).
# Peter Toft 2002

# Omgåelse af en fejl i Red Hat 7.X
unset dspmbyte

# Anvend ssh til rsync
setenv RSYNC_RSH ssh

#Dansk tastaturopsætning og danske tekster i programmerne
setenv LC_ALL da_DK
setenv LANG da

# Led efter programmer i de følgende steder.
set path = ( /bin /usr/bin /usr/local/bin /usr/X11R6/bin )

if ( ! $?prompt ) exit  # Kommandofortolkeren er interaktiv

set history = 500       # 500 af de forrige kommandoer huskes
set savehist            # Number to save across sessions
set autolist            # List choices in name completion
set correct = cmd       # Checks spelling of commands

# Kommando-prompt med login-navn maskine og sti
set prompt  = "%{^[[1m%}${user}@`hostname`%/>%{^[[0m%} "

# Findes en ~/.alias-fil med aliaser, da køres denne
if ( -e ~/.alias ) source ~/.alias

# Findes en ~/.complete.tcsh-fil, da køres denne
if ( -e ~/.complete.tcsh ) source ~/.complete.tcsh
```

Flere forslag til at skrive en `~/.tcshrc` kan findes på http://tcshrc.sourceforge.net.

1.3.4. Korn Shell (ksh)

En af de halv-gamle kommandofortolkere med et ret avancerede scripting-muligheder er Korn Shell (ksh). Denne følger typisk med de kommercielle UNIX-varianter, men ikke med Linux. Der er udviklet en erstatning for **ksh** til Linux med navnet **pdksh** (Public Domain ksh). Denne anvendes dog ikke ret mange steder.

1.3.5. Zsh

Zsh er en af de mest avancerede kommandofortolkere der findes. Den ligner Bash og Ksh, men har – specielt ved interaktiv brug – mange fordele fremfor dem. Som med Tcsh kan man selv indstille filnavnsekspansionen. Desuden er den (som Bash) stort set kompatibel med Unix' standardsyntaks for kommandofortolkere. Det betyder at man kan nyde alle Zshs fordele som interaktiv kommandofortolker og bagefter kopiere de kommandoer man er kommet frem til direkte ind i et kommandofortolkerprogram, der i de fleste tilfælde vil kunne køre uændret på et vilkårligt unix-system.

```
tyge@hven:~% ls -al /etc/sh<TAB>
-rw-r--r--    1 root     root           185 sep 28  2000 /etc/shells
```

En rigtig smart ting som ingen af de andre kommandofortolkere har er et specielt jokertegn, `**`, der betyder underkataloger i en vilkårlig dybde:

```
tyge@hven:~% ls **/*.png
foldere/linux_på_dansk/friheden.png  foldere/sslug-folder/tux.png
images/tyge.png                      linuxbog/front.png
images/hanne.png                     linuxbog/sslug.png
```

Systemvariable sættes med **export variabelnavn=VÆRDI**, mens almindelige variable bare sættes med **variabelnavn=VÆRDI**:

```
tyge@hven:~% export printer=minlpr
tyge@hven:~% echo ${PRINTER}
minlpr
tyge@hven:~% huskeseddel=/tmp/husk
tyge@hven:~% echo ${huskeseddel}
/tmp/husk
```

Zsh bruger op til otte opsætningsfiler, hvoraf de fire ligger i ens eget hjemmekatalog, og de fire andre ligger i systemopsætningskataloget:

1. `/etc/zshenv` (læses altid)
2. `~/.zshenv` (læses altid)
3. `/etc/zprofile` (læses ved login)
4. `~/.zprofile` (læses ved login)
5. `/etc/zshrc` (læses ved login og ved interaktiv brug)
6. `~/.zshrc` (læses ved login og ved interaktiv brug)
7. `/etc/zlogin` (læses ved login)

8. `~/.zlogin` (læses ved login)

Første gang man starter Zsh, har man overhovedet ingen regler for fuldstændiggørelse af kommandoer og filnavne. Det kan man ændre på ved at køre den indbyggede kommando **compinstall** (det kan være nødvendigt først at køre kommandoen **autoload -U compinstall**). Du bliver så præsenteret for et primitivt menusystem, hvor du kan konfigurere fuldstændiggørelsen. I første omgang vil du nok kunne klare dig med standardindstillingerne.

Hvis de 500-600 fuldstændiggørelsesregler zsh kommer med som standard ikke er nok, kan man selvfølgelig lave flere. Hvis man f.eks. vil have zsh til kun at foreslå `.ogg`- og `.mp3`-filer når man har skrevet **music123**, kan man lægge en fil med indholdet:

```
#compdef music123

_files -g '*.(ogg|mp3)'
```

i `/usr/local/share/zsh/site-functions/`. Den første linje fortæller at denne funktion skal bruges når det er argumenter til **music123** der fuldstændiggøres, og den anden linje at kun filer der passer til det angivne mønster skal bruges.

Man kan naturligvis bladre tilbage gennem sine gamle kommandoer med pil op/ned, og ændre i en kommando inden man udfører den igen. Til dette formål kan man både få Emacs- og vi-tastebindinger, hvis en `EDITOR`- eller `VISUAL`-variabel er sat til en af tingene vælger Zsh automatisk de tilsvarende tastebindinger.

Der er eksempler på hvordan de fire brugerdefinerede filer kan se ud i bogens eksempler (www.linuxbog.dk/unix/eksempler/shells/zsh/ (http://www.linuxbog.dk/unix/eksempler/shells/zsh/)).

Zshs hjemmeside findes på SunSite.dk, hvor man også kan finde en brugsanvisning: http://zsh.sunsite.dk/Guide/.

1.3.6. Bourne Shell (sh)

En af de ældste kommandofortolkere til UNIX-familien er Bourne Shell (**sh**). Den har den fordel, at alle UNIX-maskiner har den installeret, og den dermed er egnet til at lave portabel kode. Som kommandofortolker er Bourne Shell imidlertid ret så skrabet og har slet ikke de smarte egenskaber, der findes i de nyere kommandofortolkere.

På Linux systemer er det ofte bash, som opfører sig sig som sh, når den bliver startet med det navn (fx. gennem et link). Det kalder man, at den kører i *posix* modus. I så fald aktiverer den ikke de "smarte features": completion, kommando-editor og history.

Ligesom **bash**, **zsh** og **ksh** sættes miljø-variable med export.

```
[tyge@hven ~]$ sh
sh-2.05$ dd=3
sh-2.05$ export dd
sh-2.05$ echo $dd
3
```

1.4. Læs videre om emnet

Der er udgivet masser af bøger om kommandofortolkere – i høj grad fra forlaget O'Reilly – se mere på http://www.oreilly.com. Bøgerne kan f.eks. købes fra Polyteknisk boghandel i Lyngby eller andre velassorterede boghandlere.

På internettet kan følgende steder være af interesse:

- http://www.faqs.org/faqs/unix-faq/shell/shell-differences/
- http://www.nscp.umd.edu/shells.html
- http://www.computerbits.com/archive/1997/1100/lnx9711.html

Kapitel 2. Brugsanvisninger

I Linux og Unix er der nutildags desværre lidt af et rod med flere forskellige formater af brugsanvisninger:

- **man**-formatet er det officielle system til brugsanvisninger på Unix.
- **info**-formatet blev på et tidspunkt lavet af GNU-projektet, da de ikke var tilfredse med mulighederne i **man**-formatet.
- HTML-formatet bliver også jævnligt brugt til brugsanvisninger.
- Postscript- og PDF-formaterne bliver typisk brugt til brugsanvisninger der skal se pæne ud på tryk.

Og endelig sker det også jævnligt at brugsanvisningen bare er en rå tekstfil. Med nogle programmer kan man være så heldig at man kan finde hele brugsanvisningen i alle de ovenstående formater[1].

Derudover har GNU-projektet en regel om at man ved at køre et program med kommandolinjetilvalget `-h` eller `--help`, kan få en kort vejledning i brugen af programmet.

Eksempel 2-1. Kommandolinjetilvalget `--help`

Vi kan for eksempel prøve at få en kort vejledning i hvordan man bruger programmet **man**:

```
[tyge@hven ~]$ man --help
man, version 1.5k

usage: man [-adfhktwW] [section] [-M path] [-P pager] [-S list]
        [-m system] [-p string] name ...

  a : find all matching entries
  c : do not use cat file
  d : print gobs of debugging information
[...]
```

Programmet skriver typisk ikke noget om hvad det er beregnet til, men giver blot en liste med de vigtigste kommandolinjetilvalg, som en påmindelse til den erfarne bruger der ikke lige kan huske om det hedder **man -p et-eller-andet** eller **man -P et-eller-andet**[2].

I de følgende afsnit vil vi gennemgå, hvordan man får adgang til at læse brugsanvisninger i de ovennævnte formater.

2.1. man-systemet

Hvis brugsanvisningen til et program, for eksempel **mutt**, findes i **man**-formatet (det er desværre aldrig til at vide før man prøver), så vil kommandoen:

```
[tyge@hven ~]$ man mutt
```

starte brugsanvisningsvisningsprogrammet **man** med brugsanvisningen til **mutt** åbnet. Man kommer ud af **man** ved at taste q. Mellemrumstasten bringer én en side længere ned i brugsanvisningen, linjeskift en linje længere ned i brugsanvisningen og b en side længere op i brugsanvisningen. Et tryk på h bringer den interne hjælp i programmet frem.

Af andre programmer der kan vise brugsanvisninger i **man**-formatet kan KDE's filhåndtering og browser, Konqueror, nævnes.

man-systemet er beregnet til at give brugeren en kort men brugbar (ofte er den desværre kun kort) introduktion til kommandoen. En brugsanvisning i **man**-systemet består typisk af:

- et afsnit med kommandoens navn og en beskrivelse på én linje,
- en samlet liste med alle kommandoens mulige tilvalg,
- en længere beskrivelse af hvad kommandoen bruges til,
- en detaljeret beskrivelse af alle kommandolinjetilvalgene,
- en beskrivelse af hvordan forskellige systemvariable påvirker programmet,
- henvisninger til andre relevante kommandoer og
- en liste med kendte fejl i programmet (hvorfor retter folk dem ikke bare?),

man bruger typisk programmet **less** til at vise brugsanvisningerne, så hvis du sætter dig ind i, hvordan man flytter rundt i en tekst, når du bruger **less**, så ved du samtidig også hvordan man gør det i **man**.

Brugsanvisningerne på et unix-system er kategoriseret i forskellige afsnit:

1. Om at bruge programmer.
2. Om at skrive programmer der snakker med styresystemet.
3. Om at skrive programmer.
4. Om specielle filer på systemet.
5. Om filformater.
6. Om spil.
7. Om konventioner og forskelligt andet.
8. Om systemadministrationskommandoer.

Hvis du bare skriver:

```
[tyge@hven ~]$ man info
```

så gennemgår **man** kategorierne i nummerorden indtil det finder en brugsanvisning om »info«. Det vil typisk være brugsanvisningen til programmet **info**, der findes i afsnit 1. Hvis du derimod har brug for beskrivelsen af *filformatet* »info«, så ved du at det er afsnit 5 der er interessant og kan i stedet for skrive:

```
[tyge@hven ~]$ man 5 info
```

Henvisninger til brugsanvisninger i **man**-systemet skrives i tekst typisk »navn(afsnit)«. Så hvis der står at man bør læse »info(5)« er det beskrivelsen af filformatet »info« (som man kan få frem med kommandoen **man 5 info**) der menes. Mens der med »info(1)« menes at man bør læse beskrivelsen af programmet »info« (som man kan få frem med kommandoen **man 1 info**).

Der findes en introduktion til hvert afsnit af brugsanvisningerne i **man**-systemet. Den hedder »intro«. Hvis du vil vide hvilke slags oplysninger du kan finde i afsnit 6 skriver du for eksempel:

```
[tyge@hven ~]$ man 6 intro
```

2.1.1. Opbygning

Unix-programmet **man** er meget praktisk at kende for at lære at bruge de mange parametre, der ofte findes til programmerne. Som eksempel kan du se dokumentationen for **man**-programmet ved at skrive **man man** på en kommandolinje.

```
NAME

man - format and display the on-line manual pages
manpath - determine user's search path for man pages

SYNOPSIS

man [-acdfhkKtwW] [-m system] [-p string] [-C config_file]
[-M path] [-P pager] [-S section_list] [section] name  ...

DESCRIPTION
man  formats  and displays the on-line manual pages.
```

Dette viser, at brugeren kan skrive **man PROGRAMNAVN** og få yderligere funktionalitet ved at tilføje ekstra flag, såsom **man -w PROGRAMNAVN**, som viser, hvilken fil der indeholder brugsanvisningen.

Tip: Med gvim (eller vi improved) kan du stille cursoren på et ord og trykke stort K for at se, om der er en man-page for pågældende ord.

"Synopsis" viser hvordan programmet bruges. Det, der vises i kantet parentes, er valgfrie parametre.

"Options"-afsnittet opremser alle de muligheder, programmet har. Du har allerede set, at **ls** har et tilvalg **-l**, men programmet har faktisk mange andre. Der findes generelt to typer muligheder: Den korte, startende med en bindestreg **-a**, og den lange med to bindestreger **--all**.

Det er ikke altid til at huske, hvad en kommando præcist hedder i Linux. Du sidder f.eks. og kan ikke huske, hvad en bestemt kommando hedder, men du kan huske, at den har noget med »web« at gøre. Til at lede alle **man**-siderne igennem har du kommandoen **apropos**:

```
[tyge@hven ~]$ apropos web
Galeon [galeon]      (1)  - gecko-based GNOME web browser
LWP                  (3pm)  - The World-Wide Web library for Perl
LWP::RobotUA         (3pm)  - A class for Web Robots
alevtd               (1)  - webserver for videotext pages
groff_www            (7)  - groff macros for authoring web pages
lynx                 (1)  - a general purpose distributed information browser for the World
pooltype             (1)  - display a WEB pool file
tangle               (1)  - translate WEB to Pascal
tie                  (1)  - merge or apply WEB change files
weave                (1)  - translate WEB to TeX
webcam               (1)  - capture images and upload them to a webserver using ftp
webcollage           (1)  - decorate the screen with random images from the web
```

Der kan være mange steder i **man**-systemet, hvor et ord indgår. Som du kan se, får du en kort beskrivelse med, som kan få dig til at huske, hvilket program du leder efter. Beskrivelsen er den, som står sammen med programmets navn øverst i brugsanvisningen.

Desværre leder **apropos** kun i **man**-systemet, så hvis et program kun har brugsanvisninger i **info**- eller HTML-format, så vil **apropos** ikke finde dem.

Det er også muligt at omdanne brugsanvisninger i **man**-systemet til HTML-filer med følgende kommando:

```
[tyge@hven ~]$ man2html /usr/man/man5/procmailrc.5 > procmailrc.html
[tyge@hven ~]$ man2html /sti/manpage.[1. .9] > /sti/manpage.html
```

2.2. info-systemet

Hvis brugsanvisningen til et program, for eksempel **mutt**, findes i **info**-formatet (det er desværre aldrig til at vide før man prøver), så vil kommandoen:

```
[tyge@hven ~]$ info mutt
```

starte brugsanvisningsvisningsprogrammet **info** med brugsanvisningen til **mutt** åbnet. Man kommer ud af **info** ved at taste q. Mellemrumstasten bringer én en side længere ned i brugsanvisningen og b en side længere op i brugsanvisningen. Et tryk på h bringer den interne hjælp i programmet frem.

Brugsanvisninger i **info**-formatet kan være opbygget som hypertekst med oplysningerne om programmet fordelt på forskellige sider.

Af andre programmer der kan vise brugsanvisninger i **info**-formatet kan KDE's filhåndtering og browser, Konqueror, nævnes.

info-systemet har også ofte et par forklarende ord til den, der slet ikke kender programmets virkemåde, men det er oftest lidt længere vejledninger.

Slutbemærkning:

1. Og nogle gange vil de forskellige formater indeholde forskellige oplysninger, eller de samme oplysninger i forskellig detaljeringsgrad.
2. Hvis man bruger Zsh, kan `--help` dog langt hen ad vejen erstattes med kreativ brug af tabulatortasten, da Zshs kommandofuldendelsessystem giver en del af de samme oplysninger.

Kapitel 3. Adgangsstyring

I Unix har man mulighed for at styre hvilke brugere der har adgang til at gøre bestemte ting og se bestemte data. Brugerne er opdelt i et hierarki med to niveauer. Øverst er der systemadministratoren (bruger nummer nul) der typisk, men ikke nødvendigvis, har brugernavnet "root". Nederst er der alle de andre brugere (brugernumre større end nul). Systemadministratoren har adgang til at gøre alt på systemet og kan dermed lave større ulykker end de almindelige brugere. Derfor er det vigtigt at man kun udfører programmer med systemadministratorrettigheder i det omfang det er strengt nødvendigt.

Udover brugere arbejder Unix' adgangsstyring også med grupper. Hver bruger er medlem af en eller flere grupper. Ligesom man for en enkelt bruger kan styre hvad han skal have adgang til, så kan man også gøre det for en hel gruppe ad gangen.

Unix styrer adgangen til ressourcer på filniveau. Det betyder at den mest detaljerede opdeling af rettigheder som Unix kan håndtere er enkelte dokumenter, kataloger eller eksterne enheder. Hvis man har brug for en mere detaljeret opdeling af adgangen til data, klarer man det typisk ved at køre et databasesystem ovenpå Unix.

Hver ressource tilhører i Unix én bestemt bruger og én bestemt gruppe. Man styrer adgangen til ressourcen ved at sætte eller stryge ni flag. De ni flag er delt ind i tre kategorier efter hvem de gælder for:

- brugeren som ressourcen tilhører (engelsk: "user")
- gruppen som ressourcen tilhører (engelsk: "group")
- andre (engelsk: "others")

og tre kategorier efter hvilken form for adgang de tillader:

- læse fra ressourcen (engelsk: "read")
- skrive i ressourcen (engelsk: "write")
- bruge ressourcen (engelsk: "execute")

Tabel 3-1 giver en oversigt over de ni almindelige flag, samt nogle specialflag man også kan sætte eller stryge. Det skal nævnes at flagene kan kombineres til at lave en meget fleksibel adgang til data.

Tabel 3-1. Flag til adgangsstyring

Talværdi	Symbol	Betydning
0400	u+r	Brugeren som ressourcen tilhører har adgang til at læse fra ressourcen.
0200	u+w	Brugeren som ressourcen tilhører har adgang til at skrive i ressourcen.

Talværdi	Symbol	Betydning
0100	u+x	Brugeren som ressourcen tilhører har adgang til at bruge ressourcen.
0040	g+r	Gruppen som ressourcen tilhører har adgang til at læse fra ressourcen.
0020	g+w	Gruppen som ressourcen tilhører har adgang til at skrive i ressourcen.
0010	g+x	Gruppen som ressourcen tilhører har adgang til at bruge ressourcen.
0004	o+r	Andre har adgang til at læse fra ressourcen.
0002	o+w	Andre har adgang til at skrive i ressourcen.
0001	o+x	Andre har adgang til at bruge ressourcen.
4000	u+s	Hvis ressourcen er et program, så vil det blive kørt med de rettigheder som brugeren det tilhører har.
2000	g+s	Hvis ressourcen er et program, så vil det blive kørt med de rettigheder som gruppen det tilhører har. Og hvis ressourcen er et katalog, så vil filer oprettet i det katalog automatisk tilhøre samme gruppe som kataloget tilhører.
1000	o+t	Hvis ressourcen er et program, så vil det blive kopieret til "swap", så det kan starte hurtigere.

Eksempel 3-1. Se adgangstilladelser for filer og kataloger

Kommandoen **ls** (vis katalogindhold) kan bruges til at vise hvilke adgangstilladelser der er sat for en fil. Vi kan for eksempel kigge i kataloget `~/websted/`:

```
[tyge@hven ~]$ ls -l ~/websted/
totalt 8
-rw-r--r--    1 tyge     brahe        2414 maj 18 19:51 index.html
drwxr-xr-x    3 tyge     brahe        4096 mar 30 16:09 Artikler
```

Der er en fil (`index.html`) markeret med et `-` som første tegn på linjen og et underkatalog (`Artikler`) markeret med et `d` som første tegn på linjen. Begge tilhører brugeren `tyge` og gruppen `brahe`. Brugeren `tyge` har tilladelse til både at læse og skrive i filen `index.html` mens brugere i gruppen `brahe` samt andre brugere på systemet kun har tilladelse til at læse i filen. Brugeren `tyge` har tilladelse til både at læse og skrive i kataloget `Artikler` mens brugere i gruppen `brahe` samt andre brugere på systemet kun har tilladelse til at læse i og bruge kataloget.

Eksempel 3-2. Hindre alle andre i at læse hjemmekataloget

Som en første praktisk øvelse i adgangsstyring kan man fratage alle andre alle adgangstilladelser under sit hjemmekatalog med kommandoen:

```
[tyge@hven ~]$ chmod -R go-rwx ~/
```

Kommandoen **chmod** bruges til at sætte og stryge adgangsstyringsflag. Tilvalget "-R" betyder at vi ønsker at det skal gøres rekursivt og inkludere alle filer og kataloger under det katalog vi udpeger. Tilvalget "go-rwx" betyder at vi vil fratage ("-") gruppen og andre ("go") adgangen til at læse, skrive og bruge ("rwx") kataloget (og p.g.a. tilvalget "-R" også de underliggende filer og kataloger). `~/`, hjemmekataloget, er det katalog hvis adgangsstyringsflag vi vil have **chmod** til at ændre.

Husk dog på at adgangsstyringen aldrig hindrer systemadministratoren — eller programmer der kører med systemadministratorrettigheder — i at læse, skrive og ændre i dine filer. For netværksdrev er dette dog ikke nødvendigvis sandt. Man kan fra en netværksserver hindre at lokale "root"-brugere får fuld adgang til data (eng: root-squashing).

Eksempel 3-3. Standardadgangstilladelser for nye filer

Med **umask** kan man styre de rettigheder som nye filer får. Man angiver et tre-cifret tal som anvendes efterfølgende. Skal det gemmes fra gang til gang man logger ind, så kan man tilføje det til sin kommandofortolkers opsætningsfil(er). Argumentet til **umask** er et tal (skrevet med grundtal 8) der definerer hvilke af de grundlæggende ni adgangstilladelser der *ikke* skal gives til nyoprettede filer. Tallet dannes ved at lægge talværdierne fra tabellen Tabel 3-1 for de adgangstilladelser der ikke skal gives sammen.

Hvis vi udelukkende vil begrænse adgangstilladelserne for *andre* brugere, skal vi således bruge talværdierne 0001 (ingen brugstilladelser til andre), 0002 (ingen skrivetilladelser til andre) og 0004 (ingen læsetilladelser til andre). Det er kun de sidste tre cifre der betyder noget, så kommandoen bliver således:

```
[tyge@hven ~]$ umask 007
```

Vi kan bruge kommandoen uden tilvalg til at se hvad indstillingerne er:

```
[tyge@hven ~]$ umask
007
```

Og for en god ordens skyld opretter vi en ny fil og ser hvilke adgangstilladelser den oprettes med:

```
[tyge@hven ~]$ touch Bond
```

```
[tyge@hven ~]$ ls -l Bond
-rw-rw----    1 tyge     brahe           0 maj 18 20:19 Bond
```

Til daglig er det nok mest hensigtsmæssigt at andre (også gruppen) ikke har automatisk har adgang til en brugers filer, hvilket svarer til tilvalget `077` til **umask**:

```
[tyge@hven ~]$ umask 077
```

Vi kan bruge kommandoen uden tilvalg til at se hvad indstillingerne er:

```
[tyge@hven ~]$ umask
077
```

Og for en god ordens skyld opretter vi igen en ny fil og ser hvilke adgangstilladelser den oprettes med:

```
[tyge@hven ~]$ touch James
[tyge@hven ~]$ ls -l James
-rw-------    1 tyge     brahe           0 maj 18 20:19 James
```

Eksempel 3-4. Gøre hjemmesiden synlig for Apache

Hvis Apache skal kunne vise din hjemmeside, er det nødvendigt at den bruger, hvis rettigheder Apache kører med, har adgang til at læse både kataloget hjemmesiden ligger i og selve filen med hjemmesiden. Men efter at vi i Eksempel 3-2 har hindret al adgang til filerne under `~/` for alle andre end dig selv, har Apache ikke længere adgang til det. Først giver vi alle adgang til at læse kataloget `~/websted/` og alt hvad der ligger i det og dets underkataloger:

```
[tyge@hven ~]$ chmod -R o+r ~/websted/
```

Men det alene er ikke nok, for for at komme til kataloget `~/websted/`, skal Apache også kunne komme til kataloget `~/` som vi stadig har frataget alle andre brugere alle adgangstilladelser til. For at kunne komme til en fil eller et underkatalog i et katalog skal man have adgang til at bruge kataloget, så vi sætter "x"-flaget for "andre brugere" ("o") på katalogerne `~/` og `~/websted/`:

```
[tyge@hven ~]$ chmod o+x ~/ ~/websted/
```

Hvis `~/websted/` har nogle underkataloger som Apache skal kunne læse filer fra, så bliver du også nødt til at udføre kommandoen **chmod o+x** på dem. Det kan for eksempel gøres sådan her:

```
[tyge@hven ~]$ find ~/websted/ -type d -print0 | xargs -0 chmod o+x
```

Hvis du har grund til at gøre nogle data på dit websted synlige for Apache, men ikke for alle brugere på maskinen, kan du bede din systemadministrator om at sørge for at kataloget med dit websted hører til den gruppe med hvis adgangstilladelser Apache kører med og så sætte "x"-flaget på det katalog (`~/websted/` ovenfor) for gruppen i stedet for for "andre brugere". Dette vil i praksis kun være relevant, hvis du har adgangskodebeskyttede oplysninger på webstedet, for ellers vil de andre brugere på maskinen jo under alle omstændigheder kunne komme til dataene gennem Apache.

Eksempel 3-5. Give en gruppe skrivetilladelse til et katalog

Hvis vi skal sidde og bringe orden i familiebillederne, kan det være praktisk, hvis resten af familien også kan komme med nye billeder til samlingen. Vi opretter derfor kataloget `~/familiebilleder/` som hele familien (gruppen "brahe") skal kunne læse og kataloget `~/familiebilleder/nye/`, hvor de også skal kunne tilføje billeder:

```
[tyge@hven ~]$ mkdir ~/familiebilleder/
[tyge@hven ~]$ mkdir ~/familiebilleder/nye/
[tyge@hven ~]$ chgrp -R brahe ~/familiebilleder/
[tyge@hven ~]$ chmod -R g+rx ~/familiebilleder/
[tyge@hven ~]$ chmod g+w ~/familiebilleder/nye/
```

Bemærk at der desværre ikke findes en mulighed for at give tilladelse til at tilføje data, der ikke også giver mulighed for at slette eller overskrive data. Det betyder for eksempel at medlemmerne af gruppen "brahe" nu både kan tilføje og slette billeder i kataloget `~/familiebilleder/nye/`.

Eksempel 3-6. Direkte styring af enheder

Hvis du har koblet et instrument på enheden `/dev/astrolabrium` som alle brugerne i gruppen "astronomi" skal kunne styre, bliver du som root nødt til at give dem adgang til at skrive til enheden (så de kan sende kommandoer til instrumentet) og til at læse fra enheden (så de kan modtage data fra instrumentet).

```
[tyge@hven ~]$ su -
Password:
[root@hven /root]# chgrp astronomi /dev/astrolabrium
[root@hven /root]# chmod g+rw /dev/astrolabrium
[root@hven /root]# exit
[tyge@hven ~]$ ls -l /dev/astrolabrium
brw-rw----    1 root     astronom   2,   0 apr 11  2002 /dev/astrolabrium
```

Da enheden er ejet af systemadministratoren, bliver vi først nødt til at skifte til systemadministratorkontoen med kommandoen **su -**. Bemærk også at kommandoen **ls -l** kun viser de første otte bogstaver i navnet på en bruger eller en gruppe.

Eksempel 3-7. suid

Normalt vil det være sådan, at når du starter et program op, f.eks. kommandoen **ls**, vil Linux køre dette program som den bruger, der startede programmet. Nogle gange kan det være nødvendigt at give en bruger flere rettigheder uden at skulle give brugeren systemadministratorrettigheder (på linje med `root`).

Suid (set user id) er et begreb (ikke et program), som giver mulighed for, at du kan udføre et program, som om du var en anden bruger. Det bruges normalt til at give almindelige brugere rettigheder til at udføre programmer, som om de var superbrugeren (`root`).

```
[tyge@hven MitKatalog]$ chmod +s FILNAVN
```

Når kommandoen **FILNAVN** udføres, vil Linux-kernen køre programmet med rettighederne for brugeren, der *ejer* filen, og ikke som brugeren, der starter programmet.

Dette kan f.eks. ses ved programmet **ping**, der skal være "suid root". Det skyldes, at det kun er root, der kan åbne den slags netværksforbindelse, som **ping** bruger.

```
[tyge@hven MitKatalog]$ ls /sbin/ping
-rwsr-xr-x   1 root     root        14804 Apr  7 23:21 /bin/ping
```

Suid er den største sikkerhedssynder på Unix-systemer. Det er f.eks. en dødssynd at sætte suid root for kommandofortolkerprogrammer, da det er muligt for en bruger at narre programmet til at efterlade en kommandofortolker med systemadministratorrettigheder. Jo færre suid filer du har på dit system jo bedre, men nogle ting er nødt til at være suid root for at fungere. Du kan se hvilke filer, der er suid root med følgende kommando:

```
[tyge@hven MitKatalog]$ find / -user root -perm +4000
```

Se mere i artiklen http://www.sslug.dk/artikler/Linux_sikkerhed/rootaccess.html om problemer med Suid.

Kapitel 4. Basal Unix

4.1. Hvad ligger hvor på en Linux-maskine

En af de ting der karakteriserer Unix (og måske i endnu højere grad Linux) er en idé om at alle ressourcer der er tilgængelige på systemet skal kunne betragtes på samme måde - som filer. Det gælder både egentlige filer på harddiske i computeren, selve harddisken (praktisk når den skal formatteres), systemets virtuelle konsoller, hukommelsen i computeren, tilsluttede apparater, etc. I Linux er selv opsætningsparametre til kernen og kernens status tilgængelige som filer. De eneste væsentlige ressourcer der ikke er tilgængelige som regulære filer i Linux er netværkskortene.

Når filer har så stor betydning i Unix, er det vigtigt at vide lidt om hvor de forskellige filer findes henne i systemet.

Linux har en meget velorganiseret måde at gemme forskellige filer. Den bygger primært på at man fordeler filer efter funktion og kun sekundært på hvilke programpakker de tilhører. Der er kataloger til delte biblioteker, nogle til programmer, og andre kataloger til brugerdata. Dette afsnit bygger på »Linux File-System Hierarchy Standard« (FHS) der er en standard som de forlag der udgiver linuxdistributioner har aftalt, for at det skal være lettere for programhuse og individuelle programmører at pakke én udgave at et program der vil virke på alle de forskellige linuxdistributioner (der overholder FHS).

Tabel 4-1. Oversigt over filtræet.

Filtræ	Forklaring
`/`	Toppen af katalogstrukturen.
`/bin`	Her er de mest nødvendige systemkommandoer gemt.
`/boot`	Dette katalog er reserveret til systemkernen, men nogle Linux-distributioner vælger at placere kernen i roden, dvs. `/`.
`/dev`	Indeholder alle device-filer, dvs. adgang til alle enheder såsom harddiske, cd-rom-drev, diskette, mus, tastatur og printer går igennem dette .
`/etc`	Dette katalog indeholder systemopsætningsfilerne, såvel som filerne, der er ansvarlige for systemets opstart, og grafiksystemets opsætning.
`/home`	Dette er kataloget for alle brugerkonti. Linux er et flerbrugerstyresystem. Vi kan anbefale, at man opretter en bruger til de mest normale jobs og *kun* bruger root til administration, da du som root ved en fejl kan komme til at slette vigtige filer, hvilket ville være umuligt som almindelig bruger.

Filtræ	Forklaring
`/lib`	Indeholder systemets delte filer. Linux sparer hukommelse ved at lægge kode, som bruges af mange programmer, ind i en fil, kaldet "shared library". På den måde vil der kun eksistere én kopi af filen i hukommelsen, når den bliver brugt.
`/opt`	Er reserveret til installation af valgfrie programpakker. SuSE placerer store programpakker som KDE og StarOffice i `/opt`, så hvis du installerer SuSE skal du derfor sørge for, at dette katalog har en del plads.
`/lost+found`	Hvis harddisken laver fejl, vil systemet selv forsøge at genskabe filerne, og hvis det ikke kan finde ud af, hvor i filtræet filen var placeret, vil den blive placeret i `/lost+found`.
`/mnt`	Hvis du vil have adgang til en cd-rom, et ZIP-drev eller en diskette, er det som regel i denne del af filsystemet, du får adgang til dem.
`/proc`	Dette katalog indeholder dynamisk opdaterede oplysninger om de kørende programmer og Linux-kernens status.
`/root`	Systemadministratorens hjemmekatalog.
`/sbin`	Her ligger de programmer der generelt kun er brugbare for systemadministratoren og er nødvendige for at systemet fungerer.
`/tmp`	Beregnet til midlertidige filer. De fleste programmer lægger automatisk midlertidige filer her. Som regel sletter systemet dem, når det starter op.
`/usr`	Her ligger delte data og programmer der ikke ændres og ikke er specifikke for lige netop den specifikke maskine. I princippet skal `/usr` efter at styresystemet er installeret monteres som et filsystem der kun kan læses fra og ikke skrives til.
`/usr/local`	Her kan systemadministratoren installere programmer og data der specifikke for maskinen. På systemer der overholder *Linux Standard Base* vil `/usr/local` ikke blive overskrevet ved systemopgraderinger.
`/var`	Indeholder datafiler hvis indhold ændres mens systemet er i drift. Det er for eksempel post- og printerkøer, logfiler og administrative data.

Der kan være yderligere dele af filtræet, såsom en Windows-disk som man ofte giver tilgang til via `/dos` eller `/dosc`.

Læs i øvrigt http://www.pathname.com/fhs/announce-2.0.html for mere information om filers placering.

4.2. Se indholdet af filer og kataloger

Indtil nu har du arbejdet med filer, men du har stadig ikke set på deres indhold. Indholdet af tekstfiler kan let vises. Kommandoen **cat** viser indholdet af en fil, f.eks. kan du se indholdet af `passwd` ved at skrive:

```
[tyge@hven ~]$ cat passwd
daemon:*:2:2:daemon:/sbin:
adm:*:3:4:adm:/var/adm:
lp:*:4:7:lp:/var/spool/lpd:
sync:*:5:0:sync:/sbin:/bin/sync
shutdown:*:6:0:shutdown:/sbin:/sbin/shutdown
halt:*:7:0:halt:/sbin:/sbin/halt
mail:*:8:12:mail:/var/spool/mail:
news:*:9:13:news:/var/spool/news:
uucp:*:10:14:uucp:/var/spool/uucp:
operator:*:11:0:operator:/root:
games:*:12:100:games:/usr/games:
gopher:*:13:30:gopher:/usr/lib/gopher-data:
ftp:*:14:50:FTP User:/home/ftp:
nobody:*:99:99:Nobody:/:
tyge:x:501:501:Tyge Brahe„„:/home/tyge:/bin/bash
```

cat er, som du ser, meget let at bruge, men kommandoen har en dårlig side: Hvis filens indhold fylder mere end en skærmside, kommer filen alt for hurtigt over skærmen, og bagefter ser du kun den sidste side (dvs. det antal linjer, som kan vises på din skærm). Du kan dog se nogle få sider tilbage ved at bruger Shift+PageUp. Kommandoerne **less** og **more** er derfor mere velegnede, da det er muligt at bladre frem og tilbage i filen. Kommandoen viser en side ad gangen, og du kan bladre frem ved at trykke på **f** (eng. "forward") eller mellemrum og tilbage ved at trykke på **b** (eng. "backward"), og i **less** kan pil op og ned også anvendes. Det er også muligt at søge ved at taste **/** efterfulgt af den tekst man leder efter og **return**. Søgningen gentages ved tryk på **n** (eng. "next"). Man afslutter både **less** og **more** ved at trykke **q** (eng. "quit").

4.3. Oprette, kopiere, flytte og slette filer og kataloger

Du har allerede set, hvordan du kan kopiere filer og se indholdet af dem. Nu kan det tænkes, at du finder ud af, at en fil skal have et andet navn, dvs. den skal omdøbes. Til det formål har du kommandoen **mv** (eng. "move"):

```
[tyge@hven ~]$ cp passwd nyFil
[tyge@hven ~]$ mv nyFil megetNyFil
[tyge@hven ~]$ ls -l
-rw-rw-r--   1 tyge    tyge         652 Jul 14 22:32 passwd
```

```
-rw-rw-r--   1 tyge    tyge         652 Jul 14 22:34 megetNyFil
```

Først tager vi med **cp** en kopi af filen `passwd` som vi kalder `nyFil`. Bagefter omdøber vi `nyFil` til `megetNyFil`. Som kommandoens navn antyder, kan den mere end bare omdøbe filer; den kan flytte dem til andre steder i filsystemet.

Her flytter vi for eksempel filen `megetNyFil` til kataloget `MitKatalog`:

```
[tyge@hven ~]$ mkdir MitKatalog
[tyge@hven ~]$ mv megetNyFil MitKatalog
[tyge@hven ~]$ cd MitKatalog
[tyge@hven MitKatalog]$ ls -l
-rw-rw-r--   1 tyge    tyge         652 Jul 14 22:34 megetNyFil
[tyge@hven MitKatalog]$ cd ..
[tyge@hven ~]$ ls -l
drwxrwxr-x   2 tyge    tyge        1024 Jul 14 22:34 MitKatalog
-rw-rw-r--   1 tyge    tyge         652 Jul 14 22:32 passwd
```

Nu kan det tænkes, at du vil slette filen `megetNyFil`. Kommandoen **rm** er lige det, du mangler **rm** (eng. "remove"):

```
[tyge@hven ~]$ cd MitKatalog
[tyge@hven MitKatalog]$ ls -l
-rw-rw-r--   1 tyge    tyge         652 Jul 14 22:34 megetNyFil
[tyge@hven MitKatalog]$ rm megetNyFil
[tyge@hven MitKatalog]$ ls -l
[tyge@hven MitKatalog]$
```

Du skal være meget forsigtig med at bruge **mv** og **rm**, idet der ikke er nogen mulighed for at fortryde. Hvis du vil blive spurgt, om det er rigtigt, at du vil flytte/slette, kan du bruge tilvalget `-i`, så det ovenstående eksempel bliver til:

```
[tyge@hven MitKatalog]$ rm -i megetNyFil
rm: remove almindelig fil 'megetNyFil'?
```

Her kan du så svare **j** for ja, og **n** for nej.

> **Kommandogenveje:** Det kan være praktisk at lave en genvej til en kommando med nogle bestemte tilvalg (et alias), så man ikke bliver nødt til at huske på tilvalgene hver gang man bruger kommandoen. Det er for eksempel en god idé at lave aliaser svarende til **rm -i** og **mv -i** som man lægger ind i stedet for **rm** og **mv**, så man altid bliver spurgt inden man kommer til at slette noget ved at skrive **rm en_fil** eller lignende.
>
> **alias rm='rm -i'** gør for eksempel at når du senere skriver **rm et-eller-andet**, så laver kommandofortolkeren det om til **rm -i et-eller-andet** før det bliver udført.
>
> Det kan også være rart at kunne køre **ls** med tilvalget `-l` let. Typisk laver man til det formål aliaset **ll**:

```
[tyge@hven MitKatalog]$ alias ll='ls -l'
[tyge@hven MitKatalog]$ ll
-rw-rw-r--   1 tyge    tyge         652 Jul 14 22:34 megetNyFil
[tyge@hven MitKatalog]$ ls
megetNyFil
```

Hvis du vil have Bash til at huske dine kommandogenveje fra gang til gang, bør du skrive dem ind i filen `~/.bashrc`:

```
alias rm='rm -i'
alias mv='mv -i'
alias ll='ls -l'
```

I visse tilfælde opstår der filer med i havelåger (#), mellemrum og andre specielle tegn i. Hvis man vil slette dem, kan man med fordel skrive navnene i citationstegn:

```
[tyge@hven MitKatalog]$ rm -i '#en sær fil#'
rm: remove almindelig fil '#en sær fil#'?
```

Du kan også oprette nye (tomme) filer med kommandoen **touch**, som tillige ændrer tidsstemplet på filen, hvis den allerede var der i forvejen:

```
[tyge@hven MitKatalog]$ touch myXYZoptioner.txt
[tyge@hven MitKatalog]$ ls -l
-rw-rw-r--   1 tyge    tyge           0 Jul 14 22:39 myXYZoptioner.txt
[tyge@hven MitKatalog]$
```

Hvis du så venter et par minutter og udfører **touch** igen, så vil du se, at tidsstemplet ændres:

```
[tyge@hven MitKatalog]$  touch myXYZoptioner.txt
[tyge@hven MitKatalog]$  ls -l
-rw-rw-r--   1 tyge    tyge           0 Jul 14 22:42 myXYZoptioner.txt
[tyge@hven MitKatalog]$
```

Endelig skal vi også se på hvordan man kan lave henvisninger (eng. "links"). Hvis du har brug for at en fil, for eksempel `myXYZoptioner.txt` også optræder med et andet navn, så kan du altid lave en symbolsk henvisning (eng. "symbolic link") til filen fra det andet navn:

```
[tyge@hven MitKatalog]$ ln -s myXYZoptioner.txt o.txt
```

Her satte vi `o.txt` til at være en symbolsk henvisning til `myXYZoptioner.txt`. Det betyder at hvis du nu giver et program besked om at gøre noget ved filen `o.txt`, så vil det i virkeligheden være filen `myXYZoptioner.txt` programmet arbejder med.

Med **ls** kan du se at `o.txt` er en symbolsk henvisning og ikke en rigtig fil:

```
[tyge@hven MitKatalog]$ ls -l o.txt
lrwxrwxrwx  1 tyge tyge  4 jul 15 20:35 o.txt -> myXYZoptioner.txt
```

Endelig kan det nævnes at man ud over symbolske henvisninger også kan laves direkte henvisninger (eng. "hard links"). Effekten er langt hen ad vejen den samme, men direkte henvisninger er begrænset til at fungere indenfor samme filsystem og er bundet til selve filerne og ikke til deres navne.

4.4. Jokertegn, omdirigering og kanaler

Vi vil i dette afsnit se på et par af systemets mere smarte funktioner, som gør livet lettere for dig som bruger.

4.4.1. Jokertegnene

Forestil dig, at du har mange filer, som du gerne vil slette. Du kan naturligvis skrive **rm den_ene_fil den_næste_fil den tredje_fil** (osv.) med hvert enkelt navn, men bare med 5-10 filer bliver du hurtigt træt af det. Redningen er jokertegnene `*` og `?`.

Lad os antage, at du har tre filer i et katalog, og at du gerne vil slette dem alle. For øvelsens skyld opretter vi først filerne med **touch**-kommandoen og sletter dem derefter. Nedenfor ser du hvordan:

```
[tyge@hven MitKatalog]$ touch aaa abc ccc
[tyge@hven MitKatalog]$ ls
 aaa  abc   ccc
[tyge@hven MitKatalog]$ rm *
[tyge@hven MitKatalog]$ ls
[tyge@hven MitKatalog]$
```

Asterisken (`*`) er et jokertegn som her betyder *alle filer*. Hvis du i stedet havde skrevet `a*`, ville det betyde alle filer som begynder med `a`.

```
[tyge@hven MitKatalog]$ touch aaa abc ccc
[tyge@hven MitKatalog]$ ls
 aaa  abc   ccc
[tyge@hven MitKatalog]$ rm a*
[tyge@hven MitKatalog]$ ls
 ccc
```

```
[tyge@hven NytKatalog]$
```

Asterisken (`*`) betyder helt generelt nul eller flere tegn i et filnavn. For eksempel betyder `a*b` alle filer, der begynder med `a` og slutter på `b`, hvilket vil inkludere filerne `ab`, `aDuErGodb`, men ikke en fil med navnet `abe`.

Ligesom asterisken fungerer spørgsmålstegn (`?`) også som jokertegn. Et spørgsmålstegn står for et (og netop et) vilkårligt tegn i et filnavn.

Eksempel 4-1. Kombination af flere jokertegn

Som et eksempel på hvordan jokertegnene kan kombineres med hinanden beder vi **ls** om at give os en liste med alle de filer i kataloget `/usr/bin`, hvis tredje bogstav er et z:

```
[tyge@hven MitKatalog]$ ls /usr/bin/??z*
/usr/bin/mozilla  /usr/bin/size86  /usr/bin/unzip
/usr/bin/size     /usr/bin/tgz     /usr/bin/unzipsfx
```

Da det er kommandofortolkeren og ikke de enkelte kommandoer der tager sig af at fortolke jokertegnene, kan man bruge præcist de samme mønstre til at angive samlinger af filer til alle programmer på systemet.

Et meget simpelt eksempel: Et katalog indeholder 3 filer: `fil1`, `fil2` og `ccc`. Skriver du: **ls f***, vil kommandofortolkeren først ekspandere f* og derefter kalde **ls** med: **ls fil1 fil2** . Output fra **ls f*** er blot:

```
[tyge@hven MitKatalog]$ touch fil1 fil2
[tyge@hven MitKatalog]$ ls
ccc   fil1   fil2
[tyge@hven MitKatalog]$ ls f*
fil1   fil2
```

og **ls** laver således et yderst banalt arbejde.

Men der er flere muligheder i dette. Et eksempel: For at se forskellen mellem `fil1` og `fil2` kan du skrive: **diff fil1 fil2**. Men du kan også nøjes med at skrive: **diff fil?** og så lade kommandofortolkeren regne ud at `f*` betyder `fil1 fil2`. Da de to filer er ens (de er begge tomme), så er der ingen forskel, og derfor intet at vise:

```
[tyge@hven MitKatalog]$ diff f*
[tyge@hven MitKatalog]$
```

Somme tider kan denne konsekvente ekspansion af jokertegn dog være en ulempe. For eksempel: Et katalog indeholder 2 filer:

```
[tyge@hven film]$ ls
dogme95.zip      film_index.html
```

Du kan også bruge zip-filer under Unix, og du kan se indholdet af zip-filen ved f.eks. at skrive:

```
[tyge@hven film]$ unzip -v d*
Archive:  dogme95.zip
Length Method Size Ratio Date    Time   CRC-32   Name
------ ------ ---- ----- ----    ----   ------   ----
 14853 Defl:N 5224 65%  10-27-98 16:12 944a4af4 festen.html
 14844 Defl:N 5401 64%  11-14-98 19:53 e55c1636 idioterne.html
  1941 Defl:N 1024 47%  03-12-99 22:12 5ecb7d23 mifune.html
------       ----- ---                          -----
 31638       11649 63%                          3 files
```

Nu vil vi pakke `festen.html` ud af zip-filen. Det er den eneste fil i zip-filen, der starter med `f`, så under DOS kunne man blot skrive: **unzip d* f***. Men prøver man det under Unix, går det galt:

```
[tyge@hven film]$ unzip d* f*
Archive:  dogme95.zip
caution: filename not matched:  film_index.html
```

Hvorfor det? Fordi der i kataloget i forvejen ligger en fil, der matcher `f*`, så vil kommandofortolkeren ekspandere *både* `d*` og `f*` og kalde **unzip** med: **unzip dogme95.zip film_index.html**.

Du bliver som minimum nødt til at skrive: **unzip d* fe***. Der er ingen fil i kataloget, der matcher `fe*`, så kommandofortolkeren vil kun ekspandere `d*` og kalde unzip med: **unzip dogme95.zip fe***. Det overlades nu til **unzip** at ekspandere `fe*`.

Alternativt kan man sætte en '\' foran stjernen og derved fortælle kommandofortolkeren at stjernen ikke skal ekspanderes. Altså: **unzip d* f***

Se også næste afsnit om regulære udtryk.

Det skal også nævnes, at hvis du kun vil finde et tegn, så brug et spørgsmålstegn. Vil du f.eks. finde alle filer aaa, baa, caa osv. men ikke caaa, så kan du bruge følgende

```
[tyge@hven MitKatalog]$ touch aaa baa caa caaa
[tyge@hven MitKatalog]$ ls
aaa   baa   caa   caaa   ccc   fil1   fil2
```

```
[tyge@hven MitKatalog]$ ls ?aa
aaa   baa   caa
```

"Globbing" og regulære udtryk: Jokertegnene `?` og `*` er en lille del af et større system til at danne mønstre for filnavne. Dette system kaldes "globbing". Du kan finde en detaljeret gennemgang af hvordan man bruger "globbing" i bøgerne »Unix in a Nutshell« og »Linux in a Nutshell«.

Det er vigtigt ikke at komme til at blande "globbing" sammen med det system man i Unix bruger til at danne mønstre for tekst i al almindelighed – regulære udtryk (eng. "regular expressions"). Regulære udtryk er et væsentligt stærkere system end "globbing", men de bruges altså sædvanligvis ikke til mønstre for filnavne.

4.4.2. "Globbing"

Jokertegnene `?` og `*` (omtalt i Afsnit 4.4.1) er de grundlæggende elementer i "globbing".

På kommandolinjen i for eksempel Bash kan man bruge noget der minder om regulære udtryk der kaldes "globbing". Med "globbing" kan du f.eks. liste alle dine filer, der slutter med bogstavet a ved at skrive **ls *[a]**. For at liste filer der *ikke* slutter på a skrives **ls *[^a]**. Det er vigtigt at skelne regex og globbing fra hinanden.

Tabel 4-2. Oversigt over "globbing"

Syntaks	Beskrivelse
*	Svarer til nul eller flere vilkårlige tegn.
?	Svarer til netop ét vilkårligt tegn
[a-d]	Svarer til et af tegnene `a`, `b`, `c` eller `d`. Men pas på. Nogle systemer kan ignorere forskelle mellem store og små bogstaver og regne med enten det danske eller det latinske alfabet.
[^a-d]	Svarer til alt andet end tegnene `a`, `b`, `c` eller `d`.
{Skåne,Sjælland,Hven}	Svarer til netop én af strengene `Skåne`, `Sjælland` og `Hven`.
*foo??ba[rz]{fubar,qux}*quux	Svarer til nul eller flere vilkårlige tegn, efterfulgt af strengen `foo`. Så to vilkårlige tegn efterfulgt af strengen `ba` og et af tegnene `r` eller `z`. Dernæst en af strengene `fubar` eller `qux` efterfulgt af ingen eller flere vilkårlige tegn og så strengen `quux` til sidst. Med dette udtryk ses det (let) at man kan finde filnavnene `foo++barquxquux` og `XYZfoo77bazfubarYESquux` i én og samme søgning.

4.4.3. Regulære udtryk

Mange programmer (blandt andre **ed**, **egrep**, Emacs, Perl, PHP, **sed**, **slocate** og **vi**) understøtter et meget effektivt system til søg-og-erstat af tekst efter angivne mønstre. Dette system kaldes *regulære udtryk* (eng. "regular expressions").

Tabel 4-3. Oversigt over regulære udtryk

Syntaks	Beskrivelse
.*	Svarer til nul eller flere vilkårlige tegn.
.	Svarer til netop ét vilkårligt tegn
[a-d]	Svarer til et af tegnene `a`, `b`, `c` eller `d`. Men pas på. Nogle systemer kan ignorere forskelle mellem store og små bogstaver og regne med enten det danske eller det latinske alfabet.
[^a-d]	Svarer til alt andet end tegnene `a`, `b`, `c` eller `d`.
(Skåne|Sjælland|Hven)	Svarer til netop én strengene `Skåne`, `Sjælland` og `Hven`.
.*foo..ba[rz](fubar|qux).*quux	Svarer til nul eller flere vilkårlige tegn, efterfulgt af strengen `foo`. Så to vilkårlige tegn efterfulgt af strengen `ba` og et af tegnene `r` eller `z`. Dernæst strengen en af strengene `fubar` eller `qux` efterfulgt af nul eller flere vilkårlige tegn og så strengen `quux` til sidst. Med dette udtryk ses det (let) at man kan finde filnavnene `foo++barquxquux` og `XYZfoo77bazfubarYESquux` i én og samme søgning.

Tabel 4-4. Definition af regulære udtryk

RegEx	Beskrivelse
[tegn]	Tegn der skal matches
RE1RE2	To forskellige regulære udtryk efter hinanden
RE*	Regulære udtryk skal passe nul eller flere gange
RE?	Regulære udtryk skal passe nul eller én gang. De regulære udtryk som følger POSIX-standarden benyttes dog "RE\?". Se f.eks. nedenstående eksempel med **slocate**.
RE+	Regulære udtryk skal passe én eller flere gange
RE1|RE2	Det ene eller det andet udtryk
(RE)	Gruppe som bl.a. kan benyttes som variabel. I **vi**, **emacs** og **sed** er det dog "\(RE\)".
^	Start af linje
$	Slutning af linje

For at gøre det nemmere at skrive og læse regulære udtryk, har man indført nogle forkortelser. Den "rigtige" syntax for at matche bogstavet 'a', er ved at skrive '[a]', men her kan man i stedet blot skrive 'a' (uden anførselstegn).

Tabel 4-5. Forkortelser af regulære udtryk

RegEx	Beskrivelse
a = a{1} = [a] = [a]{1}	Matcher tegnet 'a' én gang
[\0-\m\o-\377] = .	Matcher alle tegn undtagen \n
[*] = *	Matcher tegnet '*'
[abcd] = [a-d]	Matcher en række tegn
[\0-ac-\377] = [^b]	Undlader at matche et tegn 'b'
\d = [0-9]	Matcher tallene 0-9
[a][a][a] = aaa = a{3}	Matcher serien af tre ens tegn 'aaa'
[a][a][a]? = aaa? = a{2,3}	Matcher serien af to til tre ens tegn 'a'
a* = a{0,}	Matcher ingen eller mange tegn 'a'
a+ = aa* = a{1,}	Matcher et eller mange tegn 'a'
a? = a{0,1}	Matcher ingen eller et tegn 'a'

Et simpelt eksempel på brug af ovenstående, kunne være at matche en datoer i filen `regex`. Herunder ses en del af fil, hvor nogle linjer kun har en dato, og andre dato og en tekst. Datoen er skrevet i ISO-8601 formatet, som også kan udtrykkes som YYYY-MM-DD (ÅÅÅÅ-MM-DD).

```
# En kommentar 2002-12-01
2002-12-27
2002-12-28 Linus Torvalds 33
```

For at få listen over alle linjer der ikke indeholder en kommentar, skrives så:

```
[tyge@hven ~]$  egrep "^[^#]" regex
2002-12-27
2002-12-28 Linus Torvalds 33
```

Ovenstående giver nok ikke helt det ønskede resultat, i tilfælde af at der er skrevet noget i filen der er en ugyldig dato. Mere rigtigt vil det derfor være at matche datoerne. Da der kan forekomme datoer i kommentarer, så skal et match starte først på linjen, hvilket matches med '^'. Dernæst skal første tegn være et tal.

```
[tyge@hven ~]$  egrep "^[0-9]" regex
2002-12-27
2002-12-28 Linus Torvalds 33
```

Match af fire tal, kan enten skrives som:

```
[tyge@hven ~]$  egrep "^[0-9][0-9][0-9][0-9]" regex
```

eller lidt mere overskueligt:

```
[tyge@hven ~]$  egrep "^[0-9]{4}" regex
```

Et lidt mere præcist dato-match der både tjekker år, måned og dag bliver så:

```
[tyge@hven ~]$  egrep "^[0-9]{4}-[0-9]{2}-[0-9]{2}" regex
```

En mere løs form, hvor der kun matches tegn og bindestreger, der er nem at læse, kunne være:

```
[tyge@hven ~]$  egrep "^....-..-.." regex
```

Vil man kun have de linjer hvor der alene står en dato, indsættes et '$'-tegn der angiver slutning af linje.

```
[tyge@hven ~]$  egrep "^[0-9]{4}-[0-9]{2}-[0-9]{2}$" regex
2002-12-27
```

Det modsatte match hvor der kun vises linjer med dato og kommentar, kan så fås ved at angive at der skal være et eller flere tegn efter datoen. Det kan gøres ved at indsætte '.+'. Ikke alle programmer forstår '+', så man kan også skrive '..*' for at få det samme resultat.

```
[tyge@hven ~]$  egrep "^[0-9]{4}-[0-9]{2}-[0-9]{2}.+$" regex
2002-12-28 Linus Torvalds 33
```

I datoer anvendes ret få tal, da der for eksempel kun er 12 måneder, og maksimum 31 dage i en måned. Hvis man dertil også kun ønsker årstal imellem 2000 og 2099, kan dette skrives som:

```
[tyge@hven ~]$  egrep "^20[0-9]{2}-[01][0-9]-[0-3][0-9]" regex
2002-12-27
2002-12-28 Linus Torvalds 33
```

Da der kun er 12 måneder, kan dette matches lidt mere præcist ved at tage højde for at andet tal kun kan være 0-9 hvis første tal er et 0. Er første tal 1, kan andet tal kun være 0-2.

```
[tyge@hven ~]$  egrep "^20[0-9]{2}-(0[0-9]|1[0-2])-[0-3][0-9]" regex
2002-12-27
2002-12-28 Linus Torvalds 33
```

De ovenstående eksempler viser så dels hvordan man kan skrive forskelligt og få det samme output, og dels at man meget nemt kan komme til at matche upræcist. Netop med datoer er det meget komplekst at lave det fuldstændige match, der også tager højde for skudår, så mindre kan i mange tilfælde gøre det.

Et meget brugt regulært udtryk i tekst-programmer, er udtrykket der bytter om på to ord. Som eksempel kunne opgaven være at bytte "Linus Torvalds" om til "Torvalds, Linus" i hele dokumentet. Der skal så laves en søgning der dels matcher hvert ord, men også 'fanger' ordene og lægger dem ind i variable. Her bruges () til at 'fange' ordet og efterfølgende kan 'fangsterne' findes i nummererede variable 1, 2, 3 osv.

Eksempel 4-2. Notation af regulære udtryk i forskellige programmer

Perl:

```
s/(Linus) (Torvalds)/$2, $1/
```

Emacs:

```
M-x replace-regexp RET
\(Linus\) \(Torvalds\) RET
\2, \1 RET
```

vi:

```
:%s/\(Linus\) \(Torvalds\)/\2, \1/g
```

PHP:

```
$a = preg_replace("/(Linus) (Torvalds)/", "\\2, \\1", $a);
```

sed:

```
sed 's/\(Linus\) \(Torvalds\)/\2, \1/'
```

Der er altså en lille forskel i de forskellige programmer på brugen af regulære udtryk, men princippet er det samme.

Et lille praktisk og anvendeligt eksempel er at finde alle de filer på dine diske som enten ender på '.htm' eller '.html'. I regulære udtryk betyder '.' alle tegn undtaget \n så der skal en '\' foran. Med 'l\?' angives det, at der enten skal være et 'l' eller intet tegn. '$' angiver slutning på filnavnet (altså linjen). Med `slocate` gøres dette således:

```
[tyge@hven MitKatalog]$ slocate -r '\.html\?$'
```

Et eksempel på et regulært udtryk kunne være at finde alle brugernavne på systemet, som har brugergruppe-privilegier under 100. `/etc/passwd` indeholder oplysningerne.

```
..
halt:*:7:0:halt:/sbin:/sbin/halt
..
```

Brugernavn 'halt' tilhører gruppe '0' som angivet i fjerde kolonne. I Perl kunne kommandoen se således ud:

```
[tyge@hven MitKatalog]$  perl -ne '/^[^:]+:[^:]+:[^:]+:..?:/ and print' /etc/passwd
```

Idéen i ovenstående er at gå tre kolonner frem, og se om der står et eller to tegn imellem kolon. Hvis udtrykket matches skal det printes ud. '/' er skille-tegnet. Der skal matches fra første tegn på linjen '^'. '[^:]+:' betyder match et eller flere tegn som ikke er kolon, frem til næste kolon. Første match vil så være 'halt:'. Dette kolonne-match er gentaget tre gange. '..?:' matcher et eller to tegn, efterfulgt af kolon. Da der kun står tal i denne kolonne, er det kun '0'-'9' og '10'-'99' der vil blive listet.

Skal man matche det samme mange gange, måske flere end tre, kan man bruge gentagelsesoperatoren {n}, hvor n er et range. I førnævnte eksempel blev blokken '[^:]+:' gentaget tre gange. Da det er en blok skal der parenteser omkring før det virker: '([^:]+:){3}'. Den færdige linje ser nu således ud:

```
[tyge@hven MitKatalog]$  perl -ne '/^([^:]+:){3}..?:/ and print' /etc/passwd
```

'.' matcher ethvert tegn og vil man være sikker på at det kun er tal der bliver matchet, skal man skrive '[0-9]' i stedet for '.'. I Perl er '[0-9]' også forkortet til '\d' (eng: decimal number) så udtrykket kunne også se således ud:

```
[tyge@hven MitKatalog]$  perl -ne '/^([^:]+:){3}\d\d?:/ and print' /etc/passwd
```

Hjemmeopgaven er nu at forklare hvad '[^#]+#' og '[^;]+;' matcher. Ekstraopgaven er '[^:]?:' og '[^:]*:'.

Beskrivelse af regulære udtryk kan findes i manualsiderne med kommandoen **man 7 regex**.

Læs mere om regulære udtryk på http://zez.org/article/articleview/11/1/

4.4.4. Omdirigering

Det er ikke altid hensigtsmæssigt at få outputtet fra en kommando skrevet direkte på skærmen. Hvis du skriver **ls /usr/bin**, vil du forstå hvorfor. Derfor er det muligt at omdirigere output (eng. "redirect"). Til det formål bruger du tegnet **>** til at omdirigere outputtet til en fil.

```
[tyge@hven MitKatalog]$  ls /usr/bin > usrbin.dir
[tyge@hven MitKatalog]$  less usrbin.dir
```

Andre gange har man brug for at tilføje nye linjer til en fil, hvilket sker ved at benytte to større-end-tegn efter hinanden **>>**. Dette bruges især til logfiler.

```
[tyge@hven MitKatalog]$  date >> tidsstempel
[tyge@hven MitKatalog]$  date >> tidsstempel
[tyge@hven MitKatalog]$  cat tidsstempel
lør jul 29 17:03:29 CEST 2000
lør jul 29 17:03:31 CEST 2000
```

Tilsvarende kan nogle programmer og kommandoer fødes med indholdet af en fil ved at bruge **<**

```
[tyge@hven MitKatalog]$  cat < usrbin.dir
```

Hvilket dog i tilfældet med **cat** er det samme som **cat usrbin.dir**.

Du kan også sende skreven tekst ind til et program som forventer tastetryk. Nedenstående lille program svarer "y" på et spørgsmål som "Interaktivkommando", beder brugeren at svare på.

```
#!/bin/sh
Interaktivkommando <<EOF
y
EOF
```

4.4.5. Kanaler

Som du så i det foregående afsnit, kan du omdirigere uddata til en fil. Nu kan du med rette spørge hvorfor du skal omdirigere til en fil, hvis du kun skal se på den én gang? Svaret på dit spørgsmål er ligefremt: Det behøver du heller ikke. Du kan nemlig anvende en kanal. I Linux betyder det, at uddata fra et program bruges som inddata til et andet. Lad os give et lille eksempel:

```
[tyge@hven MitKatalog]$ ls /usr/bin | less
```

Den lodrette streg (|) sætter en kanal op, så uddata fra **ls /usr/bin** kanaliseres videre og bruges som inddata til **less**. Det er naturligvis muligt at sætte en hel række kanaler op efter hinanden og på den måde slippe for en masse midlertidige filer.

På amerikansk kaldes denne slags kanaler for »pipes«.

4.4.6. Programmet grep

Ofte vil du fra kommandoer som **cat**, **ls** og **find** få alt for meget information, og oftest ved du godt, at det kun er en begrænset del af de viste linjer tekst, du er interesseret i. Hvis du f.eks. kun skal bruge de linjer af `/etc/passwd`, der indeholder navnet "tyge", kan du bruge **grep** til at begrænse outputtet med.

```
[tyge@hven MitKatalog]$  cat /etc/passwd | grep tyge
tyge:x:501:501:Tyge Brahe„„:/home/tyge:/bin/bash
```

Du kan også bruge de forskellige i kombination. Eksemplet her finder filer i `/usr/bin` som begynder med p og indeholder ordet mail i filnavnet.

```
[tyge@hven MitKatalog]$  ls /usr/bin | grep ^p | grep mail
patch-metamail
```

```
pilot-mail
printmail
procmail
```

Hvilket også kunne have være gjort med kommandoen **ls /usr/bin | grep ^p.*mail**

4.5. Proces-kontrol

Når du starter et program op, bliver det til et kørende program. Et kørende program kaldes i Unix-sammenhænge for en proces (eng. "process"). Hver proces har en unik indikator, som er et heltal. Vi kalder denne indikator for PID, hvilket kommer af det engelske "Process IDentifier".

4.5.1. ps viser processer

Hvis du vil se hvilke programmer du har kørende, kan du bruge kommandoen **ps**. **ps** er en forkortelse for "process".

```
[tyge@hven MitKatalog]$  ps
  PID TTY STAT TIME COMMAND
  435  2   S   0:00 /bin/login -- tyge
  436  2   S   0:00 -bash
  447  2   R   0:00 ps
[tyge@hven MitKatalog]$
```

Ovenstående dialog viser, at der er et kørende program (**ps**) og to sovende (**/bin/login** og **bash**). At et program er kørende, ser du ved, at der under `STAT` står et "R" (for "running"), mens et sovende program i status-feltet har et "S" (for "sleeping"). En sovende proces er en proces, som er blevet startet, men ikke er aktiv, og nu står og venter på at blive aktiveret. Feltet, hvor der står `TTY`, viser, fra hvilken terminal programmet blev startet. I Linux kan du skifte mellem flere virtuelle konsoller, og hver virtuel konsol opfattes som en terminal.

Det skal nævnes at **ps** kun viser processer med samme tty (samme terminal) som **ps**, og du kan anvende **ps aux** til at se alle processer.

Vil du se *alt* hvad der kører på maskinen og se hvilke programmer, der kalder hinanden, da kan

```
[tyge@hven MitKatalog]$ ps auxfww
USER PID %CPU %MEM VSZ  RSS    TTY  STAT START TIME COMMAND
.... forkortet
tyge 2930 0.0  3.4 19544 6548  ?    S    21:05  0:00 kdeinit: Running...
tyge 2960 0.1  6.0 22760 11484 ?    S    21:05  0:15  \_ kdeinit: kwin
tyge 3086 0.1  6.9 24940 13192 ?    S    21:07  0:14  \_ kdeinit: konsole -icon konsole -mi
tyge 3088 0.0  0.8  2868 1628 pts/1 S    21:07  0:01  |   \_ -bin/tcsh
```

```
tyge 3116 1.2  4.6 11660 8972 pts/1 S    21:07  1:59  |   |   \_ emacs mandrake.sgml
tyge 4131 0.0  0.9  3224 1820 pts/1 S    23:33  0:00  |   |   \_ zsh
tyge 4142 0.0  0.7  2732 1516 pts/1 S    23:33  0:00  |   |       \_ -csh
tyge 4167 0.0  0.8  2848 1604 pts/1 S    23:33  0:00  |   |           \_ bash
tyge 4172 0.0  0.7  2728 1512 pts/1 S    23:33  0:00  |   |               \_ -sh
tyge 4197 0.0  0.7  2672 1408 pts/1 S    23:33  0:00  |   |                   \_ -csh
```

I eksemplet er vist en "syg" konstruktion, hvor sidste linje viser at man har startet **csh** op i **sh**, som igen er startet op i **bash**, som igen... Vil man vide hvilke af processerne der indeholder et søgeord – f.eks. ens eget login-navn, da kan man filtere med **ps auxfww | grep SØGEORD**.

4.5.2. Få et bedre overblik ved at bruge top

Som du så i foregående afsnit, kan **ps** bruges til at få et overblik over, hvilke processer du har kørende. Problemet er, at du kun får et statisk billede. Hvis du er interesseret i et mere dynamisk billede af din computers processer, kan du bruge programmet **top**. **top** opdaterer skærmen hvert femte sekund. Ved at trykke på "i" skifter du mellem "non-idle"-modus og almindelig modus. I "non-idle"-modus ser du kun de processer, som er aktive, mens du i almindelig modus ser alle. Du afslutter **top** ved at trykke på "q".

top leverer mange oplysninger, og derfor er det spændende at bruge programmet. Endvidere er det værd at læse programmets man-page.

4.5.3. At køre programmer i baggrunden

Alle de kommandoer, vi har præsenteret dig for i dette kapitel, har taget meget kort tid at udføre. Antag, at du ønsker at køre et program eller kommando, som tager en time at udføre. Hvis du nu startede programmet op på kommandolinjen, kunne du ikke udføre andre kommandoer i en time, da du ikke kunne komme til at taste nye kommandoer ind. Det er naturligvis ikke så smart, især ikke da Linux er et ægte multitasking styresystem!

Du kan løse dit ventetidsproblem ved at udføre programmet eller kommandoen i baggrunden. For at udføre et program i baggrunden sætter vi et **&** efter kommandoen. Et program der tager noget tid at køre, hvis du har mange filer, er **updatedb**. **updatedb** opretter den database der bruges af **locate**, og det tager typisk lang tid at køre denne kommando.

```
[root@hven root]$ updatedb &
[1] 7446
[root@hven root]$ ps
  PID TTY          TIME CMD
 7413 pts/2    00:00:00 su
 7416 pts/2    00:00:00 bash
 7446 pts/2    00:00:00 updatedb
 7447 pts/2    00:00:00 slocate
 7448 pts/2    00:00:00 ps
```

```
[root@hven root]$ cat /proc/loadavg
1.14 1.02 0.50 1/113 7479
[1]+  Done                    updatedb
[root@hven root]$
```

Programmet eller kommandoen kører nu samtidig med at du kan indtaste og udføre nye kommandoer. Grunden til vi siger, at programmet kører i baggrunden er, at du ikke sidder med det ved din konsol (som vi så kalder for forgrunden). Så mens **updatedb** kører i baggrunden, kan vi så kører kommandoen **ps** og se processen. Efter at have tastet en del andre kommandoer, her illustreret ved **cat /proc/loadavg**, er **updatedb** færdig med at køre, og kommer ud og skriver "Done". Her er det så baggrundsprocess nummer [1] der er færdig med at køre. **&**-kommandoen bruges på direkte fra kommandolinjen og inde i shell-scripts.

Glemte du **&** sidst på kommandolinjen, så kan du trykke Ctrl-z og så har du suspenderet jobbet (eng. "suspended"). Med **fg** (eng. "foreground") og **bg** (eng. "background") kan du styre hvad der er aktivt.

```
[root@hven root]$ updatedb
(tast Ctrl-z)
[1]+  Stopped                 updatedb
[root@hven root]$ bg
[1]+ updatedb &
[root@hven root]$
```

Har du fået sat et job igang i baggrunden, som du gerne vil stoppe, skal du først tilbage til programmet med **fg**. Tast dernæst Ctrl-c for at stoppe programmet.

```
[root@hven root]$ updatedb &
[1] 7446
[root@hven root]$ fg
updatedb
(tast Ctrl-c)
[root@hven root]$
```

Ctrl-z kan med fordel bruges mange steder. Det kunne være indefra dit post-program eller din tekst-editor. Situationen er at du sidder og er igang med at skrive et script, og så vil du lige prøve det. Efter afprøvning vil du tilbage samme sted i editoren, og stå det samme sted med markøren. Herunder er vist at editoren startes. Så tastes Ctrl-z og scriptet afprøves, hvilket giver en fejl. Til slut tilbage til editoren med kommandoen **fg**.

```
[tyge@hven ~]$  vi entest
(inde i editoren tastes Ctrl-z)
[tyge@hven ~]$  ./entest
entest: line 4: syntax error: unexpected end of file
[tyge@hven ~]$  fg
```

Det er muligt at have flere programmer kørende i baggrunden samtidigt. Det kan hurtigt blive lidt uoverskueligt at gøre det, men det vil i nogle tilfælde være en fordel. For overskueligehedens skyld anvendes i det følgende blot kommandoen **sleep**, der holder en pause på nogle sekunder for så at

returnerer til prompten. Ved at starte **sleep 100** og **sleep 200** skulle det være lidt nemmere at følge med i hvilket program man vender tilbage til. Herunder starter vi så de to kommandoer, og vender så tilbage til

```
[tyge@hven ~]$ sleep 100 &
[1] 7533
[tyge@hven ~]$ sleep 200 &
[2] 7534
[tyge@hven ~]$ fg
sleep 200
```

Ovenstående er helt som ønsket, hvis det altså er process 2 man vil tilbage til. Hvis man i stedet vil tilbage til process 1, skal kommandoen **%1** bruges, hvilket angiver baggrundsprocess nummer 1.

```
[tyge@hven ~]$  sleep 100 &
[1] 7533
[tyge@hven ~]$  sleep 200 &
[2] 7534
[tyge@hven ~]$  %1
sleep 100
```

Og selvfølgelig kan man starte flere programmer samtidigt på kommandolinjen og lægge dem alle i baggrunden, så de stadig kører samtidigt.

```
[tyge@hven ~]$  sleep 100 & sleep 200 &
[1] 7550
[2] 7551
[tyge@hven ~]$
```

Er du startet op i grafisk tilstand og gerne vil starte et grafisk program, men ikke lige vil flytte hånden over til musen og bruge menuerne, så er det:

```
[tyge@hven ~]$  gimp &
```

4.5.4. Kør flere programmer efter hinanden

Ofte har man brug for at køre flere programmer efter hinanden flere gange på kommandolinjen. Nu kan man jo blot taste pil op et par gange, og køre kommandoen igen. Som eksempel er det følgende to kommandoer vi vil kører gentagende gange.

```
[tyge@hven ~]$  date>foo
[tyge@hven ~]$  md5sum foo
7643ce159d2b9269e21cdd1fb88f79ba  foo
[tyge@hven ~]$
```

Ovenstående kunne løses ved at lave et script med disse to linjer, men det er lidt meget at gøre ud af det, når scriptet ikke senere skal bruges til noget. I stedet kan man sætte et **;** (semikolon) imellem, og derved først køre det ene program, efterfulgt af det andet.

```
[tyge@hven ~]$  date>foo ; md5sum foo
6d1a33063a8eba547c278a9776b7b59d  foo
[tyge@hven ~]$
```

I ovenstående eksempel sker der nok ikke nogen fejl – tror vi. Nu kunne man være så uheldig at filen `foo` var skrivebeskytte, hvilket gør at der ikke bliver skrevet nye data til filen. Dette beyder så at det ikke giver mening efterfølgende at køre **md5sum**. Først et eksempel på hvor galt det kan gå.

```
[tyge@hven ~]$  date>foo ; md5sum foo
bash: foo: Permission denied
6d1a33063a8eba547c278a9776b7b59d  foo
[tyge@hven ~]$  date>foo ; md5sum foo
bash: foo: Permission denied
6d1a33063a8eba547c278a9776b7b59d  foo
[tyge@hven ~]$
```

Problemet er at kommandoen **md5sum** bliver kørt, til trods for at det gik galt med den forgående kommando. Som vist bliver der hele tiden udregnet den samme md5-sum, hvilket ikke var meningen. Dette kan løses ved i stedet at bruge kommandoen **&&**. Derved bliver den næste kommando kun kørt, hvis den første gik godt.

```
[tyge@hven ~]$  date>foo && md5sum foo
bash: foo: Permission denied
[tyge@hven ~]$
```

Et mere praktisk eksempel på ovenstående vil være at kompilere en Linux-kerne. Først **make dep**, så **make**, så osv. osv. Men de efterfølgende kommandoer må kun køres hvis den forgående gik godt.

```
[tyge@hven ~]$  cd /usr/src/linux
[tyge@hven linux]$  make dep && make && make install
```

Eksemplet med **make** vil man både bruge på kommandolinjen og i shell-scripts. Et andet eksempel som typisk kun ville blive brugt i shell-scripts, kunne være først at teste om en fil eksiterer, og i herefter gøre noget med den.

```
#! /bin/sh
[ -e foo ] && md5sum foo
```

Mange syntes ikke om den måde at skrive det på, og det er nok også nemmere at læse det, hvis der i stedet står:

```
#! /bin/sh
if [ -e foo ]; then
 md5sum foo
fi
```

Kommandoen **[** er blot et symlink til **test**. Læs om den under **man test**.

4.5.5. Start en sub-shell på kommandolinjen

Når man starter en ny kommandofortolker, får man nye omgivelser (eng. environment) som programmet køres i. Dette ses når man har et kommandofortolkerprogram hvor systemvariable ændres eller tilføjes, eller der skiftes til et andet katalog. Når programmet er færdigt med at køre, er alt tilbage ved det gamle. Til tider kan det være nemmere ikke at skulle oprette et program først, men blot skrive det hele direkte på kommandolinjen. Først et lille simpelt eksempel som et program, der først skifter til kataloget `/tmp` og dernæst lister filerne. Tricket er, at når programmet er færdigt med at køre, står vi igen i samme katalog som vi kaldte programmet fra.

```
#! /bin/sh
cd /tmp ; ls
```

For at gøre det samme på kommandolinjen skal der blot paranteser omkring:

```
[tyge@hven ~]$ (cd /tmp ; ls)
```

Det eneste "ødelæggende" der skete ved ovenstående, var at der blev skiftet til et andet katalog, men det kunne også være en variabel der blev ændret:

```
[tyge@hven ~]$ (cd /tmp ; PATH=. ; min_test)
```

Et mere praktisk eksempel der er besværligt at løse uden brug af "()", kan illustreres ved noget hvor der skal holdes en pause. Som vist før kan man blot udskifte det som står imellem paranteserne med et script, hvilket i det følgende ikke er praktisk.

Vil man gerne vide lidt om hvordan en web-server er sat op, kan dette gøres med **telnet** og brug af HTTP-protokollen. Først åbnes en forbindelse til port 80. Når man får en forbindelse, indtastes HEAD-kommandoen og så venter man indtil der kommer et svar. Manuelt gøres det således:

```
[tyge@hven ~]$ telnet www.sslug.dk 80
Trying 130.228.2.150...
Connected to ns.sslug.dk (130.228.2.150).
Escape character is '^]'.
HEAD / HTTP/1.0

HTTP/1.1 200 OK
Date: Wed, 24 Jul 2002 07:21:16 GMT
Server: Apache/1.3.26 (Unix) (Red-Hat/Linux) OpenSSL/0.9.5a PHP/3.0.18
```

```
Last-Modified: Wed, 24 Jul 2002 04:45:00 GMT
Connection: close
Content-Type: text/html; charset=iso-8859-1

Connection closed by foreign host.
```

Bemærk at efter **HEAD** skal der tastes ENTER to gange. Her kunne man være fristet til at kanalisere HEAD-kommandoen direkte til **telnet**, men det går ikke da forbindelsen bliver lukket inden der kommer ouput retur fra web-serveren. En lille pause på et par sekunder med **sleep** løser problemet, men så skal kommandoen indkapsles med (). Det færdige eksempel er så:

```
[tyge@hven ~]$ (echo -e "HEAD / HTTP/1.0\n"; sleep 2) | telnet www.sslug.dk 80
```

Som før nævnt, kan det der står i parantes skrives i et script:

```
#! /bin/sh
echo -e "HEAD / HTTP/1.0\n"
sleep 2
```

4.5.6. Proces substituering

Man kommer til tider ud for at man bliver nødt til at oprette midlertidige filer, der senere skal bruges af et andet program. Nogle programmer kan kun tage filer som input, og hvis output kommer fra en proces, skal der gøres noget andet. Disse midlertidige filer skal til slut slettes, hvilket ofte ikke er noget problem. Med kommandoen **<()** kan man undgå dette midlertidige trin.

Programmer såsom **tar**, **diff** og **comm** skal bruge filer som input på kommandolinjen. Er input til disse programmer så noget der kommer fra en proces, vil man typisk gemme dette i en midlertidig fil. Her er et eksempel på hvordan det kunne se ud med et shell-script:

```
#! /bin/sh
# Find filer der skal tages backup af:
find | egrep "\.c$" > temp
# 'tar' filerne nævnt i 'temp'
tar cvf backup.tar -T temp
# Ryd op
rm -f temp
```

Ovenstående program er måske overskueligt, men man skal stadig huske at fjerne midlertidige filer, og man skal finde på et godt midlertidigt navn til sin `temp`-fil. I stedet for at oprette en midlertidig fil, kan man tage output fra **find** og lave dette om til noget der for **tar** ligner en fil. Med **<()** (processubstituering) bliver det så:

```
[tyge@hven ~]$ tar cvf backup.tar -T <(find | egrep "\.c$")
```

tar opfatter **<()** som en fil, og behandler den på denne måde. Lad os lige prøve at se om ikke systemet selv kan forklare hvad en **<()** er. Hvis nu **<()** er en fil, så ville man kunne skrive filnavnet ud på skærmen.

```
[tyge@hven ~]$ echo <(true)
/dev/fd/63
```

Her ses at der bare oprettes noget der ligner en midlertidig fil et eller andet sted, som **bash** selv holder styr på. Hvis man så har to gange **<()** på kommandolinjen, kan **bash** stadig håndtere det:

```
[tyge@hven ~]$ echo <(true) <(true)
/dev/fd/63 /dev/fd/62
```

<() ser ud som en fil, men er rent faktisk en *kanal*, hvilket er mere effektivt end at bruge en fil. Følgende kommandoer giver lidt yderligere information:

```
[tyge@hven ~]$ ls -l <(true)
lr-x------    1 tyge  tyge      64 Jul 25 20:15 /dev/fd/63 -> pipe:[100342]
[tyge@hven ~]$ stat <(true)
  File: "/dev/fd/63" -> "pipe:[100408]"
  Size: 64              Blocks: 0          IO Block: 0      Symbolic Link
Device: 3h/3d   Inode: 419070015   Links: 1
Access: (0500/lr-x------)  Uid: (  506/    tyge)   Gid: (  512/    tyge)
Access: Fri Jul 25 20:16:21 2002
Modify: Fri Jul 25 20:16:21 2002
Change: Fri Jul 25 20:16:21 2002
```

Ovenstående eksempel med **tar** kan selvfølgelig løses på mange andre måder, hvor **<()** ikke er nødvendigt. Til gengæld skulle det lille eksempel være nemmere at forstå. Et mere drilsk eksempel kan laves med **comm**, der er et program til at sammenligne filer. Ved almindelig brug anvendes det således:

```
[tyge@hven ~]$ comm fil-1 fil-2
```

Nu kan man være så uheldig at de filer der skal undersøges er uddata fra for eksempel **grep**. Uden brug af **<()** kunne et shell-script se således ud:

```
#! /bin/sh
grep foo test1 > fil-1
grep foo test2 > fil-2
comm fil-1 fil-2
rm -f fil-1 fil-2
```

Omskrevet med **<()** bliver det blot en enkelt linje:

```
[tyge@hven ~]$ comm <(grep foo test1) <(grep foo test2)
```

For mere information om *processubstituering* (eng. "process substitution") se http://www.tldp.org/LDP/abs/html/process-sub.html

4.5.7. Dræb en proces

Nu kan det ske, at du har fået startet et program, som du bliver træt af. Du vil altså gerne afbryde det, inden det er kørt færdigt. Unix-verdenen er barsk, for man taler ikke om at afbryde en proces, men om at slå den ihjel (eng. "kill"). Når du vil slå en proces ihjel, kan du bruge kommandoen **kill**. Som argument til **kill** giver du PID. Nedenfor er vist et eksempel.

```
[tyge@hven MitKatalog]$ ps
  PID TTY STAT TIME COMMAND
  435  2   S   0:00 /bin/login -- tyge
  436  2   S   0:00 -bash
  447  2   R   0:00 ps
  585  2   R   2:34 ls
[tyge@hven MitKatalog]$ kill 585
[tyge@hven MitKatalog]$ ps
  PID TTY STAT TIME COMMAND
  435  2   S   0:00 /bin/login -- tyge
  436  2   S   0:00 -bash
  763  2   R   0:00 ps
```

Det skal også nævnes, at enkelte gange kan en proces være kørt helt i skoven, og så må du tage kraftigere skyts i brug. I stedet for **kill PROCESNUMMER** kan du bruge **kill -9 PROCESNUMMER**. Forskellen er at **kill** som standard sender en besked til programmet om at det skal stoppe, mens man med **kill -9** beder linuxkernen om at tage sig af at stoppe programmet.

4.6. $variable, export og env

I Unix (og dermed Linux) bruger man ofte systemvariable (eng. "environment variables") til at gemme vigtig opsætningsinformation for de enkelte programmer eller for hele systemets virkemåde. Prøv f.eks. at skrive:

```
[tyge@hven ~]$ env
PWD=/home/tyge/linuxbog
VENDOR=intel
```

```
PAGER=less
.
.
.
```

Der kom mange linjer, som alle er formateret "VARIABEL=VÆRDI". Linjen "PWD" betyder f.eks. at jeg nu står i kataloget `/home/tyge/linuxbog`. De fleste er sat af systemet, mens brugeren kan vælge at tilføje nye eller overskrive de eksisterende. Typisk vil dette ske i `~/.profile` hvis brugeren anvender **bash**-shellen. Lad os se om det er tilfældet:

```
[tyge@hven ~]$ env | grep SHELL
SHELL=/bin/bash
```

En nem måde at se alle variable på er at skrive:

```
[tyge@hven ~]$ echo $<TAB><TAB>
$BASH             $HOME             $OPTERR           $SECONDS
$BASH_VERSINFO    $HOSTNAME         $OPTIND           $SHELL
$BASH_VERSION     $HOSTTYPE         $OSTYPE           $SHELLOPTS
$COLORS           $IFS              $PATH             $SHLVL
$COLUMNS          $LANG             $PIPESTATUS       $TERM
$DIRSTACK         $LESSOPEN         $PPID             $UID
$ENV              $LINENO           $PROMPT_COMMAND   $USER
$EUID             $LINES            $PS1              $USERNAME
$GROUPS           $LOGNAME          $PS2              $_
$HISTCMD          $LS_COLORS        $PS4              $ftp_proxy
$HISTFILE         $MACHTYPE         $PWD              $http_proxy
$HISTFILESIZE     $MAIL             $QTDIR            $langfile
$HISTSIZE         $MAILCHECK        $RANDOM           $sourced
```

Man kan også se **set** eller **printenv** for flere eksempler på bash-variable.

Skal man så bruge den variabel i en udprintning, f.eks. på skærmen, anvendes $VARIABEL til at tilgå værdien:

```
[tyge@hven ~]$ echo "Jeg bruger $SHELL"
Jeg bruger /bin/bash
```

Vil du sætte systemvariable, så er syntaksen **export VARIABEL=VÆRDI**.

Lad os se på et par af de meget anvendte variable. $HOME (eller $USER) Beskriver stien til en brugers hjemmekatalog. Ofte er denne sat til `/home/BRUGERNAVN`

`$PATH` er en liste med de kataloger kommandofortolkeren skal gennemsøge for at finde et program, hvis man ikke eksplicit angiver den fulde sti til det. Ofte har en normal bruger ikke "`/sbin`" i sin `$PATH` mens systemadministratoren har.

For at rette i $PATH se `/etc/profile` hvis ændringerne skal gælde alle brugere. Man kan også skrive "export PATH=$PATH:/home/mig/bin" Hvis man gerne vil have at ens shell også skal søge efter de programmer man har lagt i `/home/mig/bin`.

$DISPLAY angiver hvilken X-server man vil have at et program skal vises på. Når du starter X, binder skærmen sig til 0:0 normalt på Ctrl + Alt + F7, man kan starte en ny X-server på 0:1 (F8) med **startx -- :1** hvis man har lyst til det.

4.7. Rette i tekstfiler

Indtil videre har du set, hvordan du kan manipulere filer, men det er ofte meget nyttigt at kunne redigere i en tekstfil. Se Kapitel 5 for en omtale af nogle af de editorer der findes til Linux/Unix i tekstmodus.

4.8. Flere Unix-kommandoer

I dette appendiks vil vi gennemgå en række Unix/Linux-kommandoer. Gennemgangen er overfladisk, men du kan finde flere oplysninger i programmernes brugsanvisninger. Desuden er http://www.sslug.dk/artikler/begyndertips.html et godt sted at få mere information.

4.8.1. Mere om omdirigering

Linux (som Unix) arbejder med følgende input/output terminologi:

- Standard input (`stdin`). Normalt tastaturet.
- Standard output(`stdout`). Normalt skærmen.
- Standard error (`stderr`). Normalt skærmen.

Vi har før været inde på, at `stdout` kan omdirigeres med **>**. **cat fil1 fil2 > fil3** vil samle indholdet af filerne `fil1` og `fil2` i filen `fil3`.

Da måden, du omdirigerer på, er afhængig af valg af kommandofortolker, er det en god idé at undersøge dette nu. Prøv med:

```
[tyge@hven ~]$  env | grep ^SHELL | cut -d/ -f3
```

Svaret skulle gerne være **bash**, **zsh**, **csh** eller **tcsh**. Det kan være, at du anvender en anden skal – der findes mange. Hvis det er tilfældet, kan du se i brugsanvisningen for den aktuelle kommandofortolker (**man SKALLENS_NAVN**).

Hvis du selv starter en ny skal (enten fra kommandolinjen eller i et program), er det ikke sikkert, at ovenstående metode virker. Prøv derfor at skrive **ps**. En liste over de kørende processer vil da blive udskrevet til skærmen. Den nederste proces, der ender på "sh", angiver din skal – normalt.

Fælles for Bourne-Again Shell (**bash**) og C-Shell er følgende: Lad os antage, at filen `fil3` indeholder information, vi ønsker at bevare. Vi vil tilføje (append) indholdet af `fil1` og `fil2` til `fil3`. Det gøres ved:

```
[tyge@hven ~]$  cat fil1 fil2 >> fil3
```

Indholdet af en fil kan også anvendes som argument(er) til en kommando. F.eks. vil følgende kommando sende indholdet af filen `megenRos` til SSLUG's webmastere:

```
[tyge@hven ~]$  sendmail www_admin@sslug.dk < megenRos
```

Linux skelner (som Unix) mellem normale uddata (`stdout`) og fejluddata (`stderr`). Til tider kan det være rart kun at omdirigere det ene sæt meddelelser.

Specielt for **bash** gælder følgende: Omdirigering af `stdout` (1) og `stderr` (2) i **bash** er forholdsvis simpel.

Hvis kun fejlmeddelelserne fra en kommando – her **ls** - ønskes:

```
[tyge@hven ~]$   ls  1> /dev/null
```

Hvis kun `stdout` ønskes vist og fejlmeddelelserne skal sendes til en fil:

```
[tyge@hven ~]$  ls 2> fejlfil
```

Og endelig, hvis du ønsker at akkumulere fejlmeddelelser i en fil, kan **>>** anvendes – f.eks.

```
[tyge@hven ~]$  ls 2>> fejlfil
```

De to skaller **csh** og **tcsh** adskiller sig fra **bash** på følgende måder. Når du omdirigerer med **>** eller **>>**, er det kun `stdout`, der omdirigeres. Hvis `stderr` skal med, skal du anvende **&** efter omdirigeringen. `stderr` kan ikke omdirigeres alene, men med lidt krumspring lykkedes det alligevel:

```
[tyge@hven ~]$ ls –l /* | tee >& fil1 | diff fil1 – >fejlFil
```

Ovenstående kommando sender både `stderr` og `stdout` til filen `fil1`.

4.8.2. Hvem er logget ind?

Hvis du ønsker at vide, hvilke brugere der er logget på samme computer som dig, bruger du kommandoen **who**. Du vil så få en liste med brugere, der har logget ind, og fra hvilken (virtuel) terminal de er koblet til. Denne kommando har ikke den store værdi, med mindre du arbejder i et flerbruger-system. Arbejder du i et større netværk, kan det være, at **rwho** virker. **rwho** viser dig, hvem der er logget ind på hvilke computere i netværket. Dette kræver dog at både klient og server kører rwho-dæmonen.

```
[tyge@hven ~]$ who
tyge     tty1     Dec 21 17:25
tyge     pts/0    Dec 21 17:26
tyge     pts/1    Dec 21 17:26
tyge     pts/2    Dec 21 17:26
katja    pts/5    Dec 21 17:31 (k5.sslug)
```

Dette viser at brugeren tyge er logget ind via konsollen tty1, og har sidenhen (kl. 17:26) åbnet tre terminalvinduer – under X. Brugeren katja er også logget ind på maskinen kl. 17:31 fra maskinen "k5.sslug" og har kun et terminal-vindue (pts/5) åbent.

Man kan få mere information end blot antal terminaler ved at skrive

```
[tyge@hven ~]$ w
  5:35pm  up  7:20,  8 users,  load average: 0.14, 0.15, 0.10
USER  TTY      FROM     LOGIN@  IDLE   JCPU   PCPU  WHAT
tyge  tty1     -        5:25pm  9:40   1.26s  0.03s  /bin/sh /usr/X1
tyge  pts/0    -        5:26pm  7:03   2.50s  2.50s  /usr/bin/pine
tyge  pts/1    -        5:26pm  9:01   0.04s  0.04s  /bin/cat
tyge  pts/2    -        5:26pm  3:48   0.40s  0.40s  -bin/tcsh
katja pts/5    k5.sslug 5:31pm  3:13   0.37s  0.37s  -tcsh
tyge  pts/6    -        5:33pm  1.00s  0.25s  0.04s  w
```

Man får igen samme information om de to brugere som er logget ind, men der er mere information her. Man kan også se hvor lang tid de enkelte terminalvinduer har været urørte under "IDLE" og under "WHAT" kan man se om der kører et program i den enkelte terminal. I terminalen pts/0 kører der eksempelvis pine – dvs. et e-post program.

En anden vej at udvide information er at skrive

```
[tyge@hven ~]$ who am i
k6.sslug!tyge     pts/6    Dec 21 17:33
```

Resultatet er at man får maskin-navnet "k6.sslug" på den maskine man sidder ved og dernæst kommer brugernavnet. Det er meget nyttigt hvis man f.eks. anvender DHCP eller logger meget ind fra en maskine til mange andre over netværket.

Det næste vi ser på er finger, som giver lidt ekstra informationer. Man skriver "finger BRUGERNAVN" for at se information om BRUGERNAVN. Man får information tilbage om brugeren er logget ind på maskinen og man kan også se brugerens rigtige navn under "Name:"-feltet.

```
[tyge@hven ~]$ finger katja
Login: katja                            Name: Katja B
Directory: /home/katja                  Shell: /bin/tcsh
On since Fri Dec 21 17:31 (CET) on pts/5 from k5.sslug
   11 minutes 9 seconds idle
     (messages off)
No mail.
No Plan.
```

Hvis man vil have en oversigt over alle der er logget ind på systemet kan det klares ved blot at køre **finger** uden nogle kommandolinjetilvalg.

Finger læser filerne; `~/.plan` og `~/.project`. Er man systemadministrator (root) kan **last** og **lastlog** også give seneste login information.

Nu har vi set et par forskellige måder til at få information om hvilke brugere som anvender maskinen. En nem måde at få kontakt til en anden bruger på maskinen er at anvende "talk" (dette kræver at talk-dæmonen er installeret og startet). Men "talk BRUGERNAVN" kan man få en ICQ-lignende chat-session i gang hvor skærmen deles i to. Er brugeren logget ind i flere terminaler kan man anføre et ekstra argument til kommandoen "talk BRUGERNAVN TERMINAL", hvor terminal en af de terminaler, som BRUGERNAVN anvender – ovenfor kan brugeren "katja" findes på pty/5.

4.8.3. Søg og du skal finde

find bruges til at finde filer med. Syntaksen er: **find** hvorfra hvad [handling]. Ikke alle Linux/Unix-varianter kræver 3. argument. Lad os se nærmere på argumenterne.

- 1. argument, hvorfra:
  - `./` angiver aktuelt katalog.
  - `~/` angiver hjemmekatalog.
  - `/` angiver roden.
  - `/usr` angiver `usr` og alt under.
- 2. argument, hvad:
  - `-name foo`
    med navnet `foo`
  -

`-type d`
som er et katalog.

- 3. argument, [handling]:
  - `-print`
    udskriver, hvor filen er fundet
  - `-ls`
    udfører **ls -l** på søgeresultatet
  - `-exec cmd {}\;`
    udfører kommandoen **cmd** på søgeresultatet.
  - `-ok`
    som
    `exec`
    men spørger først.

Som et sødt lille eksempel på hvor smart **find** er, så kan vi tælle det totale antal af linjer i en række HTML-tekstfiler, der ligger spredt i nogle underbiblioteker.

```
[tyge@hven ~]$  (find . -name "*.html" -exec cat {} \;) | wc -l
```

4.8.4. Hvordan ændres datomærkningen?

Kommandoen **touch** anvendes til at oprette tomme filer eller til at ændre tidspunktet for sidste modifikation.

Lad os antage, at filen `minFil` eksisterer, og filen `minIkkeEksisterendeFil` ikke gør, da vil **touch minFil** sætte tiden for sidste modifikation af filen til det aktuelle klokkeslet. Kommandoen **touch minIkkeEksisterendeFil** vil oprette en tom fil med navnet `minIkkeEksisterendeFil`.

4.8.5. Hvilken filtype?

Kommandoen **file** forsøger at gætte, hvilken filtype der er givet som argument. Hvis **file** tror, at det er en ascii-fil, vil **file** læse de 512 første tegn og forsøge at gætte programmeringssproget.

file gætter desværre forkert fra tid til anden og kan f.eks. ikke genkende filer, der indeholder programmer, som er skrevet i Pascal eller Lisp.

4.8.6. Tid og dato

date udskriver den aktuelle dato og det aktuelle klokkeslet.

```
[tyge@hven ~]$  date
 lør jan 16 17:50:55 CET 1999
```

cal er en hel lille kalender. Uden argument udskrives kalenderen for den aktuelle måned. Med et argument (tal) regnes argumentet for et årstal. Vær i øvrigt opmærksom på, at **cal** er År 2000-klar, dvs. **cal 99** udskriver kalenderen for år 99, mens **cal 1999** skriver kalenderen for 1999.

Med 2 argumenter regnes det første som måned og det andet som årstal. Det er værd at bemærke, at **cal** antager, at skiftet fra den julianske kalender til den gregorianske kalender skete i september 1752, hvilket passer til engelske forhold (i katolske lande skete det i 1582 og i Danmark i år 1700).

```
[tyge@hven ~]$ cal 12 1991
   December 1991
Su Mo Tu We Th Fr Sa
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31
[tyge@hven ~]$ cal 9 1752
   September 1752
Su Mo Tu We Th Fr Sa
       1  2 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
```

4.8.7. Sortering

sort sorterer en fil linje for linje. **sort** kan også flette flere filer samtidig med at indholdet sorteres. Omdirigering af **sort** til en fil er mulig med tilvalget "-o". **sort fil1 fil2 > fil1** vil give et pudsigt resultat: Da `stdout` omdirigeres til `fil1`, som eksisterer i forvejen, slettes `fil1`, og en ny, tom `fil1` oprettes. Når `fil1` og `fil2` flettes og sorteres, vil `fil1` være tom. Derfor vil kun indholdet af `fil2` komme med i `fil1` - kryptisk – prøv selv! Husk blot, at **sort** er den eneste (standard) kommando, hvor `stdout` omdirigeres med tilvalget "-o".

Men **sort** kan mere: En fil kan være inddelt i poster, f.eks. filen `navne`.

```
[tyge@hven ~]$  cat navne
   poul nyrup 52
   holger beck-nielsen 90
   william gates 55
   torvalds linus ??
```

Lad os illustrere **sort** ved en række eksempler. **sort +1 navne** vil sortere på efternavn (første felt er felt 0). **sort +2-n navne** vil sortere numerisk på hattestørrelse. **sort -r navne** vil sortere i omvendt rækkefølge.

```
[tyge@hven ~]$  sort +2 -n -r navne
  torvalds linus 91.2
  holger beck-nielsen 90
  william gates 55
  poul nyrup 52
```

4.8.8. diff

diff udskriver forskellen mellem to filer. Lad os se på filerne `fil1` og `fil2`.

```
[tyge@hven ~]$  cat fil1
  Per
  Poul
  Bent
[tyge@hven ~]$  cat fil2
  Per
  Bjarne
[tyge@hven ~]$  diff fil1 fil2
  2,3c2
  < Poul   (oversat: ud går Poul)
  < Bent   (oversat: ud går Bent)
  > Bjarne (oversat: ind kom Bjarne)
```

Eksempel 4-3. diff mellem hele katalog-strukturer

Kommandoen **diff** er faktisk ret smart. Man kan endda få den til at finde alle rettelser lavet mellem to kataloger, hvor der laves rekursiv søgning. Har man f.eks. katalogerne `tux-ny` og `tux` og skal vide hvad der er lavet om fra `tux` til `tux-ny`, så skriver man

```
[tyge@hven ~]$  diff -ur --new-file tux tux-ny
diff -ur --new-file tux/README tux-ny/README
--- tux/README    Sun Jan  6 00:24:23 2002
+++ tux-ny/README Sun Jan  6 00:22:13 2002
@@ -1,5 +1,3 @@
-Dette katalog indeholder tekster svarende til
-bogen, hvis vi har ment dette er relevant.

 Seneste bøger kan findes på
 www.linuxbog.dk
```

Parameteren `u` giver en pænere diff-fil med en rimelig syntaks (unified diff). Parameteren `r` betyder rekursiv søgning ned i katalog-strukturen og endelig så er `--new-file` med for at sørge for at nye filer også kommer med.

4.8.9. Lappe på filer

Vi fortsætter ud fra Eksempel 4-3. Hvad der nu er smart er at en anden bruger, `tyge`, der har den gamle katalog-struktur `tux`, og denne skal opgraderes med ændringerne fra brugeren `tyge`. Først gemmer `tyge` ændringerne i en fil, `lap` og sender denne til `tyge`:

```
[tyge@hven ~]$ diff -ur --new-file tux tux-ny > lap
```

Brugeren `tyge` kan derefter tage filen og køre **patch** for at få opdateret underkataloget `tux`:

```
[tyge@hven ~]$ patch -p0 -u < lap
patching file tux/README
```

Nye filer skabes og slettede filer forsvinder automatisk. Det er på denne måde rettelser og tilføjelser (lapper) til Linux-kernen distribueres.

4.8.10. cat, uniq, wc og cmp

uniq fjerner ens linjer, der kommer efter hinanden, hvilket illustreres nedenfor.

```
[tyge@hven ~]$   cat per3
  Per
  Per
  Per
[tyge@hven ~]$  uniq per3
  Per
```

cmp sammenligner filer og stopper læsningen af filerne, når **cmp** finder en forskel.

wc står for Word Count, og som navnet antyder, tæller den ord i en fil. **wc** har nogle tilvalg: "-l", "-w" og "-c" for Lines, Words og Characters.

Vil du vide, hvor mange linjer en fil indeholder, skriver du **wc -l fil**. **wc** er særlig god sammen med andre kommandoer og **|** (kanaler), f.eks. vil nedenstående kommando tælle op, hvor mange filer, der er i kataloget `/usr/bin`

```
[tyge@hven ~]$  ls /usr/bin | wc –l
  1208
```

Eller hvis du vil vide, hvor mange kataloger der er i `/etc`:

```
[tyge@hven ~]$ find /etc –type d –maxdepth 1 | wc –l
  23
```

Forklaringen er som følger: **find /etc** finder alle filer og kataloger under `/etc`. Tilvalget `-type d` gør at kun kataloger bliver vist. Og tilvalget `-maxdepth 1` gør at søgningen bliver begrænset til `/etc` (ellers ville `/etc`s underkataloger og underkatalogers underkataloger også blive gennegået). Uddata fra **find** kommer som standard med et filnavn per linje. **wc -l** tæller antallet af linjer den modtager.

4.8.11. Hoved og hale af filer

tail – uden tilvalg – udskriver de 10 sidste linjer af en fil, "-5" vil udskrive de 5 sidste linjer, og "+8" vil udskrive fra og med linje 8 i en fil. Lad os se på et eksempel.

```
[tyge@hven ~]$  cat sang
    Jeg bærer med smil min byrde,
    jeg drager med sang mit læs;
    jeg er som den vilde hyrde,
    der genner sit kvæg på græs.
[tyge@hven ~]$  tail –2 sang
    jeg er som den vilde hyrde,
    der genner sit kvæg på græs.
```

tail har et meget nyttig "f"-tilvalg. Den får **tail** til løbende at vise de sidste 10 linjer af en fil, f.eks. vil du med **tail -f /var/log/messages** kunne følge med i, hvad alle system-dæmoner og lignende rapporterer.

head svarer til **tail**, men i stedet for slutningen af en fil, er det begyndelsen. **head** – uden tilvalg – udskriver de 10 første linjer af en fil. Tilvalg kan gives som for **tail**.

4.8.12. "cut" og "paste"

cut tager det, du specificerer i tilvalg, ud af en linje, dvs. -cx-y: tager fra tegn nr. x til tegn nr. y på hver linje i den specificerede fil. Er der kun ét tal, tages kun dette tegn ud. -fx-y er som for c, men her drejer det sig om felter, og -d'X' angiver felt-separatoren.

Vil du se, hvilke brugere der har adgang til systemet, så prøv: **cat /etc/passwd | cut -d':' -f1**

paste samler filer lodret, hvor **cat** samler (kan samle) filer vandret. Lad os se på følgende eksempel. Du ønsker nu at samle to filer, `navne` og `iq`, således at linje 1 fra `navne` efterfølges af linje 1 fra `iq` (uden at dette dog skulle være konkluderende, for såvidt angår de tilfældige sammenstillinger af for- og efternavne samt tal).

```
[tyge@hven ~]$  cat navne
      poul nyrup 52
      holger beck-nielsen 90
      william gates 55
      torvald linus ??
[tyge@hven ~]$  cat iq
      50
      230
      120
      ??
[tyge@hven ~]$  paste navne iq
      poul nyrup 52 50
      holger beck-nielsen 90 230
      william gates 55 120
      torvald linus ?? ??
```

4.8.13. Søg og du skal erstatte

tr erstatter det første argument med det andet. **tr** er en lidt speciel sag. Den forventer, at få input fra `stdin`. Derfor må du bruge følgende fremgangsmåde, hvis du vil erstatte noget i `fil1`, og skrive indholdet til `fil2`. Hvis du ønsker, at alle små tegn skal erstattes med store, skulle følgende kunne lade sig gøre:

```
[tyge@hven ~]$  tr '[a-z,æ,ø,å]' '[A-Z,Æ,Ø,Å]' < fil1 > fil2
```

4.8.14. xargs

En af de meget nyttige kommandoer er **xargs**, som kan bruges som kobling mellem en kommando der giver en række filnavne, hvor der skal udføres en speciel kommando på hver af de filer. Som eksempel kan vi prøve at få antal linier i alle filer der ender på `.sgml`.

```
[tyge@hven ~]$ ls  *.sgml
indhold.sgml
mandrake.sgml
unix.sgml
```

Hvis man prøver at kanalisere filnavnet ind i **wc**, så går det galt

```
[tyge@hven ~]$  ls *.sgml | wc --lines
3
```

Det der kommer som svar er at der er tre filer i **ls**-kommandoen. I stedet bruger vi **xargs**

```
[tyge@hven ~]$  ls  *.sgml | xargs -i= wc --lines =
 700 indhold.sgml
 232 mandrake.sgml
1232 unix.sgml
```

Programmet **xargs** får her filnavnene ind, og kører en kommando for *hver* af strengene. Argumentet `-i=` betyder at i den efterfølgende kommandoer skal lig-med (=) erstattes med en af de tekst-strenge (her er det et filnavn). Kommandoen er her **wc --lines =**, som viser antal linier i filnavnet =, dvs. det som kommer fra **ls**-kommandoen.

Der der i det nuværende katalog er tre filer, så er det nemt at vise hvad der egentlig bliver udført at ovenstående **xargs**-kommando:

```
wc --lines indhold.sgml
wc --lines mandrake.sgml
wc --lines unix.sgml
```

Det skal nævnes at anvendelsen af lig-med i eksemplet ikke er essentiel. Man kan også anvende et andet special-tegn.

4.8.15. Andre Unix-kommandoer

Tabel 4-6. Oversigt over de mest anvendte andre Unix-kommandoer.

Kommando	Forklaring
find	Find fil(er). Anvend f.eks. **find /usr -name "*.gif"** til at finde alle filer under biblioteket `/usr`, der ender på `.gif`. Prøv også **locate FILNAVN**.
whoami	Viser hvilket brugernavn-navn der arbejdes under.
who	Viser hvem der er logget ind på maskinen.
passwd	Skift adgangskode.
su	Skift bruger-identitet.
echo	Kommandoen **echo "TEKST"** skriver teksten på skærmen.
chown	Ændrer ejerskabet af filer.
date	Viser dato og tid.
lpr	Print ordre. Anvend **lpr -Pprinternavn filnavn** for at printe på printeren **printernavn**. De enkelte printere er defineret i filen `/etc/printcap`. De aktuelle printere kan være såvel lokale som netprintere.

Kommando	Forklaring
lpq	Printerkø forespørgsel. Anvend **lpq -Plp** til at vise, hvor langt printeren **lp** er med at printe ud.
lprm	Anvendes til at fjerne printjobs, som ikke er skrevet endnu. Se muligheder med **man lprm**.
tar	Anvendes til at pakke flere filer sammen til en. Anvend f.eks. **tar cvf tfil.tar fil1 fil2** for at pakke `fil1` og `fil2` sammen til filen `tfil.tar`. Tilsvarende kan filen pakkes ud med **tar xvf tfil.tar**.
compress	Pakker filer ind/ud. Anvend **compress filnavn** til at pakke filen til `filnavn.Z`. Tilsvarende anvendes **uncompress** til at pakke ud.
gzip	Andet og bedre pakkeprogram, der anvender `.gz` som slutning af filnavn. Tilsvarende findes **gunzip** til at pakke ud. Normalt ses også filtypen `.tgz`, som er en `tar` fil, hvor der efterfølgende er anvendt **gzip**.
bzip2	Andet og bedre pakkeprogram, der anvender `.bz2` som slutning af filnavn. Tilsvarende findes **bunzip2** til at pakke ud. Normalt ses også filtypen `.tar.bz2`, som er en `tar` fil, hvor der efterfølgende er anvendt **bzip2**.
diff	Sammenligner to filer og rapporterer forskellene.
free	Viser, hvor meget hukommelse der er brugt, og hvor meget der er til rådighed.
df	Viser, hvor meget diskplads der er brugt, og hvor meget der er til rådighed på samtlige diske.
du	Viser status over, hvor meget diskplads der er brugt under det sted, hvor du står i filtræet.
sort	Sorterer linjerne i en tekstfil.
ssh	**ssh saltholm.sslug.dk date** betyder, at du udfører kommandoen **date** på maskinen `saltholm.sslug.dk`, dvs. en anden Unix-maskine. Linux kan udføre kommandoer på andre maskiner og så vise grafik (og tekst) på din egen maskine.
telnet	Opretter en ukrypteret netværksforbindelse til en anden maskine. Nyttig til afprøvning og fejlsøgning af netværksforbindelser.
ping	Undersøger om der er forbindelse til en anden maskine. Nyttig til afprøvning og fejlsøgning af netværksforbindelser.

Kommando	Forklaring
traceroute	Viser hvilke maskiner forbindelsen til en anden maskine går over. Nyttig til afprøvning og fejlsøgning af netværksforbindelser.

4.9. Shell script

Skal man udføre mange kommandoer efter hinanden eller vil man have en anden bruger til at gøre det, kan det med fordel betale sig at lære at skrive shell scripts. I Afsnit 4.5.4 blev vist hvordan man afvikler flere kommandoer efter hinanden på kommando-linjen, men er det mere kompliceret eller der er 20 kommandoer, så vil et shell script med fordel kunne bruges.

I dette kapitel har vi medtaget nogle få almindelige kommandoer. En mere udførlig gennemgang findes på: http://www.tldp.org/HOWTO/Bash-Prog-Intro-HOWTO.html .

Eksempel 4-4. Simpelt kommandofortolkerprogram

Alle kommandoer man kan udføre på kommandolinjen, kan også udføres i et kommandofortolkerprogram (eng. »shell script«). Et kommandofortolkerprogram er blot en tekstfil der starter med at angive hvilken kommandofortolker der skal bruges til at køre det. Resten af tekstfilen indeholder kommandoer til kommandofortolkeren. For at styresystemet skal betragte filen som et program, skal den gøres kørbar (med **chmod**). Her er der et simpelt eksempel, der bare skriver »Hej med dig.«:

```
#! /bin/sh
echo Hej med dig.
```

Hvis de to linjer står i filen "`hej`", så gør du den kørbar med kommandoen:

```
[tyge@hven ~]$ chmod +x hej
```

Når du har gjort filen kørbar, så er den et program og kan køres som enhver anden kommando (næsten):

```
[tyge@hven ~]$ ./hej
Hej med dig.
```

`./` fortæller din kommandofortolker at det er et program der ligger i det katalog du står i, og ikke et kommandofortolkeren skal lede efter i kommandostien (systemvariablen "`PATH`").

Hvis du bare udfører kommandoen i filen "`hej`" direkte på kommandolinjen får du samme resultat:

```
[tyge@hven ~]$ echo Hej med dig.
Hej med dig.
```

I dette eksempel valgte vi at bruge "`/bin/sh`" som fortolker til programmet. Alternativt kunne fortolkeren blandt andet have været `/bin/bash`, `/usr/bin/gnuplot` (så ville "**echo**"-kommandoen dog ikke blive forstået) eller `/bin/zsh`. Henvisningen til fortolkeren *skal* skrives på første linje i filen.

Bemærk de første 4 tegn i filen er "**#! /**". Det er Unix-koden for at der bruges en fortolker til at køre dette program. I dette tilfælde er det så **sh** der er placeret i kataloget `/bin/`.

4.9.1. Flere kommandoer

Et af formålene med scripts er at køre flere kommandoer. Har man kun to kommandoer der skal køres, og det kun er dig selv der skal køre dem, kan man blot køre dem efter hinanden på kommando-linjen ved at angive et ; imellem kommandoerne.

```
[tyge@hven ~]$ echo "Hello, world" ; echo "Hej, verden"
Hello, world
Hej, verden
```

I et shell-script kan dette så enten skrives som:

```
#! /bin/sh
echo "Hello, world" ; echo "Hej, verden"
```

eller med en ny linje imellem hver kommando:

```
#! /bin/sh
echo "Hello, world"
echo "Hej, verden"
```

Bemærk at *;* blot betyder det samme som en ny linje med en ny kommando. I de følgende eksempler vil ; blive brugt flittigt, men det er egentlig kun for at få scriptet til at se pænere ud og blive mere læseligt.

Hvor ; kan bruges til at skrive flere kommandoer på den samme linje, kan det lige nævnes at \ kan bruges til at dele en meget lang linje op i flere linjer, så den bliver mere læselig. Dette kræver at \ står til sidst på linjen, og der må ikke være et mellemrum eller andet tegn efterfølgende.

```
#! /bin/sh
echo "Hello, " \
 "world"
```

I de følgende afsnit vil blive brugt lidt mere besværlige funktioner. Når det scriptet ikke vil som du vil, kan det ofte hjælpe lidt ved at få skrevet ud hvad der sker undervejs. Indsæt **set -x** for at se hvilke kommandoer der bliver udført.

```
#! /bin/sh
```

```
set -x
echo "Hello, world"

[tyge@hven ~]$ ./hello
+ echo 'Hello, world'
Hello, world
```

4.9.2. Variable

I shell script kommer man tit ud for at skulle bruge den samme værdi eller tekst flere gange. Her kan man med fordel bruge en variabel til at gemme værdien i. Variabelnavne kan både skrives med majuskler og minuskler, men de fleste skriver kun variabelnavne med majuskler.

```
#! /bin/sh
VERSION="0.1.2"
echo "Dette er version $VERSION af programmet."
echo $VERSION
```

Skal der være bogstaver eller tal lige efter variablen, er det nemmest at sætte et par {} omkring. Hvis ikke der sættes {} omkring variablen, vil fortolkeren opfatte den efterfølgende tekst som en del af variablens navn. Alternativt kunne man skrive to linjer, men det kan til tider ikke lade sig gøre.

```
#! /bin/sh
VERSION="0.1.2"
echo "Dette er version ${VERSION}beta af programmet."
```

Længden af en streng fås med *${#NAVN}*.

```
#! /bin/sh
echo "\$PATH indeholder ${#PATH} tegn"
```

Er en variabel ikke nødvendigvis allerede blevet tildelt en værdi, kan man angive en standardværdi (`${VARIABELNAVN:-STANDARDVÆRDI}`) eller tildele den en standardværdi (`${VARIABELNAVN:=STANDARDVÆRDI}`):

```
#! /bin/sh
unset FLOPPY # $FLOPPY har ingen værdi nu.
echo "Data skrives til ${FLOPPY:-/dev/fd0}"
echo "\$FLOPPY er stadig tom: '$FLOPPY'"
echo "Hvis \$FLOPPY er tom, så tilskriv: '${FLOPPY:=/dev/fd0}'"
echo "Nu har \$FLOPPY en værdi: '$FLOPPY'"
```

Og når scriptet køres:

```
[tyge@hven ~]$ ./hello
Data skrives til /dev/fd0
$FLOPPY er stadig tom: ”
Hvis $FLOPPY er tom, så tilskriv: ’/dev/fd0’
Nu har $FLOPPY en værdi: ’/dev/fd0’
```

Som et mere praktisk eksempel, kan vi skrive et program der enten monterer en diskette fra det første diskettedrev (`/dev/df0`) eller fra det diskettedrev programmet får på kommandolinjen:

```
#! /bin/sh
# Et mere praktisk eksempel hvor der enten bruges
# kommando-parameter eller default:
# (Du skal selvfølgelig selv slette ’echo’)
echo mount ${1:-/dev/fd0} /mnt/floppy
```

Vi afprøver det her både med og uden en kommandolinjeparameter:

```
[tyge@hven ~]$ ./hello
mount /dev/fd0 /mnt/floppy
[tyge@hven ~]$ ./hello /dev/fd1
mount /dev/fd1 /mnt/floppy
```

Variable kan splittes og manipuleres på forskellige måder med de indbyggede kommandoer. Nogle foretrækker de eksterne programmer såsom **cut** og **sed**, men de indbyggede kommandoer kan ofte klare opgaven. Skal man have fat i en del af en tekststreng, kan dette gøres med ${VAR:<start>:<antal>}, første tegn starter ved 0 (nul):

```
#! /bin/sh
TEKST=abcdef
echo "Skriv kun ’bcd’: ${TEKST:1:3}"
echo "Man kan også regne: ${TEKST:1:10-7}"
```

Skal en del af en variabel udskiftes med noget andet, bruges en substituering. Herunder er det de først forekommende 3 tegn ’bcd’ der udskiftes med 1 tegn ’X’:

```
#! /bin/sh
TEKST=abcdefabcdef
```

```
echo "Udskift 'bcd' med 'X': ${TEKST/bcd/X}"
```

```
[tyge@hven ~]$ ./hello
Udskift 'bcd' med 'X': aXefabcdef
```

Skal alle forekomster af et eller flere tegn udskiftes bruges »//«:

```
#! /bin/sh
TEKST=abcdefabcdef
echo "Udskift alle 'bcd' med 'X': ${TEKST//bcd/X}"
```

```
[tyge@hven ~]$ ./hello
Udskift alle 'bcd' med 'X': aXefaXef
```

Den første del af en tekst-streng kan slettes ved brug af '#'. Den tekst der skal fjernes, kan evt. matches ved brug af globbing, på samme måde som man kan liste filer med **ls**: *?[]. Dette kan for eksempel bruges til at slette den første del af hostname.

```
#! /bin/sh
echo "Fuldt hostname: ${HOSTNAME}"
echo "Uden maskinnavn: ${HOSTNAME#hven.}"
```

```
[tyge@hven ~]$ ./hello
Fuldt hostname: hven.sslug.dk
Uden maskinnavn: sslug.dk
```

Rigtig smart bliver det ved brug af globbing, hvor man kan fjerne alt frem til det første punktum.

```
#! /bin/sh
echo "Fuldt hostname: ${HOSTNAME}"
echo "Uden maskinnavn: ${HOSTNAME#*.}"
```

```
[tyge@hven ~]$ ./hello
Fuldt hostname: hven.sslug.dk
Uden maskinnavn: sslug.dk
```

En mere 'grådig' variant kan fjerne frem til den sidste forekomst af punktum. Her er det så '##' der skal bruges.

```
#! /bin/sh
echo "Fuldt hostname: ${HOSTNAME}"
echo "Uden maskinnavn og domæne: ${HOSTNAME##*.}"
```

```
[tyge@hven ~]$ ./hello
Fuldt hostname: hven.sslug.dk
```

```
Uden maskinnavn og domæne: dk
```

På samme måde som den første del af en tekst-streng kan fjernes, kan den sidste del fjernes med %.

```
#! /bin/sh
echo "Fuldt hostname: ${HOSTNAME}"
echo "Uden TLD: ${HOSTNAME%.*}"
```

```
[tyge@hven ~]$ ./hello
Fuldt hostname: hven.sslug.dk
Uden TLD: hven.sslug
```

Og den mere 'grådige' variant, hvor kun maskinnavn er tilbage ved brug af %%.

```
#! /bin/sh
echo "Fuldt hostname: ${HOSTNAME}"
echo "Uden TLD: ${HOSTNAME%%.*}"
```

```
[tyge@hven ~]$ ./hello
Fuldt hostname: hven.sslug.dk
Kun maskinnavn: hven
```

Et andet praktisk eksempel hvor den sidste del af tekst-streng skal fjernes, er ved filnavne. Har man en bunke filer der skal konverteres fra fx 'gif' til 'jpg', men beholde det samme filnavn, kan en lille løkke klare det.

```
#! /bin/sh
for FIL in *.gif; do
 echo "Konverterer ${FIL} til ${FIL%.*}.jpg"
 convert ${FIL} ${FIL%.*}.jpg
done
```

Og kør scriptet:

```
[tyge@hven ~]$ ./hello
Konverterer otto.gif til otto.jpg
Konverterer axel.gif til axel.jpg
```

Skal man splitte en tekststreng der er adskildt af et tegn, kan det gøres på flere måder. Nogle vil nok foretrække den mere behændige kommando **cut**, men det kan gøres direkte med de indbyggede kommandoer i shell. I det forgående er beskrevet hvordan ${X#*.} kan slette starten af en tekst der matcher #*.. Kommandoen ${X%%.*} sletter den sidste del der matcher %%.*. Hvis disse to regler kombineres lidt, kan tekststrengen splittes ved for eksempel tegnet punktum.

```
[tyge@hven ~]$ echo "Første del: ${HOSTNAME%%.*}"
Første del: hven
[tyge@hven ~]$ echo "Resten: ${HOSTNAME#*.}"
Resten: sslug.dk
```

Nu mangler vi så blot at gå ind i en løkke, indtil alle dele adskildt af '.' er skrevet ud. Til det formål skal vi bruge kommandoen *while*, hvilket vi kommer tilbage til. Ved det sidste element i listen går det galt, for da er der ikke noget punktum tilbage der kan slettes. Derfor kan løkken stoppes når både *REST* og *DEL* er ens. Sådan kan det gøres:

```
#! /bin/sh
REST=$HOSTNAME
DEL=""
while [ "$REST" != "$DEL" ]; do
        echo "Rest: '$REST'"
        DEL=${REST%%.*}  # slet bagfra til sidste forkomne punktum
        REST=${REST#*.}  # slet frem til og med første punktum
        echo "Del : '$DEL'"
done
```

Og kør scriptet:

```
[tyge@hven ~]$ ./hello
Rest: 'hven.sslug.dk'
Del : 'hven'
Rest: 'sslug.dk'
Del : 'sslug'
Rest: 'dk'
Del : 'dk'
```

Hvis du lige fjerne den linje der skriver $REST ud, så er det nok noget nær det du gerne vil have.

Læs mere om streng-manipulering med **man bash** under "Parameter Expansion".

4.9.3. Parametre til script

De fleste kommandoer eller programmer i UNIX kan startes med parametre på kommando-linjen, og få programmet til at opfører sig lidt anderledes. Parametre nummeres fra 1 og opefter, hvor ${0} er programmet selv.

```
#! /bin/sh
echo "Dette program hedder: ${0}"
echo "Første parameter: ${1}"
echo "Anden parameter: ${2}"
```

Programmet kan så afprøves:

```
[tyge@hven ~]$ ./hello Hello world
Dette program hedder: ./hello
Første parameter: Hello
Anden parameter: world
```

Skal du vide hvor mange parametere der er på kommando-linjen, ligger denne værdi i den specielle variabel $#.

```
#! /bin/sh
echo "Antal parametre: $#"
echo "Første parameter: $1"
echo "Anden parameter: $2"
```

Alle parametre til et script kan skrives ud på flere forskellige måder. Med **$*** skrives alle parametre ud, adskilt af indholdet af den specielle variabel *IFS*. Med **$@** skrives alle parametre ud som enkeltstående strenge. For at prøve de følgende eksempler, skal der to parametre på når du kalder scriptet, hvoraf det ene skal være en tekst med to ord og " omkring.

```
#! /bin/sh
echo "Alle parametre: $*"
IFS=","
echo "Alle parametre, nu adskilt af komma: $*"
echo "Alle parametre er adskilt af mellemrum: $@"
```

```
[tyge@hven ~]$ ./hello Hej "Linus Torvalds"
Alle parametre: Hej Linus Torvalds
Alle parametre, nu adskilt af komma: Hej,Linus Torvalds
Alle parametre er adskilt af mellemrum: Hej Linus Torvalds
```

Et program der kører under Linux har tildelt et *proces-id* – et nummer. Det er de numre som ses med for eksempel **ps** og **pstree -p**. Skal man stoppe et program der kører, kan det gøres ved brug af dette proces-id. Proces-id på det script der kører, findes i variablen *$$*. Denne værdi kan gives videre til et andet program eller script, men i dette eksempel vil vi blot stoppe scriptet selv.

```
#! /bin/sh
echo "Dette script har proces-id: $$"
echo "Det kan også ses med 'ps' kommandoen:"
ps | grep $$
kill $$
```

Starter man et program i baggrunden, så man har to programmer kørende samtidigt, kan det være interessant pludseligt at stoppe programmet i baggrunden. Proces-id for baggrundsprogrammer findes i *$!* variablen.

```
#! /bin/sh
# Start et program i baggrunden der tager tid
sleep 10 &
echo "'sleep' har proces-id: $!"
echo "Det kan også ses med 'ps' kommandoen:"
ps | grep $!
kill $!
```

For mere information om de specielle variable, se da **man bash** og søg efter "*Special Parameters*" .

4.9.4. if

Betingelser for et program eller et script er næsten uundgåeligt. Den mest brugte kontrol-struktur er **if**, som i den simpleste form ser således ud, hvor kommandoer der skal udføres er rykket ind for at gøre det mere læseligt:

```
#! /bin/sh
if true; then
 echo "Værdien er sand"
 echo "Man kan skrive flere linjer her"
fi
```

Først skal det bemærkes at **true** ikke er en specielt indbygget kommando i fortolkeren. **true** er et program der kaldes, og hvis det returnerer 0, er værdien sand. **if** har selvfølgelig også en **else**.

```
#! /bin/sh
if true; then
 echo "Værdien er sand"
else
 echo "Værdien er falsk"
fi
```

I Afsnit 4.9.1 blev ; nævnt som et tegn der bruges til at adskille kommandoer. Som det ses herover, er der sat et ; ind foran **then**, hvilket lige så godt kunne stå på næste linje. De fleste der skriver shell-scripts foretrækker dog at skrive det på ovenstående måde, selvom det ved første øjekast ser lidt underligt ud med en ; efter].

Parametren til **if** er altså et program der kaldes, og så udføres enten kommandoerne efter **then** eller efter **else**.

if bruges altid sammen med et andet program der enten returnerer sand eller falsk. Ofte er det kommandoen **[** eller **test if** bruges sammen med, hvilket er beskrevet i Afsnit 4.9.5.

4.9.5. Test [

Til test for om noget er sandt eller falsk, bruges slet og ret programmet **test**. I shell-scripts vil man dog ofte ikke skrive **test**, men derimod bruge det symbolske link **[**. Når man bruger [sammen med if, tror mange til at starte med, at [er en del af kommandoen til if, men det er altså et selvstændigt program. Når man har valgt at lave et symbolsk link til test, skyldes det blot at det er mere læseligt med [. Prøv kommandoen **ls -l /usr/bin/[** .

En meget brugt kommando med [, er at undersøge om en fil eksisterer. Dette gøres med parametren '-e'.

```
#! /bin/sh
if [ -e foo ]; then
```

```
 echo "Ja, filen 'foo' findes."
else
 echo "Nej, filen 'foo' findes ikke."
fi
```

Er det kun hvis filen ikke findes, at der skal gøres noget, bruges not-kommandoen **!**.

```
#! /bin/sh
if [ ! -e foo ]; then
 echo "Åh-nej, filen 'foo' findes ikke."
fi
```

Bemærk at der i ovenstående eksempel er brugt en hel del mellemrum ved **if [! -e foo]** . *Alle* disse mellemrum skal være der, ellers kommer der mange underlige fejl der er svære at få øje på. I andre programmeringssprog vil du nok undlade nogle af mellemrumene, men ingen kan undværes i dette eksempel.

test har et væld af muligheder, hvor vi kun viser de mest brugte her. For yderligere information om [kaldes **man test** .

```
#! /bin/sh
if [ -z "$1" ]; then
 echo "Fejl! Første parameter findes ikke (zero)."
fi

if [ ! -z "$1" ]; then
 echo "Første parameter findes, og er: $1"
fi

if [ -n "$1" ]; then
 echo "Første parameter findes (non-zero), og er: $1"
fi

if [ -x foo ]; then
 echo "Programmet 'foo' findes, og du kan køre det."
fi

TEKST="Ja"
if [ "$TEKST" = "Ja" ]; then
 echo "Tekstsvaret var 'Ja'."
fi

NUMMER=3
if [ $NUMMER -eq 3 ]; then
 echo "Heltallet i variablen \$NUMMER er 3"
else
 echo "\$NUMMER er lig med $NUMMER"
fi

if [ $NUMMER -lt 4 ]; then
```

```
 echo "\$NUMMER er mindre end 4"
fi

if [ $NUMMER -gt 0 -a $NUMMER -lt 4 ]; then
 echo "\$NUMMER er større end 0 og mindre end 4"
fi

if [ $# -le 2 ]; then
 echo "Der er 2 eller færre parametre på kommando-linjen"
fi
```

[kan også bruges i en mere kort form, hvis det kun er en enkelt kommando der efterfølgende skal udføres. Ved at sætte && ind, køres den anden kommando, hvis første kommando var sand. Læs mere om && i Afsnit 4.5.4.

```
#! /bin/sh
[ -e foo ] && echo "Filen 'foo' findes"
```

Læs mere om ovenstående parametre til [i **man test**, eller **man bash** under "CONDITIONAL EXPRESSIONS".

4.9.6. Case

Skal man undersøge mange forskellige tekst-strenge, kan det nemt blive uoverskueligt med **if**-kommandoer. Med **case** kan man nemt undersøge en hel række og gøre noget forskelligt ved hver situation. Herunder et simpelt eksempel der undersøger hvilket HOSTNAME der er på systemet.

```
#! /bin/sh
case $HOSTNAME in
 (hven.sslug.dk)
  echo "Vi er på Hven"
  ;;
 (saltholm.sslug.dk)
  echo "Vi er på Saltholm"
  ;;
 (*)
  echo "Vi er et andet sted: $HOSTNAME"
  ;;
esac
```

Ved kørsel på dit system får du nok et andet svar, fx:

```
[tyge@hven ~]$ ./hello
Vi er et andet sted: peberholm.sslug.dk
```

I stedet for at matche teksten helt præcist, kan man bruge *globbing* som med filnavne:

```
#! /bin/sh
case $1 in
 (ventilatore)
  echo "Match på 'ventilatore'"
  ;;
 (ventil*)
  echo "Match på 'ventil*': $1"
  ;;
 (ven*)
  echo "Match på 'ven*': $1"
  ;;
 (*)
  echo "Ingen match: $1"
  ;;
esac
```

Herefter kan forskellige ord så prøves af:

```
[tyge@hven ~]$ ./hello
Ingen match:
[tyge@hven ~]$ ./hello ventilator
Match på 'ventil*': ventilator
[tyge@hven ~]$ ./hello vender
Match på 'ven*': vender
[tyge@hven ~]$ ./hello world
Ingen match: world
```

Der er et mere avanceret eksempel med **case** i Afsnit 4.9.9 der er kombineret med **while**.

4.9.7. Processubstituering

Man kan komme ud for at skulle bruge output fra et program til et andet program. Dette kan gøres enten ved bruge af **‘kommando‘** eller **$(kommando)**. De to måder at gøre det på, udfører det samme. En simpel forklaring på hvornår processubstituering med fordel kan bruges, er hvis man først skal finde en fil og dernæst vil vide hvor stor filen er.

```
[tyge@hven ~]$ which mount
/bin/mount
[tyge@hven ~]$ ls -ho /bin/mount
-rwsr-xr-x    1 root          67K Feb 26  2002 /bin/mount*
```

Nu gør vi det samme, men denne gang 'gribes' output fra which, og bruges som parameter til ls.

```
[tyge@hven ~]$ ls -ho `which mount`
-rwsr-xr-x    1 root          67K Feb 26  2002 /bin/mount*
```

Og sidder du ved et fremmed tastatur hvor det ikke lige er til at finde tegnet **‘**, så brug **$()**.

```
[tyge@hven ~]$ ls -ho $(which mount)
-rwsr-xr-x    1 root          67K Feb 26  2002 /bin/mount*
```

I shell scripts ser man ofte at output fra et andet program bliver gemt i en variabel. Det kunne være et filnavn der er dannet ud fra dags dato.

```
#! /bin/sh
DATO=$(date -I)
FIL="backup-$DATO.tar.gz"
echo "Dagens backup gemmes i: $FIL"
tar czvf $FIL /home
```

```
[tyge@hven ~]$ ./hello
Dagens backup gemmes i: backup-2002-12-31.tar.gz
```

Mere information om kommando erstatning findes i **man bash** under *Command Substitution*.

4.9.8. For-løkker

For-løkker kan udføres på flere måder i shell scripts. Det kan være en liste af ord, liste af filnavne eller et matematisk udtryk. Den simpleste måde er en liste af ord.

```
#! /bin/sh
for NAVN in Tyge Otto Axel; do
 echo "Han hedder $NAVN"
done
```

```
[tyge@hven ~]$ ./hello
Han hedder Tyge
Han hedder Otto
Han hedder Axel
```

En liste af ord kan også stamme fra et andet program, hvor hvert ord så behandles for sig. Kommandoen **date** giver som udganspunkt en liste af ord og tal.

```
#! /bin/sh
date
for TEKST in $(date); do
 echo "Del-streng: $TEKST"
done
```

```
[tyge@hven ~]$ ./hello
Tue Dec 31 23:59:59 CET 2002
Del-streng: Tue
Del-streng: Dec
Del-streng: 31
...
```

For-løkker kan også bruges med fil-navne. Har man en masse filer der skal gøres noget med, skal man stå i katalog hvor filerne er, og så lave en globbing med for eksempel **.html*.

```
#! /bin/sh
for FIL in *.html; do
 echo "Filnavn er: $FIL"
 # Her gøres noget med filen.
 lynx -dump $FIL | lpr -o page-left=36
done
```

Når der så er filerne `index.html` og `tyge.html` bliver resultatet:

```
[tyge@hven ~]$ ./hello
Filnavn er: index.html
Filnavn er: tyge.html
```

Er det udvalgt liste af filer der skal gøres noget med, fx kopieres eller installeres, kan man skrive det pænt og organiseret.

```
#! /bin/sh
# Husk: \ til sidst på linjen, betyder at teksten
# fortsætter på næste linje.
FILER="\
index.html \
tyge.html \
otto.html \
"
for FIL in $FILER; do
 if [ -e $FIL ]; then
  echo "Filnavn er: $FIL"
 fi
done
```

```
[tyge@hven ~]$ ./hello
Filnavn er: index.html
Filnavn er: tyge.html
Filnavn er: otto.html
```

En for-løkke kan selvfølgelig også være en simpel talrække. Herunder udskrives blot tallene 1 til og med 4.

```
#! /bin/sh
for (( N=1; N<5; N++ )); do
 echo "Tal $N"
done
```

En anden måde man kan lave lange talrækker er ved brug af kommandoen **seq**. Herunder udskrives blot tallene 1 til og med 4.

```
#! /bin/sh
```

```
for N in $(seq 1 1 4); do
 echo "Tal $N"
done
```

En af fordelene ved **seq** er at tallene kan formateres, for eksempel med nuller foranstillet:

```
#! /bin/sh
for N in $(seq -f "%04g" 1 1 4); do
 echo "Tal $N"
done
```

Resultat:

```
[tyge@hven ~]$ ./hello
0001
0002
0003
0004
```

4.9.9. Løkke, while

while bruges til at blive ved med at udføre nogle kommandoer, sålænge en betingelse er opfyldt. Et simpelt eksempel der blot skriver det samme hele tiden:

```
#! /bin/sh
while true; do
 # Skriv sekunder
 date +%S
done
```

Et mere avanceret eksempel der tager parametre fra kommandolinjen og gør noget forskelligt med hver parameter:

```
#! /bin/sh
while [ $# -gt 0 ]; do
 case $1 in
  --help)
   echo "Her er hjælpen"
   exit 0
   ;;
  -f)
   # Næste parameter er filnavn
   # Det hentes frem med 'shift'
   shift
   FIL=$1
   ;;
  *)
   echo "Ukendt parameter: $1"
```

```
   exit 1
   ;;
 esac
 shift
done
```

Ovenstående eksempel er inspireret af shell-scriptet `/sbin/mkbootdisk`.

4.9.10. Regne, heltal

Ved brug af dobbelt-paranteser kan regneoperationer udføres direkte, og det er ikke nødvendigt at bruge **$** foran variable. Er er nogle eksempler:

```
#! /bin/sh
(( N=1 ))
echo "Antal filer: $(( N+2 ))"
(( N += 1))
let N += 1

(( N = 5 ))
while (( N -= 1 ))
do
 echo "$N"
done
```

4.9.11. read, input fra bruger

En nem måde at lave interaktive programmer er ved hjælp af shell-kommandoen **read**. Med **read** kan man indlæse input fra brugeren i en variabel, der så efterfølgende kan bruges i scriptet. En simpel måde at bruge **read** er ved at lave et ophold i et script, hvor brugeren så enten kan vælge at taste Enter eller Ctrl-C.

```
#! /bin/sh
echo "Tast Enter for at slette filen"
read
rm -f hello.world
```

read har mulighed for selv at skrive en ledetekst, og så bliver scriptet lige en linje mindre:

```
#! /bin/sh
read -p "Tast Enter for at slette filen"
rm -f hello.world
```

read er nok mest brugt til at indlæse input fra brugeren, og det kan gøres så simpelt på en kommandolinje hvor der indlæses til variablen "NAVN":

```
[tyge@hven ~]$  read -p "Navn: " NAVN
Navn: Tycho
[tyge@hven ~]$  echo $NAVN
Tycho
```

Ovenstående ser ud på samme måde i et script:

```
#! /bin/sh
read -p "Navn: " NAVN
echo "Navnet er: $NAVN"
```

Man kan undlade at angive hvilken variabel der skal indlæses til, og i stedet bruge den som **read** selv sætter. Navnet på den er `REPLY`. Scriptet kunne så se således ud:

```
#! /bin/sh
read -p "Navn: "
echo "Navnet er: $REPLY"
```

En anden parameter man kan give **read** er hvor mange tegn der skal indlæses. Det gøres med parametren **-n #**. Det kunne for eksempel bruges på følgende simple måde:

```
#! /bin/sh
read -n 1 -p "Tast en vilkårlig tast for at forsætte"
```

read giver en returkode ved kørsel. Hvis man afslutter indtastning med Enter er returkoden "0" og afslutter man med Ctrl-D er den "1". "0" svarer til "true" og "1" til false, og det kan kombineres med for eksempel while. Se følgende eksempel:

```
#! /bin/sh
echo "Indtast navne. Tast Ctrl-D for at afslutte."
while read NAVN; do
 echo $NAVN
done
```

I ovenstående eksempel er det reelt EOF (End-Of-File) der får løkken til at stoppe. Dette kan også bruges hvis input kommer fra et andet program. I eksemplet kommer input fra **ls** og kanaler så over til **read**. Det ser nok ud til at det er **while** der modtager input, men det er altså **read** der udføre indlæsningen.

```
#! /bin/sh
ls | while read FILNAVN; do
 echo "Filens navn er: $FILNAVN"
done
```

read kan også indlæse data fra første linje i en fil.

```
#! /bin/sh
read FIRSTLINE < foo.bar
echo $FIRSTLINE
```

Her har nu været gennemgået eksempler som har beskrevet hvordan **read** kan bruges til de mest almindelige ting. Skal det være mere "poppet" med farver og styring af skærmposition, så kan det gøres med ANSI-koder. For at bruge ANSI-koder skal man sende en ESCAPE til terminalen. Det er vist herunder med ^*[*, hvilket kan fås i editoren **vi** ved først at tast Ctrl-V og så Esc.

```
#! /bin/sh
read -p "^[[41mDit navn: " NAVN
echo "^[[0mNavn er: $NAVN"
```

Inden du kaster dig ud i at bruge ANSI-koder er det nok klogest at se på programmet **dialog** som nok kan bruges til mange af de ting du gerne vil.

4.10. Videre med Linux

Vi har vist en masse Linux-kommandoer, men dermed skal du ikke tro, at vi har vist dig alle muligheder i Linux. Dette kapitel yder på ingen måde Linux og Unix retfærdighed. Der er meget at lære, men fordelen er, at du hele tiden lærer noget nyt, men ikke behøver at være ekspert for at komme i gang. Der findes naturligvis et hav af bøger om Unix generelt – både gode og dårlige.

Vi vil her anbefale tre:

- *Unix* af Dave Taylor og James C. Armstrong, forlaget IDG. Er udkommet i år 2001 på dansk.
- Göran Andersons bog på svensk (http://www.sslug.dk/gnulinux)
- *Introduktion til Unix* af Knud Jørgen Kirkegaard og Torben Krog, Teknisk Forlag.

Desuden kan du måske have glæde af John Ray: *Linux på 10 minutter* fra IDG, som dog er noget overfladisk. Tilsvarende kan du måske også have glæde af *Linux for dummies* af Phil Hughes, som er udkommet på IDG. Begge bøger er oversat til dansk.

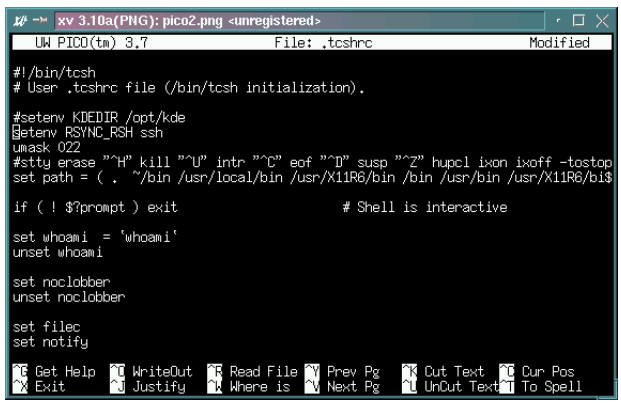
Kapitel 5. Teksteditorer

I Linux er tekstbehandling ofte nødvendig. F.eks. når opsætningsfiler skal tilrettes. Derfor er det vigtigt at vide noget om, hvordan tekster kan manipuleres med forskellige teksteditorer. Hvilke muligheder giver de forskellige teksteditorer? Hvilke begrænsninger har de enkelte teksteditorer? Hvornår skal jeg bruge **vi**? Hvilke andre editorer kan benyttes?

5.1. pico

Editoren **pico** installeres som en del af pine RPM-pakken. Med **pico** får du en ret nem teksteditor som også er en integreret del af postprogrammet **pine**. På Figur 5-1 er vist et terminal-vindue, hvor **pico** er startet op med en tekst-fil.

Figur 5-1. pico



Du skal ikke lære ret mange kommandoer med **pico** og de står endda i bunden af vinduet. Her betyder ^ at du skal holde Ctrl-tasten nede og samtidig trykke den efterfølgende tast.

5.2. mcedit

Programmet **mcedit** er teksteditoren som hører til filhåndteringsprogrammet Midnight Commander (en Norton Commander efterligning, som kaldes med **mc**). For en tidligere DOS-bruger er **mcedit** let at gå til og den har indbygget en simpel syntaksfremhævning til en hel del sprog.

Hvis **mcedit** skal kunne vise de danske tegn æ, ø og å, skal det ikke sættes op i **mcedit**, men i selve Midnight Commander. I **mc** vælges "F9, Options, Display Bits, Full 8 bits output og Full 8 bits input".

Derefter gemmes indstillingen med "Options, Save setup".

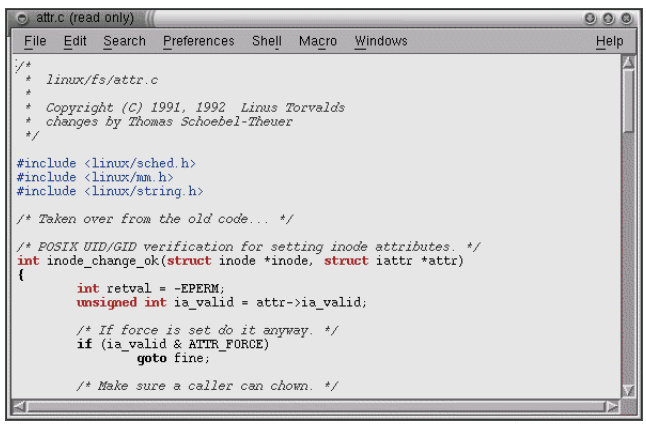
Den naturlige måde at benytte **mcedit** er at starte den inde fra Midnight Commander. Det gøres ved at placere markørbjælken på den ønskede fil og trykke F4 (Edit).

5.3. nedit

Hvis du er ude efter en enkel editor, som ikke kræver ret meget tilvænning, bør du se på **nedit**, som følger med SuSE, men ikke Red Hat. (Begge kører RPM-format, så det er ofte muligt at installere fra en SuSE-cd-rom på en Red Hat maskine).

Programmet **nedit** kan, som det er vist på Figur 5-2, lave syntaksfremhævning (eng. highlight). Der er en del muligheder i **nedit**, og alligevel er den nem at gå til, så vi kan varmt anbefale den. Der er dog også grund til at lære f.eks. **vi**, **pico** og/eller **emacs** senere, idet **nedit** ikke kan køre uden X-vinduesystemet. Med andre ord, hvis du skal redigere en fil i Linux teksttilstand, er det godt at kunne lidt andet. Desuden er **nedit** normalt ikke installeret på andre Unix-systemer, men **emacs** er ofte og **vi** er altid.

Figur 5-2. nedit med en C-fil læst ind



5.4. Den klassiske Unix-editor vi

Indtil videre har du set, hvordan du kan manipulere filer, men det er ofte meget nyttigt at kunne redigere i en tekstfil.

En editor, som man ikke kan komme uden om er **vi**. Som gamle Unix-folk vil vi forfattere give dig det råd, at du lærer at bruge **vi** og bruger editoren til at foretage mindre rettelser i filer. Editoren **vi** (som udtales vi-aj) har eksisteret i mange år, og det er nok en af de mest udbredte editorer i verden. Grunden er bl.a., at **vi** findes på alle Unix-systemer og er så lille, at den næsten altid startes op uanset belastning af maskinen. En anden meget god grund til (også) at lære en tekstbaseret editor som **vi** er at den dag din Linux-maskine har f.eks. et hardware-problem og du måske ikke kan starte i grafiktilstand eller skal køre i enkeltbrugertilstand (kun som systemadministrator), så er det editorer som **vi** du skal kunne bruge sikkert.

Du skal vide at **vi** nok kan være en pine for nybegynderen, men for den erfarne er **vi** en ganske kraftig teksteditor, med mulighed for kald af makroer/funktioner. **vi** findes også til andre styresystemer end Linux/Unix.

Lad os antage, at du vil skrive en eller anden tekst, og du vil have, at filen skal hedde `minFil.txt`. Editoren **vi** startes på følgende måde:

```
[tyge@hven MitKatalog]$  vi minFil.txt
```

Til at begynde med står der intet andet end en masse ~-tegn (kaldet tilde) – ét ~-tegn på hver linje. Det betyder, at disse linjer ikke eksisterer i filen endnu. **vi** arbejder i et af to modi: indsæt- eller kommando-modus. Trykker du på *i*, når du er i kommando-modus, går du over i indsæt-modus, mens du skal trykke på Escape, hvis du ønsker at komme i kommando-modus. Du kan se, hvilket modus du er i nederst på skærmen: Når der står -- *INSERT* --, er du i indsæt-modus.

En god måde at komme i gang med **vi** er at køre hjælplektionen (eng. tutor) igennem. Du finder `tutor`-filen under dokumentationen for **vi** og kopierer den over i dit eget katalog, så du ikke ødelægger den oprindelige, når du gennemgår lektionerne. Placeringen af `tutor`-filen kan variere lidt alt efter Linux-distribution og version af **vi** (5.6 nedenfor er versionnummeret).

```
[tyge@hven MitKatalog]$  cp /usr/doc/vim-common-5.6/tutor/tutor .
```

vi har en række funktioner, som du kan bruge i kommando-modus. Her er de mest almindelige.

Tabel 5-1. Oversigt over de mest anvendte vi-kommandoer.

Kommando	Forklaring
:help	Hjælp. Tast :q for at komme tilbage.
i	Skift til indsæt-modus. Nu kan der føjes til filen, fra hvor man står.
a	Skift til indsæt-modus for at tilføje. Nu kan der føjes til filen, efter der hvor man står.
A	Gå til slutning af linjen og skift til indsæt-modus.
R	Skift til overskrivnings-modus.

Kommando	**Forklaring**
o	Skift til indsæt-modus (open line) og indsæt ny linje under markør.
O	Skift til indsæt-modus (open line) og indsæt ny linje over markør.
ESC	Skift tilbage til kommando-modus.
h j k l	I kommando-modus virker disse som piletasterne til at flytte rundt i filen.
Piletaster	Bruges til at flytte rundt i filen.
_	Gå til første tegn på linjen.
$	Gå til slutning af linjen.
w	Gå et ord frem.
b	Gå et ord tilbage.
gg	Gå til første linje i filen.
G	Gå til sidste linje i filen.
<n>G	Gå til n'te linje i filen.
}	Gå et tekstafsnit frem.
{	Gå et tekstafsnit tilbage.
ma	Sæt mærke med tegnet 'a'.
'a	Gå til mærket 'a'.
tA	Gå fremad til tegnet 'A'.
x	Slet bogstav.
X	Slet bogstav til venstre for markør.
J	Slet linjeskift (join).
dw	Slet ord (delete word).
dd	Slet linje.
D	Slet resten af linjen fra markøren.
3dd	Slet tre linjer startende med linjen, du står i.
d}	Slet tekstafsnit fremad (frem til næste blanke linje).
dtA	Slet frem til tegnet 'A'.
u	Fortryd sidste ændring (undo).
Ctrl-R	Gentag sidste ændring (redo).
/tekst	Søg fremad efter "tekst". Tryk `/ ENTER` eller `n` for at søge videre.
?tekst	Søg bagud efter "tekst". Tryk `? ENTER` eller `n` for at søge videre.
: %s/STR1/STR2/g	Søg i alle linjer og erstat STR1 med STR2.
Y	Kopiér linjen du står i (yank).
4Y	Kopiér 4 linjer til buffer startende med linjen, du står i (yank).

Kommando	Forklaring
p	Indsæt fra buffer (put). Dette virker både efter yank- og delete-ordrerne.
:set number	Vis linjenumre.
:set nonumber	Vis ikke linjenumre.
<n>G	Gå til linje <n>.
Ctrl-f	Gå en side frem.
Ctrl-b	Gå en side tilbage.
Ctrl-g	Viser nuværende linjenummer
:w	Gem filen.
:w!	Gem filen ubetinget.
:q!	Stop `vi` uden at gemme filen.
:wq!	Gem filen ubetinget, og stop editoren.
ZZ	Gem filen, og stop editoren (samme som :wq!).
:w <NAVN>	Gem filen som <NAVN>.
:e <NAVN>	Hent filen <NAVN>.
:set wrap	Sæt linjeombrydning til.
:set nowrap	Slå linjeombrydning fra.
:set ts=2	Sæt tab-stop til 2 mellemrum.

En lidt nemmere måde at styre blok-markeringer i **vi** (eller rettere **vim**) er ved at bruge *visuel modus*. Tast **v**, **V** eller **Ctrl-v** og brug derefter piletasterne. Tast dernæst **d** for at klippe og **y** for at kopiere. Jump-kommandoer såsom **G** virker også.

Nok er **vi** en teksteditor, som er meget udbredt i Unix-verdenen, men du kan også læse lidt frem; der er alternativer, som er nemmere, såsom **pico** og **mcedit** samt nogle, der kan meget mere, såsom **emacs**.

Se mere om **vi** og udvidede versioner på http://www.vim.org. En ting man kan finde her, er at den udvidede **vi**, kaldet **vim** har syntaksfremhævning :)

Andre gode ressourcer er

- VIM Online http://vim.sourceforge.net/
- Vi Lovers Home Page http://www.thomer.com/thomer/vi/vi.html
- Mastering the VI editor http://www.eng.hawaii.edu/Tutor/vi.html

5.5. Emacs

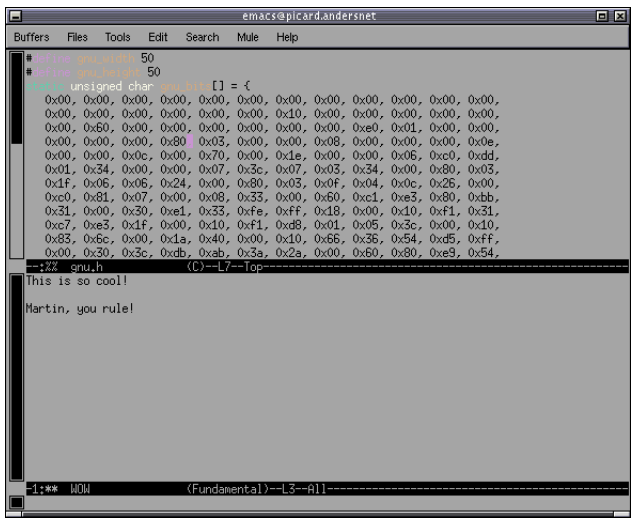
I en klasse for sig selv er Emacs (**emacs**) og søsterprogrammet XEmacs (**xemacs**), som begge kan

utroligt mange ting, men kræver meget hukommelse (den kørbare fil i X-udgave er på ca. 2,3 Mb). XEmacs er relativt enkel at bruge, idet der er både menuer og kommandoer – Emacs har ikke helt så smarte knapper, men kun tekstmenuer. Desværre følger kun Emacs og ikke XEmacs med de fleste udgaver af Red Hat, men XEmacs findes som Red Hat-pakker, og det er let at installere den.

Emacs og XEmacs har syntaksfremhævning til næsten alle programmeringssprog (også Matlab og LaTeX), hvilket gør dem meget lækre at arbejde med. Syntaksen i koden forstås, så indrykning af f.eks. C-kode sker automatisk, og koder vises med enten forskellige skrifttyper og/eller farver.

XEmacs og Emacs kan sættes op gennem et meget slagkraftigt makrosystem, så programmering eller tekstbehandling kan tilpasses ud fra næsten alle ønsker. Hvis det måtte ønskes, kan Emacs styre næsten alt. Du kan fra Emacs oversætte C-programmer, du har redigeret, og køre en debugger direkte fra Emacs. Du kan sende og modtage elektronisk post gennem Emacs. Der er endvidere mulighed for at starte en kommandofortolker op i Emacs, hvorfra programmer kan udføres. Alt dette kan køres samtidig med redigering af et vilkårligt antal filer. Mange siger tit "Emacs kan bl.a. alt", og pånær at lave kaffe er det ikke modsagt. Til tekstediting og programmering er Emacs en Rolls Royce.

Figur 5-3. Emacs med to filer åbne



Filer redigeres ikke direkte på harddisken, men arbejdes på i såkaldte buffere, altså kopieres filens indhold til hukommelsen, hvor Emacs så behandler den. Således forhindrer man de mest alvorlige fejl, der skyldes at to programmer vil bruge samme fil samtidigt. Det er dog i daglig brug ikke til at skelne fra anden redigering af tekstfiler, så man taler normalt om buffere og åbne filer i flæng. Der er én buffer som er helt speciel, nemlig minibufferen, der altid er åben. Den er altid den nederste linje, og den bruges af Emacs til at vise dig forskellige beskeder i, og det er den, man skriver kommandoer mv. i.

Start Emacs op ved at skrive **emacs**, hvad enten du er i en tekst-tilstand eller i en grafisk

brugergrænseflade. Emacs har menuer, hvorfra mange af funktionerne kan styres. Ud for hver funktion står også en tastaturkode. Derfor lærer du hurtigt at styre Emacs, også hvis du foretrækker tastaturkoder. Her gives en kort oversigt over de mest anvendte tastaturkoder. Ctrl står for kontroltasten, og Meta står for Meta-tasten, der i Linux er lig med Alt-tasten (eller Escape).

Tabel 5-2. Oversigt over de mest anvendte Emacs-kommandoer.

Kommando	Forklaring
Ctrl-h ?	Hjælp om hjælpesystemet. Den indbyggede hjælp er enormt god, når man lige lærer at bruge den.
Ctrl-x Ctrl-f	Find fil (til indlæsning). Indtast navnet på den ønskede fil, eller anvend tabulator til at få fil-liste, og brug musen (midterste knap) til at vælge. Hvis filen ikke findes i forvejen, vil den blive oprettet. Bemærk, at **Ctrl-x f** er noget helt andet!
Ctrl-x k	Dræb buffer. Anvendes til at fjerne den indlæste fil.
Ctrl-x Ctrl-s	Gem fil med samme filnavn.
Ctrl-x Ctrl-w	Gem fil med nyt filnavn.
Ctrl-s	Søg efter det, du angiver. Der skal ikke trykkes return til sidst. Tryk Ctrl-s for at søge til næste fremkomst af det valgte mønster.
Meta-%	Søg og erstat (tryk Alt-Shift-5). Til de enkelte steder, hvor der kan erstattes, kan der tastes y for erstat, n for spring videre og endelig kan tastes ! (udråbstegn) for at erstatte alle kommende forekomster.
Markering af tekst	Dette kan ske ved at trykke venstre museknap ned og trække ned over teksten. Teksten er nu i en redigerings-buffer.
Ctrl-_	Undo.
Ctrl-x 1	Vis kun én ramme i vindue.
Ctrl-x 2	Del aktuelle vindue i to rammer, som kan bruges til at redigere forskellige filer.
Ctrl-q TEGN	Quote – bruges til at indtaste f.eks. linjeskift Ctrl-M (Skriv Ctrl-q Ctrl-M).
Ctrl-w	Klipper markeret tekst til redigeringsbuffer. Sker hurtigere med et dobbeltklik på højre museknap.
Ctrl-y	Yank text, dvs. indsæt tekst. Bemærk, at dette kan ske hurtigere med midterste museknap.

Kommando	**Forklaring**
Meta-q	Formatér tekst til at fylde linjerne ud. Meget anvendelig til almindelig tekst, hvis linjerne er delt meget skævt. Emacs kan også genkende formler, som ikke vil blive ombrudt.
Meta-x	Kør en kommando. Man skriver kommandoens navn i minibufferen og trykker enter.

Det kan også nævnes, at Ctrl-venstre museknap giver en oversigt over filer, der er indlæst. På den måde kan du hoppe mellem filerne, du har indlæst. Med Ctrl-højre museknap kan skærmens skrift ændres. Bemærk, at Emacs kan og bør sættes op til brugeren. Det gøres i en `.emacs`-fil, der gemmes i hjemmekataloget. Her er et eksempel på indholdet af en `.emacs`-fil. Du kan nøjes med de første fire linjer til at begynde med. Senere kan du bygge videre på standardindstillingerne – alt efter dine ønsker.

```
;;De første fire linjer vil gøre, at de danske bogstaver virker rigtigt.
(set-language-environment "Latin-1")
(set-input-mode (car (current-input-mode))
  (nth 1 (current-input-mode))
  0)


;;Bind nogle gode funktioner til Ctrl- og musetasterne
;;Ctrl-venstre mus giver menu over åbne filer
;;Ctrl-midterste mus viser afsnit i filen som der kan hoppes til
;;    For C-filer er det alle funktioner (cool).
;;Ctrl-højre mus for at vælge skrifttype
(global-set-key [\C-down-mouse-1] 'mouse-buffer-menu)
(global-set-key [\C-down-mouse-2] 'imenu)
(global-set-key [\C-down-mouse-3] 'mouse-set-font)

;; Praktiske genveje for home, end, Ctrl-left/right og delete tasterne
(define-key global-map [home]    'beginning-of-line)
(define-key global-map [end]     'end-of-line)
(define-key global-map [C-left]  'backward-sexp)
(define-key global-map [C-right] 'forward-sexp)
(define-key global-map [delete] 'delete-char)

;;Følgende kommando gør at emacs virker (dvs. backspace virker), i teksttilstand
;;(hvilket man i X kan opnå ved at skrive "emacs -nw").

(keyboard-translate ?\C-h ?\C-?)
(define-key global-map "\C-x?" 'help)

;;Meta-g for goto linje nummer
(global-set-key "\M-g" 'goto-line)

;;Vis linje og kolonne numre
(setq line-number-mode t)
(setq column-number-mode t)
```

```
;;For C og HTML filer skal filerne vises med smart farvning
(setq c-mode-hook 'font-lock-mode)
(setq html-mode-hook 'font-lock-mode)

;; Udkommenter følgende linjer hvis du ikke vil
;; have menu-linjer med
;(menu-bar-mode 'nil)
;(tool-bar-mode 'nil)


;;Filer der ender på .c og .h er C-filer og .html er HTML-filer
(setq auto-mode-alist
  (append '(("\\.c$"  . c-mode)
            ("\\.h$"  . c-mode)
            ("\\.html$" . html-mode)
            ) auto-mode-alist))
```

Emacs kan som sagt alt på nær at lave kaffe (hvilket er forkert, da adskillige i tidens løb har fået Emacs til netop dette). Det betyder dog ikke at Emacs kan alt samtidigt på samme buffer; hver buffer tilpasser sig filtypen, således at hvis man redigerer en C-fil, optimerer Emacs sig til sådan brug, og hvis man skriver et LaTeX-dokument, tilpasser Emacs sig dette. Ved at Emacs tilpasser sig forstås, at menuerne, syntaksfremhævning, tastaturbindinger/genvejstaster, tilgængelige kommandoer osv. ændres således, at det, man skal bruge, altid er let tilgængeligt. Dette er såkaldte major modes, altså tilstande der i væsentlig grad ændrer den måde, hvorpå Emacs virker. Lige nu har jeg ca. 12 aktive buffere i Emacs, hvor den, jeg netop nu skriver i bruger SGML major mode, en anden dired major mode, nogle stykker fundamental mode (den allermest grundlæggende, der ingen specielle muligheder har) osv. I denne buffers nuværende tilstand, kunne jeg trykke Ctrl-c Ctrl-c, og Emacs ville kalde et program, der validerer den nuværende buffer, men hvis det havde været et LaTeX-dokument, jeg havde skrevet på, ville Ctrl-c Ctrl-c have kaldt **latex** på bufferen. Prøv at trykke Ctrl-h m i Emacs; det vil få en beskrivelse af den aktuelle major mode frem.

Man skal normalt ikke bekymre sig om, hvordan man får fremkaldt en ønsket major mode, da Emacs selv holder styr på filtyper ved hjælp af "efternavne" eller specielt indhold (eksempelvis kan Emacs se, at filer, der begynder med #!/bin/sh altid er shell-scripts). Er man dog ikke tilfreds med den major mode, man befinder sig i, kan man sagtens ændre den. Det gøres med **Meta-x [tilstandsnavn]-mode** (f.eks. **Meta-x sgml-mode**).

Hvor major modes tilbyder funktionalitet, der er specielt rettet mod visse filtyper, er der også minor modes, der er mere generelt anvendelige, og virker på tværs af major modes. Altså kan en minor mode som f.eks. **flyspell-mode**, der foretager løbende stavekontrol, bruges i **sgml-mode**, **LaTeX-mode** og så fremdeles. Når der foretages automatisk linjebrydning og syntaksfremhævning, er det faktisk en minor mode, der laver arbejdet, om end mange minor modes virker lidt forskelligt fra major mode til major mode. Syntaksfremhævningen skal f.eks. ikke farve de samme elementer i et LaTeX-dokument som i C-kode. Minor modes kaldes på samme måde som major modes.

Nu et lille sidespring. Historien går, at der i 1980'erne var mange Unix-folk, som var stærke **vi**-fanatikere, og der var måske lige så mange **emacs**-fans. For at få afklaret én gang for alle, hvilken editor der var bedst, blev der afholdt en dyst i paintball, hvor **vi**-holdet fik en kneben sejr. Dette bør naturligvis efterprøves en dag, da **emacs** har udviklet sig meget siden da.

Vi kan endvidere anbefale at købe en bog om Emacs, f.eks. *SAMS Teach yourself Emacs in 24 hours* af Jesper Pedersen, fra forlaget SAMS.

I øvrigt kan det nævnes, at det ikke bare er i Emacs at de fleste viste tastaturgenveje kan anvendes, men også i bash, tcsh, tekstindtastningsfelter i Netscape mv.

Er du ude på at finde bedre tricks til `.emacs` dvs. opsætningen af Emacs, så er et godt sted at starte på http://www.dotemacs.de/. Et eksempel er http://www.dotemacs.de/dotfiles/BenjaminRutt.emacs.html.

Tip: Skal man søge efter en tekst i Emacs og gerne vil have et overblik over de linjer hvor teksten optræder, så kan **Alt-x occur** anvendes. Man indtaster derefter søgestrengen og alle linjer med søgestrengen vises.

Tip: I bogen "Linux – Friheden til at programme" står der under CVS et afsnit om hvordan man smart kan sammenligne filer med hinanden med Emacs.

5.5.1. Mus med hjul og Emacs

For at benytte hjulet til at rulle op og ned i Emacs laves en Emacs Lisp-fil (en el-fil). Som root kan den placeres sammen med de andre el-filer i `/usr/share/emacs/20.2/lisp`. Dette muliggør lettere adgang, idet Emacs' "library-path" umiddelbart finder filen, når den skal indlæses. Man kan f.eks. kalde filen `mwheel.el`, og den kan se ud som følger:

```
;;; No copyright

;; Maintainer: Jan Eggert Kofoed  mailto:jan.kofoed@person.dk
;; Keywords: intellimouse

;; This file can be used with GNU Emacs.

;; The code was taken from
;;   www.inria.fr/koala/colas/mouse-wheel-scroll
;; which is maintained by Colas Nahaboo, but the code is put there
;; with courtesy of
;;    Sylvia Knight, Sylvia.Knight@cl.cam.ac.uk

;;; Code:

(defun up-slightly () (interactive) (scroll-up 5))
(defun down-slightly () (interactive) (scroll-down 5))
```

```
(global-set-key [mouse-4] 'down-slightly)
(global-set-key [mouse-5] 'up-slightly)
(defun up-one () (interactive) (scroll-up 1))
(defun down-one () (interactive) (scroll-down 1))
(global-set-key [S-mouse-4] 'down-one)
(global-set-key [S-mouse-5] 'up-one)

(defun up-a-lot () (interactive) (scroll-up))
(defun down-a-lot () (interactive) (scroll-down))
(global-set-key [C-mouse-4] 'down-a-lot)
(global-set-key [C-mouse-5] 'up-a-lot)
```

Filen kan også byte-oversættes med Emacs. Emacs kan så læse filen, når du skriver følgende linje ind i `~/.emacs`:

```
(load-library "mwheel")
```

5.6. Andre teksteditorer

Til Linux findes også **vim**, som er en udvidelse af **vi** bl.a. med farver, når du redigerer C-kode.

Lidt større editorer er **joe** og **jed**. Joe har et par aliases, når den er installeret, som hedder **jstar**, **jpico** og **jmacs**, og som standard er joe sat til at være **jstar**. Jstar er wordstar-kompatibel, og kommandoer ligner dem som blandt andet Borland anvendte i Turbo Pascal/Turbo C-oversætternes IDE. Jpico og Jmacs er selvfølgelig beregnet til at ligne hhv. pico og emacs i tastatur og funktioner.

En anden brugt editor som i øvrigt også findes til andet end Linux/Unix er Jed. Jed kan lave syntaksfremhævning og sættes til at emulere blandt andet Emacs, Wordstar, EDT og Brief.

Både Joe og Jed kan i øvrigt sættes op via hhv. `~/.joerc` og `~/.jedrc` så de mere eller mindre kan emulere ens foretrukne teksteditor.

Kapitel 6. Post

Gennemgang af en række tekstbaserede postprogrammer.

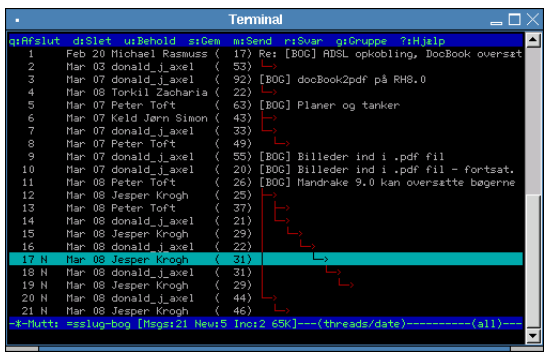
6.1. Gnus

Gnus er et postprogram der er indlejret i Emacs. Det kommer med de fleste Linux-distributioner. ...

6.2. Mutt

Mutt er et populært tekstbaseret postprogram. Det følger sandsyndligvis med din linuxdistribution, men er det ikke tilfældet kan det hentes fra http://www.mutt.org/download.html

Figur 6-1. Mutt i aktion



Før du begynder at bruge Mutt for alvor, er det nok en god idé at kigge på opsætningsfilen. Der findes en masse eksempler på opsætningsfiler på http://www.mutt.org/links.html#config

Her er et eksempel med det mest nødvendige:

```
# Den folder dine breve ligger i. Hvis du har brugt Pine tidligere,
# kan du bruge den samme
set folder=~/mail

# Sæt standard-brevbakken
set spoolfile=+mbox

# Her kan du angive i hvilken rækkefølge dine brevbakker skal vises,
```

```
# når du skifter i mellem dem
mailboxes ! +mbox +sslug-bog +sslug-misc +osv

# Vælg den epostadresse du vil have folk skal bruge
my_hdr From: Tyge <tyge@sslug.dk>

# Skjul alle felter i brevhovedet:
ignore *
# ... bortset fra nogle enkelte:
unignore From: Date: Subject: To: Cc: Reply-to:

# Sorter i tråde
set sort=threads
```

Nu skulle du være klar til at bruge Mutt. Programmet startes med kommandoen **mutt**. Som standard vises din indbakke. Du kan bevæge dig op og ned med enten piltasterne eller j og k (ligesom VI). Normalt kan man nøjes med at bruge tabulator som hopper direkte til næste ulæste brev. For at læse et brev trykker du enter og for at komme tilbage til brevbakken trykker du på i.

Når du læser et brev skal du bevæge dig rundt med enten mellemrum som viser næste side, eller enter for at gå en linje ned og backspace for at gå en linje op. Prøver du derimod at bruge pil op og ned (ligesom man gør i Pine) bevæger du dig videre til næste eller forrige brev.

Her er nogle af de vigtigste kommandoer i Mutt:

- q: (Quit) Afslutter Mutt
- d: (Delete) Sletter et brev
- r: (Reply) Svarer på et brev
- g: (Groupreply) Svarer til alle modtagere af et brev
- m: (Mail) Skriv et nyt brev
- c: (Change) Skift til en anden brevbakke
- ?: Viser hjælpen

6.2.1. Skrive breve

For at skrive et nyt brev skal du trykke på m. Du vil nu blive spurgt (i nederste linje på skærmen) om modtagerens epostadresse og emnet på brevet. Når det er indtastet startes en editor op, og du kan skrive dit brev. Når du er færdig med dette, gemmer du som normalt og afslutter editoren.

Nu har du mulighed for at tilføje vedhæftninger ved at trykke på a, skifte emnet med s, ændre modtageren med t og selvfølgelig at sende brevet med y.

6.2.2. Adressebog

Mutt har ikke en adressebog på samme måde som for eksempel Pine. I stedet tilføjes adresser til ens opsætningsfil som aliaser. For eksempel vil følgende linje give personen Tyge med epostadressen tyge@sslug.dk aliaset tyge (den første forekomst af "tyge" er aliaset, den anden er navnet på personen):

```
alias tyge Tyge <tyge@sslug.dk>
```

Dette kan også gøres simplere ved at finde et brev fra personen og trykke på a. Så kan du bare følge vejledningerne i bunden af skærmen, hvorefter personen automatisk vil blive tilføjet til din opsætningsfil.

Når du fremover vil skrive til en person, kan du nøjes med at skrive aliaset, hvorefter det vil blive skiftet ud med personens navn og epostadresse.

6.3. Pine

Pine er et tekstbaseret postprogram. Det er ikke rigtig Open Source, men man kan få kildeteksten til programmet og det fungerer på de fleste Unix-systemer. ...

Pine er et af de gode programmer til at læse og håndtere e-post. Let håndtering af flere postkasser, en lille og god indbygget teksteditor (**pico**), søgemuligheder, mulighed for at se HTML-formaterede breve og et utal af opsætningsmuligheder gør Pine til et hit.

I den version af Pine som kommer med Red Hat er der endda muligheder for at definere roller (bl.a. med forskellige email-signaturfiler) og farveopsætning. Som det ses på de følgende billeder kan man få forskellige farver på hvert svar-niveau.

Figur 6-2. Pine med farver

En egentlig vejledning til pine er at finde på http://www.washington.edu/pine/tutorial.4/index.html

Lad os tage et par nyttige tricks med pine.

6.3.1. Stavekontrol i Pine

Tip: Når du er i gang med at redigere et brev har du mulighed for at få det tjekket for stavefejl. Når du taster Ctrl-T starter Pine det stavekontrolprogram det er sat op til at bruge. Normalt vil det være Ispell med en engelsk ordbog. Hvis du skriver breve på flere sprog kan det være lidt irriterende med Pines en-sprogede stavekontrol. Programmet **pine-spell** (http://www.linuxbog.dk/unix/eksempler/pine-spell), der kommer med som et af bogens eksempler, er et forsøg på at rette op på dette problem. Hvis du installerer det på dit unix-system og sætter Pine til at bruge det, vil du få mulighed for at vælge mellem alle de sprog der er ordlister til på systemet, når du taster Ctrl-T. Du skal have installeret `aspell-da-*.rpm` ud over pine selv.

Kommandoerne der vises her installerer **pine-spell** i dit eget programkatalog (`~/bin/`):

```
tyge@hven ~/> mkdir -p ~/bin
tyge@hven ~/> wget -q www.linuxbog.dk/unix/eksempler/pine-spell -O ~/bin/pine-spell && O.k.
O.k.
tyge@hven ~/> chmod a+x ~/bin/pine-spell
```

Hvis der ikke kommer et "O.k." gik noget galt. Så bliver du nødt til at køre kommandoen uden tilvalget `-q`, så der kommer lidt information om hvad der sker.

Vi går nu ud fra at **pine-spell** er installeret i dit eget programkatalog og du har startet Pine. For at sætte Pine op skal du først ud i hovedmenuen. Hvis du ikke allerede er der, så tast M for at komme der ud. Her taster du så S efterfulgt af C for at komme ind i opsætningen. Du skal finde feltet "speller" (prøv W efterfulgt af "speller" og et linjeskift) og ændre det til "`~/bin/pine-spell`". Når det er gjort taster du

E efterfulgt af Y for at gemme ændringen. Næste gang du har skrevet et brev og vil undersøge om der er stavefejl i det, så taster du blot Ctrl-T inden du sender det.

6.3.2. Søgning med Pine

Du kan lave meget avancerede søgninger i dine e-post-foldere. For at få dette valgt, skal du fra hovedmenuen trykke **s c w agg** og afslutte med retur-tasten. Du bør nu stå på et af de mange felter i opsætningen, hvor der står "enable-aggregate-command-set". Tryk **x** for at vælge, **e** for exit og bekræft med **y**. Som et eksempel kan vi nu gå ned i en e-post-folder og finde e-post, hvor ordet Andeby er nævnt et vilkårligt sted. Når du er i e-post-folderen trykker du **;**, derefter har du en del forskellige valgmuligheder – vi vælger **t** for tekstsøgning. Igen har vi en del muligheder og da vi vil søge i alle dele af posten vælger vi **a**. Nu skrives søgestrengen `Andeby` ind og du kan nu med **z** skifte mellem alt post og kun dem, der netop indeholder Andeby. Får man disse kommandoer lært kan man virkelig søge hurtigt og effektivt.

6.3.3. Afsender og roller

Nogle personer sidder ikke på den maskine man ønsker e-post sendt tilbage til, og man har derfor brug for at sætte sin afsender-e-post-adresse til eksakt det man vil have. Fra hovedmenuen kan vi trykke **s** (for setup), **c** (for config), **w customized-hdrs** og afslutte med retur. Du er nu på `customized-hdrs`-feltet. Skriv `From: Tyge <tyge@sslug.dk>`, hvis e-post skal komme fra "Tyge <tyge@sslug.dk>". Du skal også sætte `feature-list=allow-changing-from` manuelt i `~/.pinerc`.

En anden smart mulighed i Pine er muligheden for at definere roller – dvs. at man kan have en afsender-adresse og signatur-fil for hver identitet man har defineret – f.eks. en privat hhv. en firma-identitet. Vælg først Vælg *m*(ain menu) *s*(setup) *c*(config) *w*(word to find) *confirm-role*. Sæt kryds i *confirm-role-even-for-default* (med x) og vælg e og Yes. Vælg *s*(setup) *r*(rules) *r*(roles) *a*(add). Derefter kommer man ind i en menu for en af roller man kan definere. Sæt følgende

- Nickname : Skriv et navn her for den identitet man definerer – f.eks. fornavn, loginnavn, privat, eller lignende. Det er dette "Nickname" man vælger for hver e-post der skal sendes.
- From pattern : Skriv her din fulde afsender identitet – f.eks. Tyge Brahe <tyge@sslug.dk>
- Sender pattern : Skriv her din email-adresse – f.eks. tyge@sslug.dk
- Set Signature : Her kan man skrive filnavnet som indeholder signaturfil for den nuværende identitet. Anføres ikke fuld sti vil filnavnet være i forhold til brugerens hjemmekatalog. F.eks. .signaturfil
- Set From : Sættes som "From pattern" – f.eks. Tyge Brahe <tyge@sslug.dk>

Tryk *e* (Exit Setup) når du er færdig og *a* for at addere hver af de roller du vil definere. Endelig trykkes *e* (Exit Setup) og *y* til at acceptere at der laves "Commit changes".

Når man derefter vil sende eller svare på e-post med *c* (compose) får man en ekstra menu i bunden af Pine:

```
Press Return to Compose using no role, or ^T to select a role
 ? Help    Ret [Compose] ^T To Select Role
^C Cancel
```

Tryk Ctrl-t (eller Ctrl-T) for at vælge ^T. Gå ned på den rolle du vil anvende og tryk på retur-tasten to gange. Nu skriver du e-post i den valgte rolle. Bemærk at "From"-feltet nu skal være sat svarende til den identitet du har valgt. Metoden virker også ved svar på e-post.

6.3.4. Små nyttige tips til opsætning af Pine

Tip: Under setup, skal du sætte "character-set = UTF-8" for at få dansk tegnsæt til at virke.

Tip: Pine kender ikke noget til tegnkodninger, men sender som standard bare brevene direkte ud til terminalen bit for bit. Hvis man for eksempel modtager breve kodet med UTF-8 og ens terminal arbejder med UTF-8 kan det give problemer. Man kan komme omkring det ved at lade Pine kalde et tegntabelomkodningsprogram, når det skal vise et brev kodet med en anden kodning end den terminalen forventer. Man kan for eksempel installere programmet "yudit". På Mandrake 9.0 skrives blot **urpmi yudit** som "root". På Red Hat 7.2 og 7.3 skal man køre kommandoen **rpm --upgrade ftp://rpmfind.net/linux/rhcontrib/7.2/i386/yudit-2.4-4.i386.rpm** som "root". Når Yudit er installeret skal følgende linje føjes til filen `~/.pinerc`:

```
display-filters=_CHARSET(UTF-8)_ /usr/bin/uniconv -encode iso-8859-1 -decode utf-8
```

Tip: Vil du se alle headere i din e-post, skal du under setup sætte "enable-full-header-cmd". Tryk på h for at se alle headere når du læser post.

Tip: Pine kan farvekode beskederne når du læser dem, så det er lettere at se, hvem der skriver hvad i en diskussion. Fra hovedmenuen trykkes "s k", hvorefter farver på oversigten, "quotelevels" og meget andet kan sættes op.

Tip: Hvis du vil kunne hoppe direkte til en URL angivet i en e-post, kan du give fuld sti til din favoritbrowser i setup under url-viewers, f.eks. **/usr/bin/netscape**.

Tip: Vil du gerne have pæn udskrift på printeren af dine tekst-emails, så kan du med fordel installere programmet **enscript**, der følger med de fleste Linux-distributioner (eller kan findes via http://rpmfind.net). Sæt din udprintningskommando som følger: I hovedmenuen af Pine, tryk "s" "p" "pil ned" "pil ned" "c". Sæt kommandoen til **enscript -2rG --word-wrap**, så får du to kolonner og der deles linjer ved ordgrænserne.

Du kan naturlig generelt bruge **enscript -2rG --word-wrap TEKSTFILNAVN** til at udskrive tekstfiler til printeren.

Kapitel 7. Web

Gennemgang af en række tekstbaserede webbrowsere.

7.1. Links

Kan du undvære grafik, bør du kende **links**, som er en tekstbaseret web-browser, som faktisk er ret anvendelig – pånær ved klikbare billeder. Grafik hentes kun (til disk), hvis dette vælges. Links er smart, hvis du har en langsom netforbindelse, eller hvis du vil lave applikationer, som automatisk skal kunne hente HTML eller billeder.

7.2. Lynx

Kan du undvære grafik, bør du kende **lynx**, som er en tekstbaseret web-browser, som faktisk er ret anvendelig – pånær ved klikbare billeder. Grafik hentes kun (til disk), hvis dette vælges. Lynx er smart, hvis du har en langsom netforbindelse, eller hvis du vil lave applikationer, som automatisk skal kunne hente HTML eller billeder.

7.3. Wget

Umiddelbart skulle man ikke mene at et tekst-baseret program som **wget** kunne være interessant at anvende til at hente filer fra internettet. Skal man lave en hel kopi af en hjemmeside med undersider så er det ikke sjovt at lave bare med en browser. Tilsvarende kan det være fedt at hente en del af en ISO-fil på 600 MB over et par omgange og endda selv kunne bestemme hvor meget båndbredde man vil anvende til dette.

Skal man hente en enkelt fil:

```
[tyge@hven ~]$ wget http://www.server.dk/FILNAVN
```

Skal man senere fortsætte en afbrudt hentning af filen, så kan man fortsætte med:

```
[tyge@hven ~]$ wget -c http://www.server.dk/FILNAVN
```

Vil man lave et spejl (eng. mirror) af en server med alle hjemmesider, så anvendes følgende.

```
[tyge@hven ~]$ wget -np --mirror http://www.server.dk/
```

Vil man begrænse den båndbredde som anvendes på at hente f.eks. en ISO-fil, så tilføj parameteren `limit-rate`.

```
[tyge@hven ~]$ wget --limit-rate=RATE http://www.server.dk/fil.iso
```

Se også **ncftp** for et ligende program til ftp-servere.

Kapitel 8. Usenet

Gennemgang af en række tekstbaserede nyhedslæsere.

8.1. Gnus

Gnus er en nyhedslæser der er indlejret i Emacs. Det kommer med de fleste Linux-distributioner. ...

8.2. NN (No News ...)

... lidt om NN som nyhedslæser ...

8.3. Slrn

Der er mange forskellige nyhedslæsere. Én af dem er **slrn** og den kommer med så godt som alle Linux-distributioner. Den startes ved at skrive **slrn** på kommandolinjen og er en ikke-grafisk newsreader, men det gør den nu ikke dårligere, snarere tværtimod. Den er nemlig lynhurtig.

8.3.1. Opsætning af slrn

For at **slrn** kan starte skal den vide hvilken NNTP-server den skal læse fra. Systemvariablen NNTPSERVER bestemmer hvilken server slrn læser fra; i dette eksempel egen maskine:

```
[tyge@hven ~]$ export NNTPSERVER=localhost
```

Til start skal **slrn** bruge en scorefil. Den laves med:

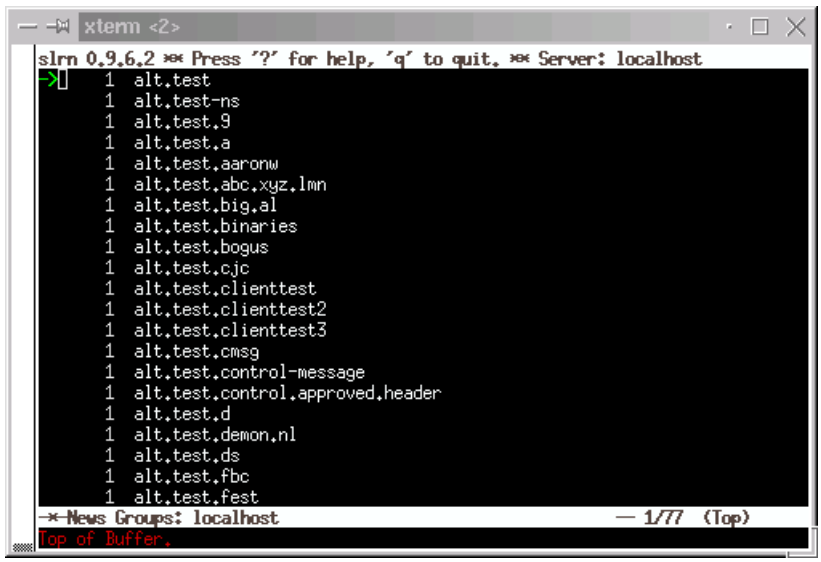
```
[tyge@hven ~]$ touch News/Score
```

hvorefter **slrn** startes første gang:

```
[tyge@hven ~]$ slrn -create
```

Det første indtryk af **slrn** er at det er et uindbydende nyheds-læseprogram, med grupper strøet efter forgodtbefindende.

Figur 8-1. slrn newsreader



Fortvivl ikke! **slrn**'s display styres med opsætningsfilen `~/.slrnrc`. En god opsætningsfil (som kan hentes under `eksempler` på www.linuxbog.dk/ (http://www.linuxbog.dk/)) kan se således ud:

```
% Opsætningsfil til slrn - gemmes i ~/.slrnrc
% sørger for at poste med rigtig e-post-adresse.
% Username og hostname bliver sat sammen til afsenderadressse.

set username "tyge"
set hostname "hven.sslug.dk"
set realname "Tyge Brahe"

% replies skal også se ordentligt ud.

set quote_string "> "

% Headers to show when viewing an articles.  This is a comma-separated
% list of strings that specify what headers to show.  Note that these
% strings are not regular expressions.  However, one may use, e.g.,
% "X-" to match any header beginning with "X-".  Similarly, "F" will
% match "From:" and "Followup".

visible_headers "From:,Subject:,Followup-To:,Reply-To:,Date:"


% Sortering
set sorting_method 9
```

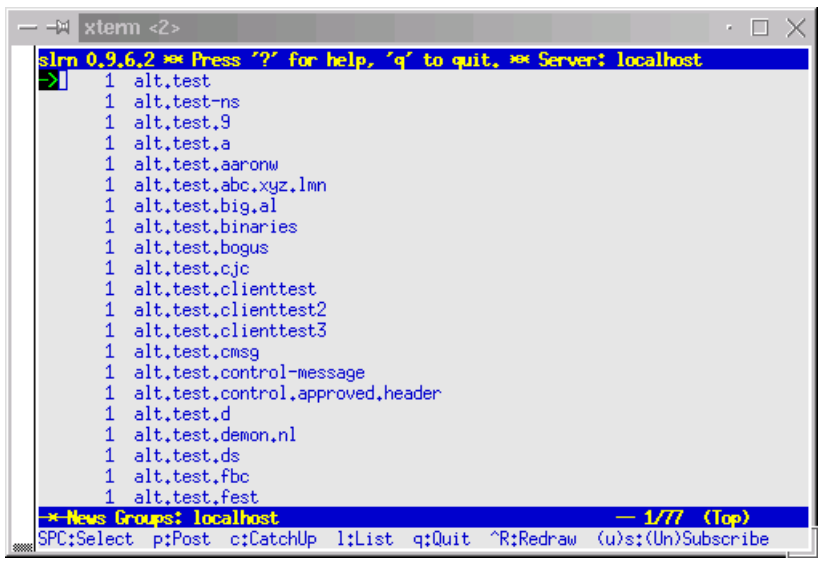
```
set new_subject_breaks_threads 0

%Definere "gode" farver.

color article           "blue"          "white"
color author            "magenta"       "white"
color boldtext          "brightblue"    "white"
color box               "black"         "white"
color cursor            "brightgreen"   "black"
color description       "brightmagenta" "white"
color error             "red"           "white"
color frame             "yellow"        "blue"
color group             "blue"          "white"
color grouplens_display "blue"          "white"
color header_name       "green"         "white"
color header_number     "green"         "white"
color headers           "blue"          "white"
color high_score        "red"           "black"
color italicstext       "magenta"       "white"
color menu              "yellow"        "blue"
color menu_press        "blue"          "yellow"
color normal            "blue"          "white"
color pgpsignature      "blue"          "white"
color quotes            "red"           "white"
color quotes1           "magenta"       "white"
color quotes2           "magenta"       "white"
color quotes3           "magenta"       "white"
color quotes4           "magenta"       "white"
color quotes5           "magenta"       "white"
color quotes6           "magenta"       "white"
color quotes7           "magenta"       "white"
color response_char     "green"         "white"
color signature         "red"           "white"
color selection         "yellow"        "blue"
color status            "yellow"        "blue"
color subject           "red"           "white"
color thread_number     "blue"          "white"
color tilde             "green"         "white"
color tree              "red"           "white"
color underlinetext     "cyan"          "white"
color verbatum          "green"         "white"

% Mime support
%-----------------------------------------------
set use_mime 1
% set mime_charset "iso-8859-1"
% If non-zero, call metamail for mime formats
% that slrn does not handle
set use_metamail 1

% En sidste ting der kunne være interessant er:
editor_command    "jed %s -g %d -tmp"
% Denne sætter slrn til at bruge jed som editor i
```

```
% stedet for default (vi). Her putter man så bare
% sin favorit editor ind i stedet for.
```

Filen kan findes under eksempler på bogens hjemmeside www.linuxbog.dk/ (http://www.linuxbog.dk/).

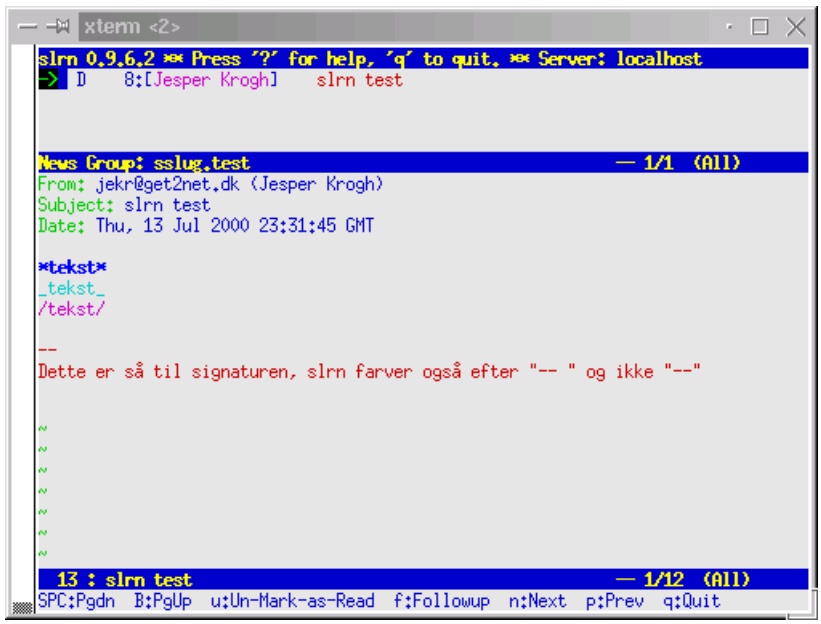
Alle disse parametre er fint beskrevet i **man slrn**. Men det mest interessante er resultatet af dette:

Figur 8-2. Slrn efter opsætning



Lidt mere brugervenlig. Derefter afmeldes de grupper man ikke ønsker at vise. Dette sker ved at trykke u og afslutte med g. Så giver man sig til at tilmelde grupper. Dette sker ved a + gruppens navn, men da man ikke lige kender det altid, ville det være skønnere med en liste. Det gøres med **"L" + sslug*** og så trykkes der bare **"s"** ud for de grupper man ønsker at abonnere på. Når man så læser et indlæg, så farver slrn efter den formatering der er markeret for i ASCII-teksten.

Figur 8-3. Nyhedslæseren slrn



Ydermere er alle basis-kommandoerne lagt på "?", så de er lige ved hånden hvis der er noget man ikke lige kan få slrn til. Derudover, så god fornøjelse med slrn.

8.4. Tin

Tin er et tekstbaseret postprogram. ...

Kapitel 9. IRC

Gennemgang af en række tekstbaserede IRC-programmer.

Kapitel 10. Linux og netværk – klientsiden

I dette kapitel vil vi se på klientsiden af netværk.

En vigtig start-kommentar, som kunne stå mange steder i denne bog er at der med Linux-kerne 2.4 (f.eks. Red Hat 7.3 og Mandrake 8.2) er problemer med at få tilgang til alle hjemmesider – det er et problem som er forstået nu, se f.eks. http://eltoday.com/article.php3?ltsn=2001-04-17-001-14-PS. Løsningen er simpel. Tilføj følgende til f.eks. `/etc/rc.d/rc.local`:

```
echo "0" > /proc/sys/net/ipv4/tcp_ecn
```

10.1. Webbrowsere

10.1.1. Verificere indhold af cd-rom

Det er irriterende ved installation at opdage at er sket en fejl da man hentede en cd-rom fra internettet. Derfor er der ofte en fil man kan hente der hedder noget med *md5sum*, f.eks. `md5sums` eller lign. Programmet **md5sum** kan udregne en tjeksum for en fil – og dermed også for et helt ISO-cd-rom-billede. At to cd-rom'er eller to filer skulle have samme MD5-tjeksum er ekstremt usandsynligt.

```
[tyge@hven ~]$ md5dum Mandrake91-cd1-inst.i586.iso
6f1581974e12420fef87868ed6caa31f  Mandrake91-cd1-inst.i586.iso
```

Har man en fil med MD5-tjeksummer, kan med nemt få tjekket alle de filer der står i den:

```
[tyge@hven ~]$ cat md5sums
6f1581974e12420fef87868ed6caa31f  Mandrake91-cd1-inst.i586.iso
87afe11ddef6b619866322aa0797e45f  Mandrake91-cd2-ext.i586.iso
ff187c7a552722f42790b5726fdb62b3  Mandrake91-cd3-i18n.i586.iso
[tyge@hven ~]$ md5sum --check md5sums
Mandrake91-cd1-inst.i586.iso: OK
Mandrake91-cd2-ext.i586.iso: OK
Mandrake91-cd3-i18n.i586.iso: OK
```

Hvis man f.eks. retter en smule i filen `Mandrake91-cd1-inst.i586.iso` så vil man få en helt anden tjeksum og anvender man igen **md5sum --check md5sums** vil man få beskeden **FAILED** som indikation for at det gik galt.

10.2. E-post

Til at læse e-breve kan du på næsten alle Unix-systemer finde programmet **mail**, som er meget simpelt. Skrives f.eks.

```
[tyge@hven ~]$  mail BRUGERNAVN@MASKINE.DOMÆNE help
```

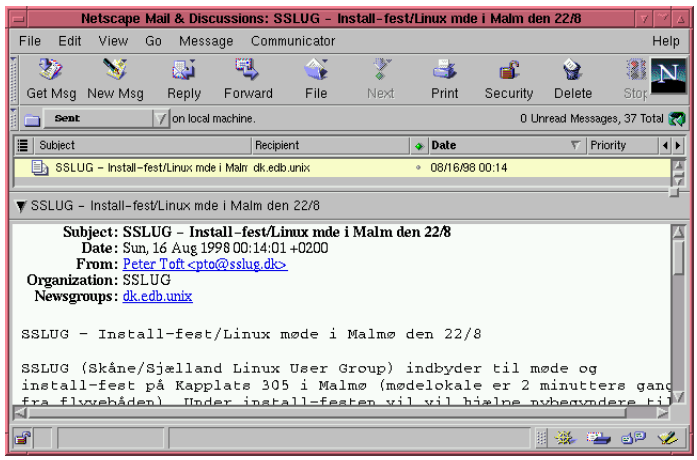
og afsluttes med Ctrl-D så er der sendt et brev til `BRUGERNAVN@MASKINE.DOMÆNE` med indholdet `help`.

Senere er Elm (**elm**) kommet med lidt flere muligheder, men ELM bliver ikke videreudviklet mere.

Til tekstbaseret læsning og skrivning af e-breve er der to gode valg. Mutt (**mutt**) er hurtig og har mange funktioner.

Det skal også nævnes, at Netscape har indbygget post-håndtering, som foretrækkes af mange. Det er grafisk og ret enkelt at bruge.

Figur 10-1. Netscapes mail-funktion



10.2.1. Vise nye e-breve

... kan man ikke sætte en systemvariabel i sin kommandofortolker, så man får besked, når der er ny post? ...

10.2.2. Hente e-post fra POP-servere med Fetchmail

fetchmail bruges til at hente e-breve fra en postserver hos din internetudbyder. Programmet styres gennem filen `.fetchmailrc` i dit hjemmebibliotek. Den kan f.eks. se sådan her ud:

```
poll post.fiktiv.dk
proto POP3
user tyge
pass McMombo
```

Når du kører **fetchmail**, vil den i dette eksempel forsøge at hente post fra serveren `post.fiktiv.dk` med protokollen POP3 (som er den mest almindelige) og logge på serveren med brugernavnet "tyge" og adgangskoden "McMombo". Hvis der er post til "tyge", leverer fetchmail den videre til det lokale postsystem, der så lægger brevene i Tyges postkasse.

De første gange, du kører **fetchmail**, vil du som regel gerne se, om tingene går, som de skal, når posten hentes. Hvis **fetchmail** køres med parameteren `-v`, skriver den, hvad der foregår undervejs.

Hvis du er til grafisk baserede opsætningsværktøjer, kan du redigere din `.fetchmailrc` ved hjælp af programmet **fetchmailconf**.

*Tip:*Nogle post-servere vil ikke tillade at du bruger POP3 fra andre maskiner og tillader kun Secure shell som login-metode. Her er et mere avanceret trick til at hente e-post via en POP3-server sat op til at køre alene på localhost. Først skal du lave en `.fetchmailrc` med følgende indhold.

```
poll localhost with proto pop3 port 1234
    user "tyge" there with password "hemmeligt" is tyge here options \
pass8bits mimedecode limit 100000 fetchall
```

Opsætningen er til at "tyge" på klient og server med adgangskode "hemmeligt" på serveren kan hente e-post op til maksimalt 100000 oktetter (ca. 80 kb per e-post-besked).

Du skal nu lave en kommando eller alias **ssh -L 1234:mailserver:110 -v mailserver**, som du starter i et terminalvindue. Du kan se at du faktisk logger ind på mailservermaskinen ved dette.

I et andet vindue på lokalmaskinen (klienten) skriver du nu **fetchmail**, hvorefter din e-post hentes via POP3 med **ssh** som krypteret tunnel mellem de to maskiner.

Tip: Hvad gør man hvis man har en mail-konto, som skal tømmes men man ikke skal gemme alle de emails? Det kan være relevant hvis man sender emails videre til en webmail-konto. Man skal have en `.fetchmailrc`-fil som gør det muligt at hente post fra POP3-serveren.

```
poll DIN_POP3_MAILSERVER
proto POP3
user DIT_BRUGER_ID
```

```
pass DIN_ADGANGSKODE
```

Derefter kan man køre **fetchmail --bsmtp /dev/null** for at tømme alle emails ud i `/dev/null` dvs. de smides ud.

10.2.3. Sortering og spam-filtrering af e-breve med procmail

Hvis du får mange e-breve ind pr. dag, så vil du nok hurtigt lave forskellige foldere til forskellige emner. F.eks. kan du lave en folder til hver af de diskussionslister på SSLUG, som du deltager i. Et program, som du kan have meget glæde af, er **procmail**, der kan sortere e-breve automatisk ned til de e-post-foldere, du har oprettet. Henter du breve med fetchmail, så kan procmail lave sorteringen ud fra opsætningsfilen `~/.procmailrc`. I eksemplet har vi post liggende i `~/mail`-kataloget, og herunder har vi forskellige filer som er brevfoldere.

```
path=$HOME/bin:/usr/bin:/bin:/usr/local/bin:.
maildir=$HOME/mail      # Her _skal_ procmail kunne aflevere din mail.
default=$MAILDIR/mbox   # Forvalgt folder til af aflevere e-breve i
logfile=$MAILDIR/.from   # Her gemmer vi en log over hvilke e-breve der kom
lockfile=$HOME/.lockmail  # Fil til at lave lås med.

# REJECT: sikkert spam e-post
# Filen ~/.procmailrc.spam indeholder da regexps der
# matches på brevets headere.
# I ~/.procmailrc.spam kan således indsættes linjer
# ^Subject: .*(\$\$|sex|adult|offer)
# som matcher det du vil slippe for. E-post, der
# fanges af reglen gemmes i spam post-mappen
:0
* ? egrep -q -f $HOME/.procmailrc.spam
spam

# Første liste er mgp-listen for Magic Point programmet
# Vi tjekker efter To eller Cc feltet er til listen mgp-users@mew.org
# og gemmer i ~/mail/mgp
:0
* ^(To:|Cc:).*mgp-users@mew.org
mgp

# Dernæst har vi sslug-teknik som altid sætter "Reply-To" feltet
:0
* ^Reply-To:.sslug-teknik@sslug.dk
sslug-teknik

# Alle breve fra brugere @imm.dtu.dk gemmer vi i IMM-folderen
:0
* ^From:.*@imm.dtu.dk
IMM

# Vi kan også gemme en ekstra kopi af breve - dvs. brevet bliver
```

```
# også processeret af de efterfølgende regler.
# Kommer der et e-brev fra statsministeren stm@stm.dk så gemmer vi lige
# en ekstra kopi i stats-folderen
:0c
* ^From:.*stm@stm.dk
stats


# Vi tager og sender en ekstra kopi af emails fra stm@stm.dk
# til redaktionen@ekstrabladet.dk
:0c
* ^From:.*stm@stm.dk
! redaktionen@ekstrabladet.dk

# Endelig kan vi faktisk også udføre kommandoer via mail
# Her lader vi e-breve med emnet EJECT udføre kommandoen eject
# og da det kun er en kopi vi udtager, så vil brevet også gå til
# det forvalgte brevfolder.

:0c
*^Subject:.eject
|/usr/bin/eject

# Alle andre mails ender i default mail folderen.
```

En langt mere restriktiv `.procmailrc`-fil kan være

```
path=$HOME/bin:/usr/bin:/bin:/usr/local/bin:.
maildir=$HOME/mail      # Her _skal_ procmail kunne aflevere din e-post.
default=$MAILDIR/mbox   # Default folder til at aflevere ebreve i
logfile=$MAILDIR/.from   # Her gemmer vi en log over hvilke ebreve der kom
lockfile=$HOME/.lockmail  # Fil til at lave lås med.

:0
* ? egrep -q -f $HOME/.procmailrc.allow
$DEFAULT

:0
spam
```

Alt gemmes i spam-folderen medmindre det er tilladt ved tjek i forhold til filen `~/.procmailrc.allow`. Denne fil indeholder f.eks.

```
^((Resent-)?(To|Cc|Bcc|From)): .*(tyge|root|.*master)@sslug\.dk
^From: freshmeat daemon <freshd@freshmeat\.net>$
^Sender: Security Portal Mailing List
^Mailing-List: contact sslug-.*@sslug.dk
^To: apacheweek@apacheweek\.com$
^Delivered-To: mailing list announce@apache\.org
```

I dette tilfælde er det kun e-post til tyge@sslug.dk (tilsvarende også root og postmaster), og de e-post-lister, som brugeren tyge kender, der ender i `$MAILDIR/mbox` – alt andet ender i spam-folderen. Det kan anbefales at tjekke sin spam-folder fra tid til anden for at være sikker på at e-post ikke er gået galt i procmail-reglerne. Er du helt sikker på dine spam-regler, så kan du sende mail direkte til `/dev/null` i stedet for spam-folderen.

Procmail er guld værd, og **man procmail**, **man procmailex** og **man procmailrc** er værd at læse. Du kan også være glad for at lave et alias **alias ms "mailstat ~/mail/.from"**, idet du så kan skrive **ms** efter at have kørt fetchmail, og du vil så få vist hvor mange breve og størrelsen af disse som er blevet fordelt til de forskellige foldere.

Et smart procmail-filter er følgende, som retter subject af brevet til en advarsel om et potentielt virus-angreb hvis attachment kan indeholde en virus.

```
:0
* ^Content-type: (multipart/mixed|application/octet-stream)
{
    :0 HB
    *  ^Content-Disposition: attachment;
    *filename=.*\.(exe|vbs|shs|com|pif|bat|src|wfs|vbe|wsh|hta)
    {
        :0 fhw
        | formail -i "Subject: Attachments kan indeholde virus !"

        :0:
        $DEFAULT
    }
}
```

Et andet smart trick er følgende spam-filter, som tjekker om en afsender er black-listet i filen `/home/dig/black.lst`. Hvis dette er tilfældet så sendes automatisk en email til afsenderen om at man mener det er spam. Samme trick kan bruges til mere venlige svar til ens venner om at man lige er væk, og at man kan prøve at ringe i stedet eller lign.

```
# Test om afsender er blacklistet. Hvis ja, send et autosvar...

from=`formail -xReturn-Path:`
subj=`formail -xSubject:`

:0
* ?
formail -x"From" -x"From:" -x"Sender:" -x"Reply-To:" -x"Return-Path:" -x"To:
" | egrep -is -f /home/dig/black.lst
| ( \
    echo "To: $FROM";\
    echo 'From: "Spam Filter" <no-spam@ditdomain.dk>';\
    echo "Subject: Spam Detected: $SUBJ"\n;\
    echo 'Dette er en automatisk advarsel !\n';\
    echo 'Hold op med at sende spam!\n';\
  ) | $SENDMAIL -U $FROM
```

Filen `/home/dig/black.lst` udfyldes med en email-adresse per linje, ala.

```
spam@spam.com
unwanted@mostwanted.dk
reklame@jatak.dk
```

Der er under bogens eksempler på www.linuxbog.dk/unix/eksempler/procmail (http://www.linuxbog.dk/unix/eksempler/procmail) et meget stort eksempel på hvordan en `.procmailrc`-fil kan se ud hvis den skal filtrere en masse spam fra og samtidig sortere SSLUG's lister fornuftigt.

10.2.4. spam-filtering med spamassasin

Har du længe haft din e-post-adresse liggende på en web-server et sted, eller har du blot én gang sendt en besked til en offentlig nyhedsgruppe, så har du sikkert også fået en del "uimodståelige" tilbud - eller spam (http://www.ironworks.com/comedy/python/spam.htm) som det rettelig hedder.

Der har været mange løsninger på hvad der kunne gøres ved det. Man kan offentligt hænge (http://www.petemoss.com/spamflames/ShifmanIsAMoronSpammer.html) spammeren ud, skrive til postmesteren ((postmaster) det hjælper sjældent), installere razor (http://razor.sourceforge.net/), installere Realtime Blackhole List (http://mail-abuse.org/rbl/) eller det sidste nye skud på stammen, SpamAssassin (http://spamassassin.org/).

SpamAssassin installeres på en klient-maskine, eller hos en internetudbyder. Her kigger SpamAssassin så den indkommende post igennem og kigger efter mønstre, der tyder på at brevene er spam. Hver ting der testes for i et brev, får så et point. Skulle der optræde et par "$" (dollartegn) i emnelinjen, giver det 2,4 point, et enkelt udråbstegn giver kun 0,1 point. Når der så er opsamlet 5 point, bliver brevet klassificeret som spam. Tærsklen kan selvfølgelig ændres, hvis man er mere eller mindre tolerant end SpamAssassins forfattere.

Har du før brugt procmail (se Afsnit 10.2.3), er SpamAssassin en nem lille ekstra ting at installere. Programpakken der skal installeres fås både som tar.gz (http://spamassassin.taint.org/downloads.html), RPM (http://www.hughes-family.org/spamassassin/) og debian-pakke. Skulle der være afhængigheder af andre RPM-pakker, kan disse findes på rpmfind.net (http://rpmfind.net/). Og **apt-get** skal nok løse sådanne problemer for debian-brugere. I din `~/.procmailrc` tilføjes så blot nedenstående linjer:

```
# Filnavn: ~/.procmailrc
path=/bin:/usr/bin:/usr/bin
maildir=$HOME/mail       #you'd better make sure it exists
logfile=$MAILDIR/log     #recommended
pmdir=$HOME/.procmail

:0fw
| spamassassin

:0:
```

```
* ^X-Spam-Status: Yes
caughtspam
```

Det vil typisk være passende at placere ovenstående i den slutningen af `~/.procmailrc`-filen, hvis du har andre regler i forvejen. SpamAssassin bruger en del CPU-kraft, så det er en fordel først at filtrere lukkede postlisterlister som for eksempel `<sslug-teknik@sslug.dk>` fra:

```
...

:0
*^Delivered-To: mailing list sslug-.*@sslug.dk$
sslug

:0fw
| spamassassin

:0:
* ^X-Spam-Status: Yes
caughtspam
```

SpamAssassin vil nu tage det brev der kommer igennem, og undersøge om der er noget der minder om spam. Hvis det opnår under 5 point, bliver der indsat en linje i brevhovedet om at det ikke er spam:

```
X-Spam-Status: No
```

Hvis der derimod findes lidt "underlige" ting, så ændres emnelinjen til `*****SPAM*****`, der indsættes en forklaring i toppen af brevet med en beskrivelse af hvorfor det blev kategoriseret som spam, og der indsættes en linje i brevhovedet om at det er spam:

```
X-Spam-Status: Yes
```

Og med ovenstående `~/.procmailrc`-fil bliver brevet lagt i folderen `caughtspam`, som man så en gang imellem kan kigge lidt i for at se om der skulle være nogle smuttere. SpamAssassin fanger rundt regnet 98% af den rigtige spam og enkelte bliver ikke fanget, men det er meget få. Her er et grelt eksempel med 16.70 point:

```
From:  Earlene Drake <av853av@erols.com>
To:  pto@sslug.dk
Subject:  To People Who Want To Get The Lowest Intrest Rate But Can't Get Started mosnbd s
Date:  Thu, 24 Jul 03 11:01:07 GMT

This mail is probably spam.  The original message has been attached
along with this report, so you can recognize or block similar unwanted
mail in future.  See http://spamassassin.org/tag/ for more details.

Content preview:  REFINANCE NOW AND SAVE BIG! If you are paying more than
  4.6% on your mortgage, we can save you money! NO COST OR OBLIGATION
  [...]
```

```
Content analysis details:   (16.70 points, 5 required)
NO_COST             (0.9 points)  BODY: No such thing as a free lunch (3)
REFINANCE_NOW       (2.9 points)  BODY: Home refinancing
BANG_MONEY          (1.7 points)  BODY: Talks about money with an exclamation!
HTML_LINK_CLICK_HERE (0.1 points)  BODY: HTML link text says "click here"
HTML_30_40          (0.6 points)  BODY: Message is 30% to 40% HTML
BAYES_80            (2.9 points)  BODY: Bayesian classifier says spam probability is 80 to 9
                    [score: 0.8411]
USERPASS            (1.3 points)  URI: URL contains username and (optional) password
HTTP_USERNAME_USED (0.7 points)  URI: Uses a username in a URL
NUMERIC_HTTP_ADDR  (1.6 points)  URI: Uses a numeric IP address in URL
MISSING_MIMEOLE     (0.1 points)  Message has X-MSMail-Priority, but no X-MimeOLE
CLICK_BELOW         (0.0 points)  Asks you to click below
MIME_HTML_ONLY      (0.1 points)  Message only has text/html MIME parts
UPPERCASE_25_50     (1.0 points)  message body is 25-50% uppercase
FORGED_MUA_OIMO     (2.8 points)  Forged mail pretending to be from MS Outlook IMO

The original message did not contain plain text, and may be unsafe to
open with some email clients; in particular, it may contain a virus,
or confirm that your address can receive spam.  If you wish to view
it, it may be safer to save it to a file and open it with an editor.
```

Resten af ovenstående mail er klippet væk, da du sikkert ikke er interesseret i selve indholdet.

10.2.5. Kryptering af post

I takt med internettets udbredelse er der kommet flere og flere mindre venligsindede mennesker på nettet. Elektronisk post er desværre meget let at aflytte, og ønsker du brev-hemmelighed, må du kryptere din post. Kryptering er en proces hvor dine data bliver skrevet i kode. Krypterede breve kan kun læses af den person, de er stilet til.

I Linux-verdenen finder der flere programmer til kryptering af post. Heldigvis findes der en standard for hvordan post bør krypteres, og det er derfor ligegyldigt hvilket krypteringsprogram, du benytter. Vi har valgt at introducere GNU Privacy Guard (GPG), idet GPG kommer under en ægte fri licens (GNU General Public License). Samtidig kan GPG arbejde sammen med det anerkendte Pretty Good Privacy (PGP).

GPG kan også bruges til at signere breve. En digital signatur er mere end en underskrift. Den digitale signatur fortæller, ud over hvem der er afsender, også hvornår brevet blev signeret. Endvidere bruges den digitale signatur til at tjekke om brevet er blevet ændret under dets rejse fra afsender til modtager. I skrivende stund kender forfatterne ikke det officielle Danmarks holdning til GPG.

Ideen bag GPG er, at du har en offentlig og en privat nøgle. Den offentlige nøgle gør det muligt for alle at tjekke din signatur, som du fremstiller vha. din private nøgle. Når du skal kryptere post, skal du først udveksle nøglepar med den person, som du ønsker at kommunikere med. Udvekslingen sker ofte i forbindelse med møder i Linux-brugerforeninger.

Du kan hente GPG fra GPG's ftp-server ftp://ftp.gnupg.org. Selve installationen er lige til at gå til – nedenfor ser du hvordan.

```
[tyge@hven ~]$ wget ftp://ftp.gnupg.org/pub/gcrypt/gnupg/gnupg-1.0.1.tar.gz
[tyge@hven ~]$ tar xzvf gnupg-1.0.1.tar.gz
[tyge@hven ~]$ cd gnupg-1.0.1
[tyge@hven gnupg-1.0.1]$ ./configure
[en masse klippet væk]
[tyge@hven gnupg-1.0.1]$ make
[en masse klippet væk]
[tyge@hven gnupg-1.0.1]$ su
Password: hemlig
[root@hven gnupg-1.0.1]$ make install
[root@hven gnupg-1.0.1]$ exit
[tyge@hven gnupg-1.0.1]$ cd ..
[tyge@hven ~]$ rm -rf gnupg-1.0.1
```

Efter installationen er du klar til at bruge GPG. Det første, du skal gøre, er at generere et nøglepar.

```
[tyge@hven ~]$ gpg
gpg: Warning: using insecure memory!
gpg: /home/tyge/.gnupg: directory created
gpg: /home/tyge/.gnupg/options: new options file created
gpg: you have to start GnuPG again, so it can read the new options file
[tyge@hven ~]$  gpg --gen-key
gpg (GnuPG) 1.0.1; Copyright (C) 1999 Free Software Foundation, Inc.
This program comes with ABSOLUTELY NO WARRANTY.
This is free software, and you are welcome to redistribute it
under certain conditions. See the file COPYING for details.

gpg: Warning: using insecure memory!
gpg: /home/tyge/.gnupg/secring.gpg: keyring created
gpg: /home/tyge/.gnupg/pubring.gpg: keyring created
Please select what kind of key you want:
   (1) DSA and ElGamal (default)
   (2) DSA (sign only)
   (4) ElGamal (sign and encrypt)
Your selection? 1
DSA keypair will have 1024 bits.
About to generate a new ELG-E keypair.
              minimum keysize is  768 bits
              default keysize is 1024 bits
    highest suggested keysize is 2048 bits
What keysize do you want? (1024) 1024
Requested keysize is 1024 bits
Please specify how long the key should be valid.
         0 = key does not expire
      <n>  = key expires in n days
      <n>w = key expires in n weeks
      <n>m = key expires in n months
      <n>y = key expires in n years
Key is valid for? (0) 0
```

```
Key does not expire at all
Is this correct (y/n)? y
You need a User-ID to identify your key; the software constructs the user id
from Real Name, Comment and Email Address in this form:
    "Heinrich Heine (Der Dichter) <heinrichh@duesseldorf.de>"

Real name: Tyge
Email address: tyge@sslug.dk
Comment:
You selected this USER-ID:
    "Tyge Brahe <tyge@sslug.dk>"

Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? O
You need a Passphrase to protect your secret key.

Enter passphrase: DetteErHemlig!
Repeat passphrase: DetteErHemlig!

We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
+++++++++++++++..+++++++++++++++.++++++++++++++++++++++++++++++++++++++
+++++++++..++++++++++.++++++++++++++++++++++++++++++++++++++>++++++++++>
+++++.....................................+++++
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
++++++++++.+++++++++++++++.++++++++++....++++++++++.+++++++++++++++.+++++++
+++.++++++++++++++++++++++++++++++++++++++++++++.+++++++++++++++>+++++....
.........+++++^^^
public and secret key created and signed
```

Du bør med det samme fremstille en nøgle, som kan bruges til at tilbagekalde den, du netop har fremstillet. Det er vigtig, idet det kan ske at en eller anden finder ud af din private nøgle. En tilbagekaldelsesnøgle genereres vha. kommandoen **gpg --gen-revoke tyge@sslug.dk** (vælg option 1 og gem hvad du ser på skærmen i en fil). Tilbagekaldelsesnøglen (gemt i en fil) bør opbevares et andet sted end dit hjemmekatalog, f.eks. kan du have den liggende på en diskette i dit pengeskab.

Når du nu har fået genereret nøgler, er du klar til at signere breve. Til at begynde med vil vi antage, at du skriver brevet først, signerer det og til sidst indlæser det signerede brev i dit postprogram. Nedenstående kommandosekvens viser dig hvordan.

```
[tyge@hven ~]$ cat brev
Kære Otto,

Du kan tro, at vi har det sjovt med GNU!

Tyge
[tyge@hven ~]$ gpg --clearsign brev
```

```
[tyge@hven ~]$ cat brev.asc

-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA1

Kære Otto,

Du kan tror at vi har det sjovt ned GNU!

Tyge
-----BEGIN PGP SIGNATURE-----
Version: GnuPG v1.0.1 (GNU/Linux)
Comment: For info see http://www.gnupg.org

iD8DBQE4elQW60LvO+QONowRAntQAKCIvUMIsyf1US2JoHC/VKgD73Ps1gCfRvRM
H9uEQNpfPp+afnkp86snovo=
=6+k8
-----END PGP SIGNATURE---
```

Den opmærksomme læser bemærker at det signerede brev ligger i filen `brev.asc`, dvs. der er blevet tilføjet `.asc` til det oprindelige navn.

For at få det største udbytte af GPG bør du læse den glimrende brugervejledning. Den finder du på http://www.gnupg.org/docs.html.

Skal du benytte pine sammen med GPG, så læs http://www.linuxsecurity.com/feature_stories/feature_story-83.html. Vær opmærksom på om **pinegpg-install** rent faktisk indsætter de rigtige linjer i `~/.pinerc` – i forhold til hvor dine GPG-programmer ligger. Måske skal man editere `/bin/filnavn` til `/usr/bin/filnavn`.

10.2.6. Hent e-post fra en Exchange-server

Microsoft har været så venlige at lave en nyere type e-post-server, MS Exchange, som uheldigvis ikke er dokumenteret for godt. Heldigvis kan man bruge serveren fra f.eks. Netscapes e-post-funktion. I Netscape skal du under "preferences"->"mail" angive at du henter e-breve fra en "IMAP" server – ikke POP – og sætte serveren til at være exchange-serveren. Det er ikke den fulde funktionalitet, du får, men det kan bruges.

10.2.7. Konvertering fra Outlook til Linux-e-post

En metode til at give Linux-brugeren adgang til e-post som ligger i Outlook koblet til en Exchange-server er følgende. Det kræver dog en Exchange-server med imap-understøttelse:

- (I Windows) kopiér indholdet af Outlook-folder (.pst filen) over på Exchange-serveren.

- opret en ny IMAP mail-server i Netscape og konfigurér den til at bruge din Exchange-server. Tjek: Indholdet af Outlook-folderen burde nu være tilgængeligt fra Netscape Messenger.
- kopiér foldere fra Exchange-serveren til local mail.

En anden måde er at benytte KDEs program **kmailcvt**, som kan oversætte dine gamle Outlook 4 og 5 postbokse til Unix-postfiler. Efter sigende skulle dette være uden problemer, dog kan adressebøger stadig give problemer.

10.2.7.1. Outlook Express til mbox konvertering

En tredie metode, er at konvertere din *.pst-fil til *.mbx (eller *.dbx) format, som Outlook Express bruger. Derefter bruges **oe2mbx** til at konvertere til mbox-format, der kan læses af de fleste post-programmer på Linux-boksen. Du skal hente Linux-versionen af oe2mbx på adressen: http://ftp.sslug.dk/oe2mbx-1.21.tar.gz. Den officielle side (http://www.micropop.com/code/) er umiddelbart lukket, men prøv at søge på http://google.com (http://www.google.com).

Det skal lige siges, at under konvertering fra MS Exchange til MS Outlook Express mister man alle "mail headers", dvs. at information om hvordan e-posten kom til din maskine går tabt. Dette kan være en Exchange-bug, eller der kan være overset en opsætningsmulighed.

Før vi går i gang, så afsæt tiden til dette projekt – det tager mange timer at komme igennem. Det kan også anbefales at have ca. 3 gange så meget ledig diskplads som den post-mængde man har i Outlook Express.

Fra Exchange til Outlook Express.

Konvertering fra Exchange kræver at du går via MS Outlook Express. Hvis du ikke bruger Exchange kan du hoppe dette skridt over. Du kan importere din post direkte fra Exchange til Outlook Express via en Wizard. Start Outlook Express, vælg "File"->"Import"-> "Microsoft Exchange". Derefter vælger du Profil og siger Næste - venter – OK.

Hvis du har lavet en `.pst`-fil i Exchange til offline-læsning kan du skabe en ny profil, når du kommer til det punkt. Du skal så vælge manuel opsætning og tilføj `*.pst` fil. (Dette kan gøres uden kontakt til Exchange-serveren). Resten er som beskrevet ovenfor.

Fra Outlook Express til Linux mbox

Windows: Læg **oe2mbx.exe** i roden (c:\) og kopiér eller flyt alle `*.dbx`-filer til mappen `c:\mail`. Åben en MS-DOS-prompt og skriv: **c:\oe2mbx c:\mail*.dbx**. Voila, du har nu én eller flere mbox-filer, klar til at kopieres over på din Linux-maskine.

Linux: Start med at kopiere alle `.dbx`-filer til `$HOME/mail/`.

```
[tyge@hven ~]$ tar xzvf oe2mbx-1.21.tar.gz
...
[tyge@hven ~]$ cd oe2mbx-1.21
[tyge@hven ~/oe2mbx-1.21]$ make
...
[tyge@hven ~/oe2mbx-1.21]$ ./bin/oe2mbx ~/mail/*.dbx
```

Fra mbox til maildir (qmail) Hvis du bruger qmail og maildir kan du til slut hente et program (perl-script) som hedder **mbox2maildir** fra http://www.qmail.org/mbox2maildir.

Tak til: "Outlook -> Netscape"-tråden på sslug-teknik den 21. dec. 2000. Se: http://www.sslug.dk/emailarkiv/teknik/2000_12/msg00776.html og http://www.sslug.dk/emailarkiv/teknik/2000_12/msg00779.html.

10.3. Internet-sikkerhed

Da man designede internettet for mange år siden, fandtes crackere ikke, og man havde ikke fantasi til at forestille sig nutidens avancerede internet-kriminalitet. Fakta er, at det meste internet-kommunikation sendes i klar tekst – inklusiv login-navne og tilhørende adgangskoder. Derfor skal man passe på, hvis man skal lave login på Linux-maskinen fra et fjernt sted i verden via internettet. Der er sikre systemer med stor fleksibilitet, som ikke transmitterer adgangskoder i klar tekst. Den bedste implementering er "Secure Shell" (ssh).

Emnet er meget vanskeligt, idet de kriminelle ofte ved uhyre meget om netværk og sikkerhedsfejl. Der er dog et par generelle kommentarer, som bør følges nøje, hvis din maskine skal stå på et usikkert netværk.

- Følg med i, hvilke programmer der kommer med opdateringer. Se f.eks. http://www.cert.dk.
- Luk af for programmer, du ikke bruger. Bliver det kendt, at der er en sikkerhedsbrist i bare ét af de kørende programmer, kan du rammes. I store træk, så indsæt `disable = yes` i samtlige filer i `/etc/xinetd.d/` og kør:

  ```
  [tyge@hven ~]#  su -
  [root@hven /root]#  killall -HUP xinetd
  [root@hven /root]#  /etc/rc.d/init.d/portmap stop
  [root@hven /root]#  /sbin/chkconfig --level 2345 portmap off
  ```

  Med "portmap" stoppet (og ikke startet sidenhen) får du ikke ubrugte stand-alone dæmoner, der kan ramme dig sidenhen. Åbn kun for services som du ved du skal bruge. Lad være med at køre en web-server eller X-server, hvis du ikke bruger det.

- Installér overvågningssystemer (såsom Tripwire og LogWatch).
- Installér Secure Shell eller OpenSSH (Se http://www.openssh.com), og lad dette erstatte **telnet** og **ftp**.
- Installér og konfigurér en brandmur (eng. firewall) (ipchains). Det er en del af Linux-kernen og dermed gratis. Der er gode HOWTO-dokumenter om dette.

- Lær mere om netværk og Linux, og brug dit system med omtanke!

Listen ser dramatisk ud og er ikke så relevant for dem, som lige logger på internettet via modem for at tjekke epost og surfe lidt. For dem, som vil lave missionskritiske forretningssystemer på internettet, er det *ekstremt* relevant. Du kan læse meget mere i bogen "Linux – Friheden til sikkerhed på internettet", der findes på www.linuxbog.dk/ (http://www.linuxbog.dk/).

Tilsvarende bør du læse http://www.sslug.dk/sikkerhed/.

Det kan anbefales at købe en god bog om netværkssikkerhed, såsom:

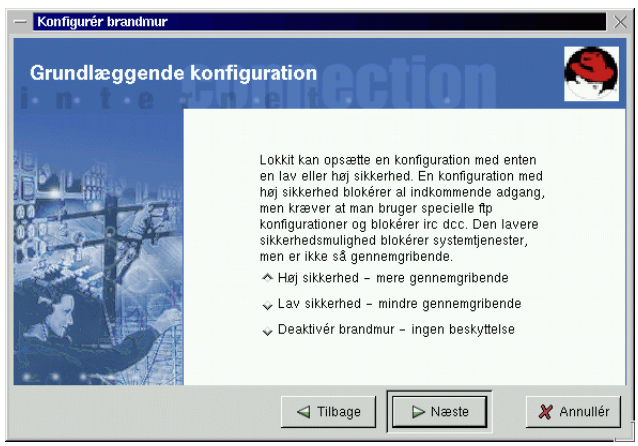
- Simon Garfinkel and Gene Spafford: *Practical Unix & Internet Security*, O'Reilly & Associates, Inc., ISBN 1-56592-148-8, 971 sider.

10.4. Nem brandmur med Red Hat

Er din maskine på internettet i lang tid ad gangen, så bør du have en brandmur på din maskine. Vil du vide mere, så læs "Linux – Friheden til sikkerhed på internettet".

I Red Hat (7.0 og senere) er der et godt program med til at sætte en brandmur op. Der er ingen garanti for at alt bliver sikkert bagefter, men modsat, så er det et glimrende udgangspunkt. Program-pakken hedder `gnome-lokkit*.rpm`. Du skal som root køre **/usr/sbin/gnome-lokkit** for at sætte brandmuren op eller **lokkit** hvis det skal være i tekstmodus. Vælg "høj sikkerhed". Dernæst om hvilke maskiner man skal tillade lav sikkerhed overfor (evt. til internt netværk). Du spørges om følgende skal tillades: DHCP (vælg ofte nej), ekstern adgang til egen webserver (normalt nej), indgående e-post (ofte nej), ssh (ofte nej), telnet (altid nej). Dernæst laver programmet hele brandmuropsætningen og den aktiveres automatisk ved næste systemopstart. Du kan også selv genstarte brandmuren ved at køre **/etc/init.d/ipchains restart**.

Figur 10-2. Firewall med gnome-lokkit



10.5. Nyhedslæsere/NNTP-klienter

Vi prøver her at give en introduktion til en række tekstbaserede nyhedslæsere (eng. »news readers), altså programmer til at læse nyhedsgrupper – både på Usenet og organisationer som SSLUG's egne nyhedsgrupper.

10.5.1. Pine

... lidt om Pine som nyhedslæser ...

10.6. FAX under Linux

Der findes en del programmer til at sende og modtage FAX under Linux. Se mere på org/pub/Linux/apps/serialcomm/fax/ (http://www.ibiblio.org/pub/Linux/apps/serialcomm/fax/).

10.7. Bøger om netværk

Netværk er et stort og kompliceret emne, som det ikke er muligt at dække på andet end den mest nødtørftige måde på den plads, der er brugt her. Hvis du har lyst til at lære mere, kan nedenstående bøger anbefales.

- Olaf Kirch: *Linux Network Administrator's Guide*, O'Reilly & Associates, Inc., ISBN 1-56592-087-2, 335 sider.

  Dette er en indføring og praktisk vejledning i TCP/IP, UUCP, DNS, PPP, mail, news osv., altså stort set alt, hvad der har at gøre med at køre netværk under Linux. Det forudsættes, at man er i stand til at redigere en tekstfil og generelt "komme omkring" i systemet.

  Da bogen oprindeligt er udsprunget af The Linux Documentation Project, kan den hentes frit på internettet, f.eks. på ftp://metalab.unc.edu/pub/Linux/docs/LDP/network-guide. Ganske ofte er filen på Linux-cd-rom'erne.

- Craig Hunt: *TCP/IP Network Administration*, O'Reilly & Associates, Inc., ISBN: 0-937175-82-X, 472 sider.

Denne bog handler om TCP/IP under Unix generelt i modsætning til Linux Network Administrator's Guide og går nok også lidt mere i dybden med tingene.

- Stig Jensen og Arne Gjelstrup: *Datakommunikation*, Teknisk Forlag, ISBN 87-571-1956-2, 872 sider.

  Hvis du vil have en endnu dybere og mere teknisk betonet viden om netværk generelt, er denne bog et glimrende valg og fungerer godt både som lærebog og opslagsværk.

  Det kan i øvrigt være interessant at følge http://users.dhp.com/~whisper/mason/ og http://www.opensource.firewall.com/.

Appendiks A. Oversigt over de vigtigste Unix-kommandoer

Oversigt over de mest almindelige Unix kommandoer.

Tabel A-1. Brugsanvisninger

man	Hjælp til enkelte kommandoer
info	Hjælp til enkelte kommandoer
apropos	Vis relevante komandoer

Tabel A-2. Filhåndteringer

mv	Flyt og/eller omdøb filer
cp	Kopiér filer og kataloger
ln	Opret henvisning til fil eller katalog
mkdir	Opret tomt katalog
rmdir	Slet tomt katalog
rm	Slet filer og/eller kataloger

Tabel A-3. Filsøgning

ls	Vis filer og kataloger
pwd	Vis nuværende katalog
find	Søg efter filer og kataloger
locate	Søg efter filer og kataloger – i database
grep	Søg efter tekst i filer og/eller strøm
df	Vis pladsforbrug på filsystemer
du	Vis pladsforbrug i kataloger
which	Vis hvor et program ligger (den udgave der vil blive kørt)
which	Vis hvor et program ligger (alle udgaver i kommandostien)

Tabel A-4. Rettighedsstyring

su	Skift identitet (bruger)
newgrp	Skift identitet (gruppe)
chown	Ændr ejerskab af filer og kataloger

chmod	Ændr adgangsrettigheder til filer og kataloger
w	Vis aktive brugere
who	Vis identitet
passwd	Skift adgangskode

Tabel A-5. Filvisning

echo	Udskriv til skærm
cat	Udskriv filer og/eller strøm
less	Vis indhold filer og/eller strøm (kan bladre)
more	Vis indhold filer og/eller strøm (kan bladre)
head	Vis starten af filer og/eller strøm
tail	Vis slutningen af filer og/eller strøm

Tabel A-6. Tekstmodifikationsprogrammer

cmp	Vis forskelle mellem to filer
diff	Vis forskelle mellem to filer
comm	Vis forskelle mellem to filer
patch	Omgør forskelle
cut	Udvælg søjler
wc	Tæl tegn, ord og linjer
tr	Søg og erstat på bogstavniveau
sdiff	Vis forskel og lav fælles sum af filerne

Tabel A-7. Andre programmer

sh	Kommandofortolker
bash	Kommandofortolker
csh	Kommandofortolker
ksh	Kommandofortolker
tcsh	Kommandofortolker
zsh	Kommandofortolker
date	Vis/sæt dato og klokkeslæt
at	Kør program på et bestemt senere tidspunkt
batch	Kør program på et senere tidspunkt
mount	Montér filsystem
umount	Afmontér filsystem
cal	Vis kalender

clear	Slet skærmbillede
mc	Norton Commander-klon

Tabel A-8. Processtyring

kill	Stop/afliv proces
ps	Vis processer
top	Vis processer interaktivt
nice	Justér procesprioritet
shutdown	Luk systemet ned
init	Skift kørselsniveau
reboot	Genstart systemet
exit	Afslut kommandofortolker
last	Vis login-historie

Tabel A-9. Overblik (Overvågning)

vmstat 10	Overblik over proces-kø, swap, IO mv. hvert 10 sekund
netstat -apn	Vis alle processer som bruger net
iostat 4	IO hvert 4 sek. [a] (installer sysstat)
mpstat	Vis processor-statistik
swapon -s	Vis brug af swap og kan aktivere swapfiler.
Noter:: a. findes ikke på alle distroer	

Tabel A-10. Netværk

ifconfig	Vis netværksenheder
ping	Test forbindelsen til en anden maskine
traceroute	Vis ruten til en anden maskine

Tabel A-11. Klienter

ssh	SSH-klient (sikker fjern-login)
scp	SSH-klient (sikker filoverførsel)
telnet	TELNET-klient (*usikker* fjern-login)
ftp	FTP-klient (*usikker* filoverførsel)
rsync	rsync-klient (filoverførsel)
rcp	rcp-klient (*usikker* filoverførsel)

talk	Snak med en bruger
write	Skriv besked til en bruger
lynx	Browser og HTML-filter
wget	HTTP-klient

Tabel A-12. Udskrift

a2ps	Oversæt tekst til Postscript
lpr	Udskriv filer og/eller strøm
lpq	Vis printerkø
lprm	Afbestil udskrift

Appendiks B. Hvilke kommandoer kommer i hvilke pakker

Denne oversigt er beregnet til at hjælpe dig, hvis der mangler en kommando fra et af bogens eksempler på det system du bruger. Her kan du slå op, hvilken pakke du skal bede din systemadministrator om at installere, for at få adgang til en specifik kommando.

Tabel B-1. Hvilke kommandoer kommer i hvilke pakker

Kommando	Debian	Gentoo	Mandrake	Red Hat	SuSE
apropos	(Debian)	(Gentoo)	(Mandrake)	man	(SuSE)
find	(Debian)	(Gentoo)	(Mandrake)	findutils	(SuSE)
info	(Debian)	(Gentoo)	(Mandrake)	info	(SuSE)
man	(Debian)	(Gentoo)	(Mandrake)	man	(SuSE)
man2html	(Debian)	(Gentoo)	(Mandrake)	man	(SuSE)
perl	(Debian)	(Gentoo)	(Mandrake)	perl	(SuSE)
sed	(Debian)	(Gentoo)	(Mandrake)	sed	(SuSE)
vi	(Debian)	(Gentoo)	(Mandrake)	vim-minimal	(SuSE)
xargs	(Debian)	(Gentoo)	(Mandrake)	findutils	(SuSE)

Appendiks C. Revisionshistorie for bogen

Igennem tiden har bogen "Linux – friheden til vælge Unix" udviklet sig meget. Vi frigiver ofte nye versioner, når der er kommer en del rettelser ind, eller nye afsnit er blevet skrevet. Kommentarer, ris og ros, og specielt fejl og mangler bedes sendt til linuxbog@sslug.dk (mailto:linuxbog@sslug.dk), men er du medlem af SSLUG, så skriv til sslug-bog@sslug.dk (mailto:sslug-bog@sslug.dk). Alle kan bidrage – se om tilmelding se på http://www.sslug.dk/tilmeld.

Her er en liste over, hvad der er ændret i bogen.

- Version 1.8.20060212 - 12. februar 2006: Donald Axel: Kommandofortolkerafsnit, korrektion og sproglig klarhed efter forslag fra Morten Lerstrup Pedersen (ruc.dk).
- Version 1.8.20040716 - 16. juli 2004: Jacob Sparre Andersen: Udvider kommandofortolkerafsnittet en smule. Flytter afsnit rundt for at skabe en mere logisk struktur. Retter sprog og opmærkning. Omrokerer eksemplet med et simpelt kommandofortolkerprogram. Skrevet mere om kommandofortolkere. Mikkel Kirkgaard Nielsen: Fanger en fejl i et alias.
- Version 1.8 - 25. januar 2004: Peter Toft tilføjer afsnit om xargs. Jacob Sparre Andersen: Retter sprog. Flytter rundt på ting og fjerner afsnit der egentlig hører til de nogle af de andre bøger. Tilføjer en oversigt over hvilke kommandoer kommer fra hvilke pakker. Mads Gige: retter sprog.
- Version 1.7 - 7. oktober 2003: Donald Axel har ændret introduktion til man og info systemet, så man hurtigere får at vide, hvordan man kommer ud af systemet :-) Lidt mere information om less som man-page browser. Jacob Sparre Andersen har tilføjet et kapitel om adgangsstyring på Unix og rettet lidt sproglige fejl. Poul-Erik Hansen og Peter Toft retter tekst om spamassassin. Rasmus Terkelsen fanger et par sproglige fejl. Peter Toft skriver videre og fletter gamle tekster ind. Jacob Sparre Andersen har, udfra forslag fra Anders Melchiorsen og Christian Treldal, skrevet lidt ekstra om skrivetilladelser til kataloger. Hans Schou har tilføjet bash kommandoen read.
- Version 1.6 - 6. april 2003: Peter Toft, skriver om wget og md5sum. Poul-Erik Hansen laver ekstra stikord og appendix med kommandooversigt. Frank Nørvig retter lidt om blacklist i procmail. Preben Mikael Bohn fanger en fejl i henvisning. Jacob Sparre Andersen bruger "Filesystem Hierarchy Standard" til at opdatere tabellen med hvor hvilke ting ligger på et unix-system.
- Version 1.5 - 1. december 2002: Mads Sejersen: Fanget en fejl i Fetchmail-afsnittet. Peter Toft: Har tilføjet et smart lille filter til procmail-afsnittet lavet af Frank Nørvig. Tilføjet information om pine og emails kodet i UTF-8 tegnsættet. Hans Schou: Tilføjet noget om wavelan. Jacob Sparre Andersen: Om installation af ‘yudit‘ på Red Hat 7.2 og 7.3. Om flere kommandofortolkere i samme Konsole-vindue.
- Version 1.4 - 1. september 2002: Hans Schou har tilføjet flere eksempler til regulære udtryk og rettet et par steder fra <screen> til <programlisting>. Og noget mere om ; og & og && og () og <(). Hans har også skrevet en artikel om spamassasin til spam-filtering som er gengivet her. Peter Toft: Har flyttet afsnit om procmail fra admin-bogen her og tilføjet et stærkt eksempel på en .procmailrc-fil med spam-filtrering. Meget input fra sslug-bog@sslug.dk. Har skrevet nyt kapitel om kommandofortolkere - Kapitel 1. Har tilføjet mere tekst under afsnittet om ps. Har flyttet afsnit om mus med hjul og Emacs til afsnittet om Emacs. Nævner lige occur i emacs. Nyt om knode. Flere stikord. Jesper Norskov Kristensen fandt en lille fejl. Jacob Sparre Andersen - Harmoniseret brugen af **tar**. Udbedret logiske fejl i SGML-koden. Rettet sproglige småfejl. Forbedret **pine-spell**, så det kan bruge **dialog** til at

spørge hvilken ordbog skal bruges. Tilføjet eksempler på opsætningsfiler til Zsh. Rettet op på kapitlet om kommandofortolkere.

- Version 1.3 - 14. juni 2002: Peter Toft: Lille tilføjelse med aspell og pine. Har tilføjet nyt afsnit om roller i Pine. Anders Pedersen: Trykfejl fundet. Jacob Sparre Andersen: Ny version af pine-spell. Rettet sproglige småfejl. Henrik Skov Midtiby har rettet en række sproglige fejl.
- Version 1.2 - 10. marts 2002: Ny licens for bogen - Åben dokumentlicens. Peter Toft: Nyt afsnit om patch og mere om diff. Mere om fetchmail. Mere om SGUI - chmod +s på kataloger. Svend Erik Venstrup: Mere om rm af underlige filer med # i navnet og sproglige rettelser. Peter Toft: Net-bank oversigt flyttet til itplatform-bogen.
- Version 1.1 - 29. december 2001: Jacob Sparre Andersen: Sproglige smårettelser. Jesper Krogh - Aars bank ud af hjemmebank-listen. Peter Toft: Mere om smbmount, w, who og finger. Anders Kristiansen: Opdatering af status for ADSL hos Orange. Magnus Østergaard: Gode links til VI og trykfejl rettet. Tilføjelse om rwho.
- Version 1.0 - 21. oktober 2001: Første version: Bogen er splittet fra "Linux - Friheden til at vælge". Resten af den bog kan læses i "Linux - Friheden til at installere" Peter Toft: Lidt om valg af tidszone. Lidt om FAX under Linux. Skrevet mere om terminal-vinduet. Links til computer-ordbøger på nettet. Tilføjet link til Red Hat-variant med ReiserFS-understøttelse. Kim Schulz har tilføjet et afsnit om Jyske Netbank, til afsnittet om hjemmebank. Afsnit om editorer kommer fra "Linux - friheden til at vælge programmer", til denne bog. Rasmus Ory Nielsen har fundet en fejl i .emacs-filen, så den virker nu. Afsnit om anvendelse af GPG til krypteret epost, skrevet af Kenneth Geisshirt er flyttet her til bogen.

Ordliste

Linux Standard Base

Linux Standard Base (http://www.linuxbase.org/) er et forsøg på at lave en fælles specifikation for hvordan en linux-distribution skal se ud, primært set fra en uafhængig softwareleverandørs synsvinkel.

Formålet er at gøre det lettere for uafhængige softwareleverandører at pakke programmer, så de fungerer på mange forskellige linux-distributioner.

Stikordsregister

Symboler

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X

Z