supermax

UNIX Udvidet

Version juni 1991

Indholdsfortegnelse

1 UNIX-repetition

1.1 Begreber

Hjemmekatalog - det katalog, man befinder sig i efter indlogning

Arbejdskatalog - det katalog, man aktuelt befinder sig i

Filnavne
- max. 14 tegn
- undgå specialtegn og æ, ø og å
- *, ? og [] giver mulighed for nemt at referere til flere filer

Fuld path - angivet fra roden. Filnavn starter med /

Relativ path - filnavn angivet i forhold til arbejdskataloget

Filtype
- ordinære filer med programmer, tekst osv
- kataloger
- special filer for ydre enheder

Link - synonym for en fil

I-node - indeks, der refererer til oplysninger om filen

Rettigheder - adgangskoder (r, w, x), der sættes for ejer, gruppe og andre

1.2 Kommandoer

cal	Udskriver kalender
cat	Udskriver indholdet af én eller flere filer
	Sammenkæder filer
cd	Ændrer arbejdskatalog
chgrp	Ændrer gruppe for en fil
chmod	Ændrer rettigheder
chown	Ændrer ejer for en fil
cp	Kopierer én eller flere filer
date	Udskriver systemets dato
echo	Gentager efterfølgende tekst
export	Eksporterer shell-variable
file	Bestemmer filtype
find	Finder en eller flere filer
fuser	Afbryder alle processerne på en terminal
grep	Søger efter mønster i fil
kill	Afbryder proces
ln	Linker filer
ls	Lister katalog
mkdir	Opretter katalog
mv	Omdøber/flytter fil
mvdir	Omdøber/flytter katalog
nohup	Bevirker at en baggrundsproces ikke dør ved udlogning
passwd	Ændrer password
pg	Udskriver indholdet af én eller flere filer
ps	Oversigt over processer
pwd	Viser arbejdskatalog
rm	Sletter én en eller flere filer
rmdir	Sletter ét eller flere kataloger
set	Udskriver brugers shell-variable
sleep	Holder udførelsen af en proces
tty	Udskriver terminalnummer
tee	Sender uddata til terminal og fil
wc	Beregner antal ord, tegn og linier i en fil
who	Udskriver indloggede brugere
whodo	Udskriver indloggede brugere

1.3 Standard Unix kontrolsekvenser

Funktion	Tryk
Afbryd program eller kommando	<Ctrl+c>
Stop udskrift til skærm	<Ctrl+s>
Fortsæt udskrift	<Ctrl+s>
Ud af en shell	<Ctrl+d>

<Ctrl+s> og <Ctrl+q> kan på nogle terminaler klares ved at benytte en speciel tast. På dde450- og dde420-terminaler, hedder denne tast **<Hold Screen>**.

2 Egne shell-scripts

Shellscripts er et særdeles nyttigt værktøj for en systemadministrator, der ønsker at lette sit eget arbejde.

Placeringen af de scripts, man selv fremstiller, afhænger af hvem, der skal benytte dem. Skal de være tilgængelige for alle brugere, kan de lægges under **/bin**, **/usr/bin** eller et nyt katalog: **/usr/bin/-local**. Er det scripts, der kun skal benyttes ved systemadministrativt arbejde, kan de placeres i kataloget **/etc**, eller indgå som menupunkter i **sysadm**-pakken.

Hvis man vil undgå at blande egne udviklede scripts sammen med standard programmellet, kan man oprette et specielt katalog, **/usr/-local/bin**, og lade dette katalog indgå i deres PATH. Tilsvarende kan der oprettes et **/usr/local/etc**-katalog til ens systemadministrative scripts.

2.1 Sysadm-pakken

Opretter man scripts, der indgår som en del af den daglige drift, bør man udføre det arbejde, der skal til for at få dem lagt under **sysadm**-pakken, så man forholdsvis hurtigt vil kunne sætte en afløser/ferie-passer ind i, hvordan scriptene afvikles.

Man kan oprette et katalog (**/usr/admin/menu/lokal**), hvor man placerer sine scripts, og en beskrivelse af disse, og man vil så kunne vælge disse som et hvilket som helst andet menupunkt i sysadm-menuen.

Opgave 1

Formål: - at oprette brugere på maskinerne

1. **Log ind som bin. (Der skulle ikke være kodeord).**

2. **Opret dig selv som almindelig bruger ved hjælp af sysadm. Du skal have hjemmekatalog i /bruger1/<login>, og din shell skal naturligvis være /bin/dsh. Du skal benytte det bruger-id som du får tildelt af instruktøren.**

3 Behandling af Parametre

I shellscripts kan man skelne mellem to forskellige typer af parametre: variable og positionelle parametre.

De positionelle parametre tildeles deres værdi i kaldet af scriptet:

```
$ script1 anders
```

Ved dette kald er **anders** en positionel parameter til **script1.**

Variable indlæses interaktivt i selve scriptet med en read-kommando.

3.1 Positionelle parametre.

De positionelle parametre refereres i shellscripts med betegnelserne **$1, $2, $3,.. $n**, hvor nummeret refererer til den rækkefølge, de angives med i kaldet. Således vil **$1** få tilknyttet værdien **anders** i det ovenstående script. Parameteren **$0** henviser til scriptets navn, og **$#** er antallet af positionelle parametre.

```
$ script1 anders lise per

Antal : 3
Navn  : script1
Par1  : anders
Par2  : lise
Par3  : per

$ cat script1

echo "Antal : " $#
echo "Navn  : " $0
echo "Par1  : " $1
echo "Par2  : " $2
echo "Par3  : " $3
```

3.2 Variable.

En variabel, **var**, indlæses fra et shellscript med kommandoen:

read var

Her tildeles variablen **var** den værdi, der angives under udførelsen af scriptet.

Herefter kan man henvise til variablen ved at skrive **$var**.

```
$ script2

Indtast navn: Andersen
Goddag Andersen


$ cat script2

echo "Indtast navn:\c"
read Navn
echo "Goddag "$Navn
```

"\c" bevirker, at der ikke skal skiftes linie efter *Indtast navn:*

3.3 Standardværdier for parametre

Ofte kan det være nyttigt at benytte standardværdier for parametre, således at de sættes til en bestemt værdi, hvis de ikke ændres. Shell åbner mulighed for følgende metoder til at substituere parametre:

${<parameter>:-<ord>}
Hvis <parameter> er sat, får udtrykket også denne værdi, ellers er den <ord>.

${<parameter>:=<ord>}
Værdien af udtrykket er <parameter>, hvis den er blevet sat, ellers er udtrykkets værdi <ord>, og parameter sættes til <ord>,

${<parameter>:?<ord>}
Værdien af udtrykket er <parameter>, hvis denne er sat. Ellers udskrives scriptets navn, parameterens navn, <ord> og scriptet stopper. <ord> er altså her en fejlmeddelelse.

${<parameter>:+<ord>}
Værdien af udtrykket er <ord>, hvis <parameter> er blevet sat, ellers får udtrykket ikke nogen værdi.

Hvis man i ovenstående udtryk udelader ":" vil shell kun undersøge, hvorvidt parameteren er sat.

Eksempler:

```
$ cat script3

PAR=$1
echo "Hunden hedder "${PAR:-ingenting}

$ script3

Hunden hedder ingenting

$ script3 Fido

Hunden hedder Fido
```

```
$ cat script4

PAR=$1
KAT=${PAR:=$HOME}
cd $KAT
echo "det nye katalog er `pwd`"
# Bemærk at der kun skiftes katalog,
# så længe scriptet kører
```

I scriptet **script4** benyttes **`pwd`**. Dette betyder at kommandoen **pwd** udføres og resultatet af dette skrives.

```
$ pwd

/bruger/jst/scripts


$ script4 /

Det nye katalog er /


$ script4

Det nye katalog er /bruger/jst
```

3.4 Brug af set-kommandoen til positionelle parametre

Med kommandoen **set** har man mulighed for at tildele værdier til de positionelle parametre inde i et script.

```
$ script5 Hansen Peter

Værdien af $1 er: Hansen
Værdien af $2 er: Peter
Værdien af $1 er nu: Andersen
Værdien af $2 er nu: Lise


$cat script5

echo 'Værdien af $1 er: ' $1
echo 'Værdien af $2 er: ' $2
set Andersen Lise
echo 'Værdien af $1 er nu: ' $1
echo 'Værdien af $2 er nu: ' $2
```

Ved kommandoen:

```
set `date`
```

vil $1 blive sat til ugedagen, $2 til måneden, $3 til månedsdagen, og så fremdeles.

3.5 Standard variable

I UNIX er defineret en række standard variable, bl.a.:

$#	**Antal positionelle parametre**
$?	**Værdien af den sidst udførte synkrone kommando**
$$	**Procesnummeret for denne shell**
$!	**Procesnummeret for den sidst startede baggrunds-proces**

$? er en værdi eller exitkode, der angiver hvordan den sidst udførte kommando er forløbet. Generelt betyder værdien 0, at kommandoen er forløbet uden fejl.

For kommandoen **grep** er værdierne f.eks. som følger:

0	**Det søgte blev fundet**
1	**Det søgte blev ikke fundet**
2	**Syntaksfejl i kommandolinien**

Opgave 2

Formål: - *at opnå fortrolighed med variable og positionelle parametre og forstå, hvorledes de anvendes i scripts*
- *at udvikle egne shellscripts til brug ved systemadministration*

1. **Afprøv de scripts, der er angivet i eksemplerne.**

   Hvad sker, hvis man angiver for mange eller for få parametre til et script?

2. **Skriv et script, der indeholder følgende linier:**

```
echo ${a:-Goddag}
echo $a
echo ${b:=Andersen}
echo $b
echo ${c:+Farvel}
echo $c
echo ${d:?Hovsa}
echo $d
```

   Afprøv scriptet.

   Hvilke af variablene får tildelt en værdi?

   Lav scriptet om, så a tildeles værdien af $1 i første linie, og prøv scriptet med en positionel variabel. Forklar forskellen.

   Lav nu scriptet om, så der i de to sidste linier benyttes

   1. **a i stedet for d.**
   2. **b i stedet for d.**

3. Lav et script, der indeholder nedstående:

```
echo "Indtast navn på floppydrev [/dev/flop]:\c"
read SVAR
FLOP=${SVAR:-/dev/flop}
echo "Der benyttes " $FLOP
```

Afprøv scriptet.

Som det forhåbentlig fremgik, vil FLOP blive sat til /dev/-flop, hvis der blot tastes retur, og ellers til den angivne værdi.

Lav scriptet om, så der ikke spørges, men blot vil blive brugt den positionelle parameter $1, og hvis denne ikke angives, skal den benytte standardværdien /dev/flop.

4. Med:

```
grep "^el:" /etc/passwd
```

udskrives linien for brugeren med login "el" i passwdfilen, da ^ først i udtrykket, betyder, at der ledes først på linien.
Lav et script, så man kan angive et bruger-login som argument, og få udskrevet den pågældende brugers linie i /etc/passwd.

5. (EKSTRA)
Fremstil et "terminal tester script".
Scriptet kan f.eks. aflæse terminalens stty2 parametrene, udskrive en meddelelse på terminalen, og aflæse sidst udførte terminology kommando på terminalen.

4 Kommunikation mellem processer

I Shell kan programmer kommunikere via temporære filer eller pipes. Således kan man sende input fra et program ned i en temporær fil, og lade et andet program tage input fra denne fil.

```
$ ls -o > /tmp/filer
$ cut -c15-20,45-80 /tmp/filer

jst     kontrol1
jst     kontrol2
jst     kontrol3
jst     prove1
jst     prove2
jst     script1
jst     script2
jst     script3

$ rm /tmp/filer
```

Når man benytter temporære filer, skal man være opmærksom på 2 ting:

- **selv at fjerne temporære filer**
- **passe på navngivning**

De bedste steder at gemme temporære filer er i **/tmp** kataloget, eller hvis man har et **/usr/tmp** katalog, da disse kataloger som standard bliver slettet hver gang, der bootes.

Med henblik på navngivning, bør man være varsom, således at man ikke kommer til at skrive i allerede oprettede filer. Den bedste metode må være navngivning ved hjælp af procesnummer, således at man i ovenstående kunne have benyttet den temporære fil:

```
/tmp/filer$$
```

Da det procesnummer, der fremkommer, er nummeret på den shell, man arbejder i, vil man i samme shell hele tiden kunne referere til **/tmp/filer$$** med dette navn.

4.1 Pipelines

Da det er temmelig besværligt først at skulle oprette temporære filer, og dernæst huske at få dem slettet, er det ofte bekvemt at benytte sig af en pipe, hvor man direkte sender output fra et program videre til et andet.

Det nævnte eksempel kunne da klares ved at skrive:

```
$ ls -o ^ cut -c15-20,45-80

jst    kontrol1
jst    kontrol2
jst    kontrol3
jst    prove1
jst    prove2
jst    script1
jst    script2
jst    script3
```

Her sparer man helt oprettelsen af den temporære fil.

Dette kan naturligvis udvides, så man konstruerer linier, hvor der er pipes fra det ene program til det næste etc. (såkaldte pipelines).

Man bør dog være opmærksom på, at fejlmeddelelser ikke sendes videre i pipen, uden at man selv specificerer dette.

4.2 Navngivne pipes

I UNIX har man mulighed for at fremstille en fil, der kaldes en navngiven pipe eller en FIFO. Disse er specialfiler netop til kommunikation mellem programmer.

Hvis man ønsker at benytte en FIFO, skal man sørge for at skrive til den fra en baggrundsproces, da selve den proces, der skriver til pipen, først starter, når man begynder at læse fra pipen.

FIFO står for "First In First Out", og skal forstås således, at de processer, der har ventet længst, kommer til først, (en "kø").

Et kendt eksempel er f.eks. spoolersystemet, der benytter en pipe med navnet **/usr/spool/lp/FIFO** til overførsel af tekster.

En FIFO, ved navn **fifo**, oprettes ved kommandoen

```
$ mknod fifo p
```

Hvor **p** står for "pipe".

```
$ mknod FIFO p
$ ls -o > FIFO &
$ cut -c15-20,45-80 FIFO

jst     kontrol1
jst     kontrol2
jst     kontrol3
jst     prove1
jst     prove2
jst     script1
jst     script2
jst     script3
```

Som det fremgår af eksemplet, virker en FIFO næsten som en temporær fil, men i modsætning til en sådan, vil man se, at størrelsen af en FIFO i filsystemet aldrig overstiger 2 Kb.

Endnu en anvendelse af FIFO'er er, at man kan starte flere programmer på samme tid ved at lade programmerne skrive til pipen. Disse vil da først starte, når man begynder at læse fra FIFOen

Opgave 3

Formål: - at arbejde med temporære filer
- at opnå fortrolighed med FIFO begrebet

1. Prøv de anførte eksempler.

2. Prøv følgende script:

```
TEMP=/tmp/$0$$
echo "Data vil blive gemt i "$TEMP
ls -la > $TEMP
```

   Hvilke "huller" bør man være opmærksom på?
   Lav scriptet om, så det rydder pænt op efter sig.

3. Forklar forskellen mellem følgende eksempler:

```
$ ls -o ^ cut -c10-20 ^ cut -c5-10
$ ls -æ ^ cut -c10-20 ^ cut -c5-10
$ ls -æ 2>&1 ^ cut -c10-20 ^ cut -c5-10
$ ls -æ 2>&1 ^ cut 10-20 ^ cut -c5-10
$ ls -æ 2>&1 ^ cut 10-20 2>&1 ^ cut -c5-10
$ ls -æ 2>&1 ^ cut 10-20 2>&1 ^ cut 5-10
```

   2>&1 bevirker at fejlmeddelelser omdirigeres til standard output.

4. Opret en FIFO.
   Lad flere programmer skrive til den, og undersøg med ps om programmerne er aktive.
   Kopier indholdet af pipen til /dev/null og undersøg igen med ps, om programmerne er aktive.

5 Kontrolstrukturer

I shell benyttes følgende valgstrukturer til at kontrollere udførelsen af kommandoforløb.

```
if list
then list
[elif list
then  list]
[else list]
fi
```

```
case word in
pattern) list;;
pattern) list;;
:
[*)      list;;]
esac
```

hvor *list* er en kommandorække, *word* er et shellord, typisk en variabel, og *pattern* er et mønster.

Strukturen **case** benyttes, når der er et valg mellem mange muligheder, mens strukturen **if** benyttes til valg mellem to, højst 3 muligheder.

5.1 Valg med if

I en **if** sætning testes på exitkoden af den første kommandorække efter **if**, der kunne være en betingelse lavet ved hjælp af **test** kommandoen. Hvis den er nul, udføres kommandorækken efter første **then**.

Hvis der herefter er en **elif**, testes på exitkoden af den efterfølgende kommandorække. Kommandoerne efter **then** udføres, hvis exitkoden herfra er nul.

Endelig, hvis der herefter er en **else**, og begge de to test (eller testen efter **if**, hvis der ikke er en **elif**) har givet exitkoder forskellige fra nul, udføres sidste kommandorække.

Eksempel 1:

```
if mount /dev/flop /install > /dev/null 2>&1
then
    break
else
    echo "Kan ikke mounte floppy." >&2
    exit 3
fi
# Her kommer masser af interessant kode...
```

I dette eksempel testes om kommandoen:

```
mount /dev/flop /install
```

giver en exitkode 0, som betyder at kommandoen forløb godt.

Eventuelt output omdirigeres til **/dev/null**, systemets "skraldespand", og eventuelle fejlmeddelelser (2, STANDARD ERROR) omdirigeres til STANDARD OUTPUT (1), som jo netop var sat til **/dev/null**.

Hvis udførelsen af kommandoen gik godt, hoppes ud af **if** sætningen med kommandoen **break**, og da **if** sætningen forløb godt, er exitkoden 0.

Hvis mountningen ikke gik godt, udskrives en fejlmeddelelse, og exitkoden sættes til 3 med kommandoen

```
exit 3
```

Eksempel 2:

```
mount /dev/flop /install > /dev/null 2>&1

if [ $? -eq 0 ]
then
        break
else
        echo "Kan ikke mounte floppy."
        exit 3
fi
# Her kommer igen den interessante kode
```

Her sker nøjagtig det samme som før. Testen ser blot lidt anderledes ud, men udfører det samme.

Eksempel 3:

```
if mount /dev/flop /install > /dev/null 2>&1
then
        break
elif tar -tvf /dev/flop > /dev/null 2>&1
then
        echo "Det er en tar-diskette."
        exit 4
else
        echo "Kan ikke mounte floppy."
        exit 3
fi
# Endnu mere interessant kode,
# der evt tester på denne exitkode.
```

Her ses et eksempel på brugen af **elif**, samt den differentierede exitkode, der gør det muligt at teste på udfaldet af denne kommando senere i scriptet.

5.2 Valg med case

I en **case** sætning, kan man lave valg mellem mange muligheder. Typisk tester man på værdien af en variabel, og udfører en aktion svarende til værdien af denne variabel.

Hvis værdien ikke er en af de opgivne *patterns*, udføres kommandorækken ud for *). Denne kommandorække skal altid komme til sidst, da værdierne testes i rækkefølge.

Eksempel 4:

```
case $1 in

flop)
                echo "Læser fra flop."
                tar -tvf /dev/$1;;

stream)
                echo "Læser fra streamer."
                btar -tvf /dev/$1;;

video)
                echo "Læser fra video."
                btar -tvf /dev/$1;;

*)
                echo "Enhed ikke angivet."
                exit 3;;
esac
```

I eksemplet benyttes argumentet $1, til at angive hvilken enhed, der skal læses fra.

I mønstrene kan man benytte maskerne *, ?, og [..], hvor betydningen er:

```
*       matcher alt
?       matcher ét enkelt tegn
[a-c]   matcher intervallet mellem a og c
[and]   matcher ét af bogstaverne a, n eller d
```

Eksempel 5:

```
case $1 in

fl?p)
                echo "Læser fra et af diskette drevene."
                tar -tvf /dev/$1;;

:
:
```

Eksempel 6:

```
case $SVAR in

ja ¦ JA ¦ yes)
                echo "Tak for tilliden.";;

*)
               echo "Du burde nu have svaret ja.";;
esac
```

Opgave 4

Formål: - at arbejde med if og case strukturerne samt exitkoder

1. Prøv de i teksten anførte scripts.

   Test efter udførelsen af et script, hvilken exitkode der sendes til shell, ved at udskrive værdien af $? umiddelbart efter udførelsen.

2. Fjern tretallet i linien

   ```
   exit 3
   ```

   i Eksempel 1, og udfør scriptet igen.
   Forklar hvorfor der nu altid returneres med exitkode 0, uafhængigt af om mountningen gik godt eller skidt.
   Fjern nu linien

   ```
   echo "Kan ikke mounte floppy."
   ```

   og udfør scriptet igen. Hvilken værdi returneres nu?
   Hvilken exitkode returnerer kommandolinien:

   ```
   mount /dev/flop /install > /dev/null 2>&1
   ```

   hvis der ikke er en diskette med et filsystem i drevet?

3. Flyt testen af *) op som den første test i eksempel 4. Udfør scriptet med og uden argumenter, og forklar resultatet.

4. Lav terminal tester scriptet om, så det også udfører en niocctl på terminalen, hvis denne er på nettet.

6 Løkker

Der findes 3 forskellige løkkestrukturer i UNIX: **for**, **until** og **while** løkker. **While** og **until** løkker benyttes når en kommandorække skal gennemløbes et antal gange afhængig af en test, og **for** løkken benyttes, når en kommandorække skal gennemløbes for enhver af en række værdier.

6.1 For løkker

Syntaksen for en **for** løkke ser således ud:

```
for word [in values]
do
        list
done
```

Her gennemløbes kommandorækken *list* for hver af værdierne i *values*, og hvis man udelader "**in** *values*", gennemløbes kommandoerne for hver af de positionelle parametre, $1, $2, ...

Eksempel 1:

```
for terminal in /dev/tty1[0-5]
do
        echo "Sender en terminology til $terminal."
        /etc/terminology -i int/dde450.t $terminal
done
```

Her sættes variablen **terminal** lig med **/dev/tty10**, ... **/dev/tty15** i rækkefølge, og der sendes en terminology til hver af disse terminaler.

Eksempel 2:

```
for bruger
do
        write $bruger << STOP
        Log venligst af..... Tak.
STOP
done
```

Dette script vil skrive meddelelsen til hver af de brugere, der angives som argumenter til scriptet.

Bemærk i eksemplet, at **write** ikke tager sit input fra en fil eller fra standard input, men i stedet fra det, der står efter kommandoen, frem til den støder på STOP anden gang. Her skal STOP stå i første position for at blive genkendt.

Man kunne også have valgt en anden "stopklods" ved f.eks. at skrive:

```
write $bruger << *
Bla, bla, bla, bla
...
*
```

Her er * valgt som "stopklods".

Denne type input kaldes et "here document".

6.2 While løkker

Syntaksen for en **while** løkke er:

```
while
list
do
        list
done
```

Her udføres kommandoerne mellem **do** og **done**, så længe den sidste kommando inden **do** returnerer en exitkode på 0.

Eksempel 3:

```
while
        who ^ grep console
do
        sleep 5
done
```

I dette eksempel vil scriptet køre i løkke, indtil der ikke længere er nogen logget ind på konsollen.

Ofte kan det være hensigtsmæssigt med uendelige løkker, og disse kan konstrueres med en **while** løkke, hvor man tester på ":", den tomme kommando, der altid returnerer værdien nul.

```
while :
do
     sleep 5
     ...
done
```

6.3 Until løkker

Syntaksen for en **until** løkke er:

```
until
   list
do
   list
done
```

Her gentages kommandorækken mellem **do** og **done**, til og med exitkoden for sidste kommando i den første kommandorække er nul. D.v.s. mindst én gang.

Eksempel 4:

```
until
who ^ grep console
do
       sleep 5
done
```

Her ventes i løkken, indtil der er nogen, der logger ind på konsollen. Bemærk, at det er det modsatte af, hvad der skete i eksempel 3.

Opgave 5

Formål: - at afprøve diverse trick i forbindelse med løkker

1. **Afprøv de angivne scripts.**

2. **Fremstil et script, der registrerer, hvornår brugere logger ind og ud fra maskinen.**

   I kan f.eks. benytte kommandoen diff, der angiver forskellen mellem to filer.

   Lad scriptet skrive ned i en logfil.

3. **Med programmet gendev, kan man få information om enheders opsætning:**

   ```
   gendev 6 /dev/tty#
   ```

   Fremstil et script, der udskriver hele /dev-kataloget, hvis der ikke angives parametre, og ellers kun undersøger for de angivne enheder f.eks. som

   ```
   devcheck tty17
   ```

   eller

   ```
   devcheck tty*
   ```

   hvor den sidste checker alle tty'erne.

7 Procesovervågning

De væsentligste kommandoer til procesovervågning er:

ps
who
whodo

som alle er standard Unix værktøjer. Derudover kan de specielle DDE værktøjer **sysstat** og **sysdisp** benyttes, men da disse er tænkt som interaktive programmer, vil de ikke blive kommenteret nærmere her.

For en systemadministrator er det ret væsentligt at kunne kontrollere processer samt deres ressourceforbrug, for at få den bedst mulige udnyttelse af maskinen.

F.eks. bør der ikke køre flere processer end nødvendigt. Systemet bør undersøges for net-dæmoner eller andet, der kun benyttes i få tilfælde, men er krævende rent MCU mæssigt.

Desuden er det nødvendigt, at kunne kontrollere, at visse processer kører. F.eks. **cron** eller spoolersystemet (**lpsched**).

7.1 ps - kommandoen

Kommandoen **ps** giver en oversigt over de processer, der kører på den pågældende terminal (i det aktuelle vindue).

```
$ ps

  PID  TTY          TIME  COMMAND
26709  tty24        0:01  menu
 8278  tty24        0:01  dsh
26725  tty24        0:01  ps30
```

Her står **PID** for det nummer processen har fået tildelt, **TTY** står selvfølgelig for terminal nummer, **TIME** er det samlede MCU tidsforbrug, og **COMMAND** er kommandoens navn.

Hvis der står et ? i kollonnen **TTY**, betyder det, at processen ikke er tilknyttet en kontrol terminal. Dette gælder f.eks. baggrundsprocesser.

Hvis der står **<defunct>** i kollonnen **COMMAND**, betyder det, at processen er termineret, men har en "forældreproces", der endnu ikke har ventet på denne proces.

Kommandoen tager bl.a. følgende options:

-u (user) efterfulgt af et login, vil give en oversigt over de processer den pågældende bruger kører.

-t (tty) efterfulgt af et tty-nummer, vil give en oversigt over de processer, der kører på terminalen.

De nævnte options kan bruges med flere argumenter, f.eks.:

```
$ ps -t03,05
```

der vil udskrive de processer, der er tilknyttet terminalerne **/dev/tty03** og **/dev/tty05.**

Hvis man benytter option **-e**, vil man få udskrevet samtlige processer, der er startet på maskinen.

Med option **-f** ændres billedet, så man også får udskrevet:

UID	**Brugerens login**
PPID	**Processens forældreproces nr.**
C	**Schedulerings nr. (ALTID 0 PÅ EN SUPERMAX)**
STIME	**Starttidspunkt for kommandoen**

og **COMMAND** udskriften ændres, således at den fulde kommando med options skrives:

```
$ ps -f

UID   PID  PPID  C     STIME TTY      TIME COMMAND
jst 26709  8305  0  15:06:35 tty24    0:01 /alib/kontor/
menu startmenu
jst  8278 26709  0  15:06:37 tty24    0:01 dsh
jst  2207  8278  0  15:31:33 tty24    0:01 ps -f
```

Processer, der blev startet mere end 24 timer før ps kommandoen, vil under **STIME** stå med dato for start. Bemærk at processer godt kan fylde mere end 1 linie, hvis det er et langt kommandofelt.

Hvis man vil teste om f.eks. processen **cron** er aktiv, kan man skrive:

```
$ ps -e ^ grep cron
```

og få udskrevet den linie, der vedrører **cron**.

7.2 Who

Kommandoen **who** rapporterer hvilke brugere, der er logget ind, på hvilke terminaler og hvornår:

```
$ who

jko     tty03       Jan 16  12:29
root    console     Jan 16  14:32
jst     tty24B      Jan 16  13:26
bfn     tty25       Jan 16  12:08
```

Som datafiler til dette, bruger **who** filerne **/etc/inittab** og **/etc-/utmp**

Kommandoen har følgende options:

-r	run level	maskinens nuværende "run level"
-b	boot	tidspunkt for sidste boot
-t	time	tidspunkt for sidste ændring af tid
-p	proces	processer forskellige fra getty og brugerprocesser
-l	login	getty processer
-u	use	tid siden sidste aktivitet på terminalen og procesnummer rapporteres
-d	dead	døde processer
-A	accounting	accounting oplysninger
-a	all	rbtpludA options
-s	short	kort version (svarer til ingen options)
-H	header	overskrifter på felterne
-T	tty	terminalernes tilstande skrives med: + kan skrives til med write, - kan ikke skrives til med write, ? terminalen "hænger".
-q	quick	kun indloggede brugere

Kommandoen kan også rapportere, hvad der er sket, hvis man angiver filen **/etc/wtmp** som argument således:

```
$ who -a /etc/wtmp
```

ellers læses fra filen **/etc/utmp**.

Hvis man benytter option **-u**, får man kolonnen **IDLE**, der angiver hvorvidt, der har været aktivitet på terminalen i det sidste minut med et punktum, og ellers angives i hvor lang tid, der ikke har været aktivitet. Hvis der ikke har været aktivitet i mere end 24 timer, skrives **old** i kolonnen.

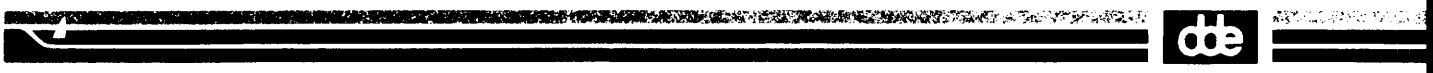
```
$ who -u

NAME    LINE     TIME             IDLE      PID      COMMENTS
erj     tty03    Jan 17 08:39     0:13     4200
jam     tty01    Jan 17 08:19      .      28713
ib      tty05    Jan 17 08:34     0:01     4119
kf      tty02    Jan 17 08:41     1:25    22761
ulf     tty11    Jan 16 12:41      .      12321
abk     tty10    Jan 17 08:30      .      28862
sch     tty09    Jan 17 10:19     0:02    28856
jst     tty24B   Jan 17 08:39      .      28869
```

Bemærk at oplysningerne i **IDLE** kolonnen ikke altid er helt korrekte.

Kolonnen **COMMENTS** kan indeholde en kommentar, der angiver noget om den pågældende terminal, f.eks. at det er et modemlogin. Denne kommentar hentes fra **/etc/inittab** filen, hvor linien for den pågældende terminal f.eks. kunne se således ud:

```
09:234:respawn:/etc/getty -h tty09 19208D none LDISSC1 # modem
```



Option **-l** benyttes til at finde de terminaler, hvor processen er login/getty.

```
$ who -l

NAME    LINE         TIME           IDLE      PID     COMMENTS
LOGIN   console      Jan 16 14:32   11:22   30988
LOGIN   /dev/tty06   Jan 17 08:40     .      4296
LOGIN   /dev/tty04   Jan 13 12:28     .      2323
LOGIN   /dev/tty19   Jan 17 10:10    0:11   12332
LOGIN   /dev/tty22   Jan 13 12:28     .      2064
LOGIN   /dev/tty09   Jan 13 12:28     .      2070     modem
```

Bemærk at processerne angives som LOGIN og ikke getty. Den oversigt, der fås på denne måde, skulle gerne stemme rimeligt overens i antal linier med

```
ps -e ^ grep getty
```

I kolonnen **IDLE** bør man være opmærksom på, om der er nogle terminaler, der ofte står med **LOGIN.** Hvis dette er tilfældet, kunne man overveje at sætte den til **off** i **/etc/inittab.**

7.3 Whodo

En kombination af who og ps kommandoerne får man med kommandoen **whodo:**

```
$ whodo

Thu jan 17 10:50:31 1991
kaka

tty03    erj        8:39
    tty03    4200    0:00 ds
    tty03     191    0:00 menu

tty05    ib         8:19
    tty05    4199    0:01 ds
    tty05    4248    0:03 stfile
    tty05    4259    0:01 menu

tty24B   jst        8:39
    tty24    8303    0:00 menu
    tty24   18544    0:00 dsh
    tty24    8372    0:00 lp
    tty24   14517    0:00 whodo
    tty24   28869    0:02 ds
    tty24    2247    0:00 menu
    tty24   26225    0:00 sh
    tty24    2250    0:00 sh
    tty24   28876    0:00 menu
    tty24    2253    0:00 stfile
    tty24    4344    2:30 sted
    tty24   18710    0:00 sthypdk
```

Kommandoen henter oplysningerne fra **/etc/utmp** og fra **/etc/ps_data.**

Reference: System V Reference Manual

Opgave 6

formål: - gennemgang af options til kommandoerne ps og who samt overførsel af resultaterne fra disse kommandoer til løkker.

1. Lav et script, der finder processer, som ikke er tilknyttet en terminal.

2. Kommandoen kill forventer man angiver et procesnummer for at kunne udføres. Fremstil et "zap" program, hvor man kan angive procesnavn i stedet for procesnummer.

   Hvad bør man være opmærksom på, når man benytter dette program.

3. Fremstil et script, der udskriver en advarsel, hvis der kører mere end én lpsched.

   Lav scriptet om, så det slår de pågældende lpsched-processer ihjel, starter en enkelt op, samt skriver en meddelelse om, at dette er gjort, i /usr/lib/errlog/log og på konsollen.

   Brug /etc/rc.d/lp til at starte flere samtidige lpsched-processer.

4. Fremstil et script, der optæller, hvor mange processer hver enkelt bruger har startet.

5. Lav et script, der rapporterer hvilke terminaler, der ikke har været aktive i mere end 15 minutter.

8 Diskpladsovervågning

Kommandoerne **df** og **du** er beregnet til at fortælle om diskpladsforbrug. Desuden kan **find** kommandoen med relevante options benyttes til at fortælle om store filer, ligesom **sysadm** programmet indeholder faciliteter for dette. Er accounting pakken installeret på maskinen, kan man endvidere benytte programmet **diskusg**.

Man bør være opmærksom på at den blokstørrelse, der benyttes i kommandoerne er på 512 b (½ Kb), mens der benyttes 2 Kb blokke på disken. 512 bytes blokstørrelse er opretholdt af hensyn til sammenligningen med tidligere versioner af Unix.

8.1 df

Kommandoen **df** angiver antallet af frie blokke og i-noder på mountede filsystemer.

```
$df

/       (/dev/dsk/ul3c8s0):      9724 blocks   26533 i-nod
/lisda  (/dev/dsk/ul3c8s1):     78460 blocks   29958 i-nod
/tmp    (/dev/dsk/ul3c8s2):     29736 blocks    7674 i-nod
/bruger (/dev/dsk/ul3c8s3):    236516 blocks   61238 i-nod
```

Benyttes option **-t** rapporteres tillige det totale antal blokke og i-noder.

Hvis man ønsker en oversigt med angivelse af, hvor mange procent af diskpladsen der er frit, kan man kalde **Sysadm filemgmt** scriptet **diskuse**

```
$ /usr/admin/menu/filemgmt/diskuse

FILE SYSTEM USAGE AS OF 01/17/91 12:59:13

File                    Free     Total    Percent
System                  Blocks   Blocks   Full
------                  ------   ------   -------
/                       9704     122880   92%
/bruger1                236516   245760   3%
/bruger2                175052   245760   28%
/lisda                  78460    122880   36%
/tmp                    29736    30720    3%
/usr/adm/KA             142116   184320   22%
```

8.2 du

Kommandoen fortæller antallet af blokke, der benyttes af angivne kataloger/filer.

```
$ du /etc/rc.d

29 /etc/rc.d
```

En mere specificeret rapport fås ved at benytte option **-a**:

```
$ du -a /etc/rc.d

1       /etc/rc.d/.InitTerm.dde
1       /etc/rc.d/MOUNTFILESYS
1       /etc/rc.d/PRESERVE
1       /etc/rc.d/RMTMPFILES
1       /etc/rc.d/cron
1       /etc/rc.d/errlog
1       /etc/rc.d/lp
1       /etc/rc.d/makeerr
1       /etc/rc.d/.opendir.dde
1       /etc/rc.d/opendir
7       /etc/rc.d/dbstart
3       /etc/rc.d/InitTerm
1       /etc/rc.d/nodename
1       /etc/rc.d/InitPrt
3       /etc/rc.d/logging
1       /etc/rc.d/job
1       /etc/rc.d/afvstart-ss
1       /etc/rc.d/uniplex
29      /etc/rc.d
```

Bemærk at den samlede katalogstørrelse angives til sidst i listen.

8.3 find

Kommandoen har forskellige options, der bl.a. kan benyttes til at finde filer, der fylder mere end en vis størrelse:

```
$ find /bruger1 -size +50 -print

/bruger1/adbsys/night/link.cmd
/bruger1/tss/piausv1509tss0
/bruger1/dkg/lars---A.ex
```

Option **-size** kan, som vist, finde filer af en vis størrelse:

+n	**finder filer, der er større end *n* blokke**
-n	**finder filer, der er mindre end *n* blokke**
n	**finder filer, der er præcis *n* blokke**

Et efterfølgende **c** kan benyttes til at finde filer, der fylder et angivet antal tegn.

Bemærk at der i **sysadm** pakken er scripts, der kan finde store eller gamle filer.

Opgave 7

Formål: - *at prøve de forskellige options til df, du og find*
- *at fremstille scripts til overvåging af plads*

1. **Prøv kommandoerne df, du og find med de forskellige options.**

2. **Lav et script, der dels udskriver hvor meget plads, der totalt udnyttes på diskene, dels gennemgår brugernes hjemmekataloger og udskriver, hvor meget plads disse optager.**

3. **Fremstil et script, der finder filer i filhierarkiet, der er større end en vis størrelse.**

   Lav scriptet, så det som parameter tager størrelsen, og hvis denne ikke angives, findes filer større end 512 Kb (1024 blokke á 512 b).

   Lav scriptet om, så man som parameter også kan angive et brugernavn, for hvilket man ønsker pladsforbruget undersøgt.

9 Regulære udtryk

Regulære udtryk benyttes til at finde bestemte mønstre i filer. De simpleste regulære udtryk indeholder ikke specialtegn, som i denne sammenhæng er:

.	matcher et vilkårligt tegn
[..]	definerer en karakterklasse
*	matcher nul eller flere af det foregående tegn
^	matcher starten af en linie
$	matcher slutningen af en linie
\	ophæver betydningen af et specialtegn

9.1 Simple udtryk

Som omtalt er de simpleste udtryk uden specialtegn:

UDTRYK	MATCHER	EKSEMPLER
/al/ /var vi/	al var vi	al, alle, hallo var vi, var vinder, svar videresendes

I eksemplet er de regulære udtryk angivet mellem to "/".

9.2 Specialtegn

Tegnet . matcher et vilkårligt tegn.

UDTRYK	MATCHER	EKSEMPLER
/.ak/	noget der indeholder et tegn og ak	tak, hakke, praktik
/ .ak/	mellemrum efterfulgt af et tegn og ak	tak, pakke

To firkantede parenteser definerer en karakterklasse:

UDTRYK	MATCHER	EKSEMPLER
/t[ae]k/	tek el. tak	stak, teknik, taktik
/[0-3][0-9]	mønstre der indeholder fra 00 til 39	24, 1258, men ikke 45678

Hvis man ikke ønsker at matche med karakterklassen, kan der sættes en ^ i starten:

UDTRYK	MATCHER	EKSEMPLER
js[^abc]	alt hvad der indeholder js efterfulgt af noget andet end a, b eller c	vejskat, men ikke hejsa

En * benyttes til at gentage det foregående regulære udtryk:

UDTRYK	MATCHER	EKSEMPLER
/ab*c/	et a efterfulgt af ingen eller flere b'e og et c	abc, abbc, ac
/k[0-5]*/	alle udtryk der indeholder k efterfulgt af 0 eller flere cifre 0-5	jakke, ok345s, sk3
/(.*)/	Længste streng mellem (og)	(Dette er en test), (Dette er en (lille) test)
/([^)]*)/	Korteste streng mellem (og)	(Dette er en test), (Dette er en (lille) test)

De to tegn ^ og $ benyttes til at henvise til henholdsvis starten og slutningen af en linie:

UDTRYK	MATCHER	EKSEMPLER
/^T/	Et T i starten af en linie	Tak for i aften, Tænk
/:$/	Et : i slutningen af en linie	...har vi: .../bruger2/ole:

Hvis man ønsker at henvise til et af specialtegnene i sit søgemønster, gøres dette ved at benytte en \ (backslash) foran tegnet:

UDTRYK	MATCHER	EKSEMPLER
/3\.5/	3.5	763.546
/\/[a-z]*/	en / efterfulgt af 0 eller flere bogstaver	/, /andersen

Når man benytter regulære udtryk, skal man være opmærksom på, at der altid foretages det længst mulige match, som f.eks. anført i eksemplet med **(Dette er en test)**.

Reference: System V Reference Manual, ed side 2 - 5

Opgave 8

Formål: - at afprøve forskellige regulære udtryk
Metode: - kommandoen grep benyttes til at finde linier i en tekst ved hjælp af regulære udtryk:
grep '<regulært udtryk>' tekst

1. **Find alle de logins, der starter op i en dsh ved hjælp af grep kommandoen. (Det regulære udtryk skal sættes i anførselstegn).**

2. **Find linen med login "bin" i /etc/passwd.**

3. **Find ud af, om der findes nogle logins der hedder "adm", "bdm" eller "cdm".**

4. **Find de logins, der indeholder netop to bogstaver.**

5. **Find de linier i /etc/passwd, der indeholder et "a", et vilkårligt antal tegn og et "b".**

10 Streameditoren

Programmet **sed** behandler hver enkelt linie fra standard input i rækkefølge. Man kan så lade **sed** ændre på disse linier i overensstemmelse med kommandoerne for **ed**.

For at finde de linier, man ønsker at behandle, kan der benyttes regulære udtryk.

Ofte vil man ønske at foretage substitutioner og mindre ændringer af standard input, men man kan også lave endog meget komplicerede **sed** programmer.

Et program til sed består af en række kommandoer, der udføres på linierne én af gangen. Scriptet vil indeholde linier af formatet:

[adresse [adresse]] funktion [argumenter]

Input linierne kopieres cyklisk til et område kaldet **pattern space**, hvorefter funktionen udføres, hvis linierne ligger inden for ovenstående adresseområde.

En adresse kan enten være et linienummer eller linienumre udvalgt med regulære udtryk.

10.1 sed kommandoer

Addresseringsmulighederne er angivet i parentes. (.) betyder 1 adresse og (.,.) betyder 2 adresser.

(.)**q**	quit	list til og med det pågældende linienummer
(.,.)**d**	delete	slet de angivne linienumre
(.)**r**	read	indlæs en angivet fil efter det pågældende linienummer
(.,.)**w**	write	udskriv til en angivet fil
(.)**a**	append	tilføj efter den angivne linie
(.)**i**	insert	indsæt før den angivne linie
(.,.)**c**	change	udskift de angivne linier
(.,.)**s**	substitute	erstat
(.,.)**p**	print	udskriv de angivne linier

Kommandoen substitute er uhyre anvendt i forbindelse med **sed**. Der findes 3 flag, som kan anvendes sammen med substitute:

w	**write**	skriv ændringer til en ny fil
p	**print**	udskriv de ændrede linier
g	**global**	erstat alle forekomster på linien

10.2 Eksempler på sed kommandoer

Udskrift af de 10 første linier i /etc/passwd:

```
$ sed -n '1,10p' /etc/passwd
root:xR90iCUoYQT1s:0:1:0000-Admin(0000):/:/bin/dsh
daemon:disabled:1:1:0000-Admin(0000):/:
bin:1yWFZsZx7k/eU:2:2:0000-Admin(0000):/bin:
sys:disabled:3:3:0000-Admin(0000):/usr/src:
adm:disabled:4:4:0000-Admin(0000):/usr/adm:
uucp:disabled:5:5:0000-uucp(0000):/usr/lib/uucp:
nuucp:vsOFzyri6p4oA:10:10:0000-uucp():/usr/spool/uucppublic
:/usr/lib/uucp/uucico
rje:disabled:18:18:0000-rje(0000):/usr/rje:
trouble:disabled:70:1:trouble(0000):/usr/lib/trouble:
lp:disabled:71:2:0000-lp(0000):/usr/spool/lp:
```

Udskrift af linierne fra brugeren root til adm (hvis ikke mønsteret adm indgår i nogen af de mellemliggende linier):

```
$ sed -n '/root/,/adm/p' /etc/passwd
root:xR90iCUoYQT1s:0:1:0000-Admin(0000):/:/bin/dsh
daemon:disabled:1:1:0000-Admin(0000):/:
bin:1yWFZsZx7k/eU:2:2:0000-Admin(0000):/bin:
sys:disabled:3:3:0000-Admin(0000):/usr/src:
adm:disabled:4:4:0000-Admin(0000):/usr/adm:
```

Tilføj /bin/dsh til alle linier for brugere, der starter i /bin/sh (de tilsvarende linier i /etc/passwd slutter med et ":"):

```
$ sed 's-:$-:/bin/dsh-' /etc/passwd
```

Bemærk, at i det sidste udtryk er det regulære udtryk angivet mellem bindestreger, da udtrykket indeholder to "/".

I de to første eksempler er option **-n** brugt til at undertrykke udskrivning af de linier, der ikke matcher adresserne.

Som et eksempel på brug af **sed**, hvor der benyttes flere liner i et script, vil vi betragte følgende opgave:

Vi ønsker at få at vide, hvilke terminaler, der er i brug.

Til denne opgave vil vi benytte kommandoen **who**, hvor det normale output indeholder oplysninger om login, terminaler samt tidspunkt for login:

```
$ who
jam         tty01       Feb 18 10:50
abk         tty10       Feb 18 08:40
sch         tty09       Feb 18 07:48
ulf         tty11       Feb 18 12:31
amg         tty18       Feb 18 11:48
jst         tty24B      Feb 18 08:36
pet         tty25       Feb 18 10:00
```

Dette output vil vi sende videre til **sed** via en pipe.

I første omgang ønsker vi kun at ændre alle mellemrum til et enkelt mellemrum.

```
$ who ^ sed 's/  */ /g'
jam tty01 Feb 18 10:50
abk tty10 Feb 18 08:40
sch tty09 Feb 18 07:48
ulf tty11 Feb 18 12:31
amg tty18 Feb 18 11:48
jst tty24B Feb 18 08:36
pet tty25 Feb 18 10:00
```

Bemærk, at der er to mellemrum foran *, da der skal være mindst ét mellemrum, for at vi substituerer.

Herefter ønsker vi at få fjernet det første ord samt mellemrum. Dvs. brugerens loginnavn.

```
$ who ^ sed 's/  */ /g
> s/[^ ]* //'
tty01 Feb 18 10:50
tty10 Feb 18 08:40
tty09 Feb 18 07:48
tty11 Feb 18 12:31
tty18 Feb 18 11:48
tty24B Feb 18 08:36
tty25 Feb 18 10:00
```

Her søgte vi efter en række "ikke-blanktegn" efterfulgt af en blank, som vi erstattede med ingenting (slettede).

Til slut skal vi blot fjerne alt fra det første mellemrum til slutningen af linien:

```
$ who ^ sed 's/  */ /g
> s/[^ ]* //
> s/ .*//'
tty01
tty10
tty09
tty11
tty18
tty24B
tty25
```

Referece: System V Reference Manual, sed

Opgave 9

Formål: - *at blive fortrolig med streameditoren*
- *at lave større programmer med sed*

Metode: - *udskriv resultaterne til skærmen uden at overskrive systemfilerne.*

1. **Hvad laver følgende kommandoer:**

   a) **$ sed ' ' /etc/passwd**

   b) **$ sed '10q' /etc/passwd**

   c) **$ sed 's/:.*//' /etc/passwd**

   d) **$ ls -l ^ sed 's/ .* / /'**

   e) **$ who am I ^ sed 's/ .*//'**

2. **Lav om på linierne i /etc/passwd, så alle brugerne starter op i en dsh. Husk at en bruger starter op i en sh - shell, hvis ingen shell type er angivet.**

3. **Lav et script, der folder lange linier i f.eks. /etc/gettydefs. (Se på kommandoer i sed).**

4. **Fremstil et script, der fortæller, hvor en kommando er fundet.**

   Benyt variablen PATH til at angive hvilke kataloger, der skal søges i. Husk at et kolon i start eller i slutning af PATH eller to efterfølgende koloner betyder, at der skal søges i arbejdskataloget.

Ekstraopgave

1. Lav et script, så man kan ændre tilstanden for en terminal i /etc/inittab fra respawn til off og omvendt.

   Kaldet af scriptet kunne f.eks. se således ud:

   ```
   $ gettychange off tty17
   ```

   hvis man vil slå getty fra på den pågældende terminal.

2. Fremstil et script, der udskriver en terminals indgang i /etc/gettydefs.

```
PROMPT Generating Script to Move Index from one tablespace to another
ACCEPT from_tb_space char prompt "Enter Old Tablespace Name:"
ACCEPT to_tb_space   char prompt "Enter New Tablespace Name:"

spool move_idx.sql;

set verify off;
set feedback off;

set serveroutput on

declare

uniqueness      user_indexes.uniqueness%TYPE;
idx_name        user_indexes.index_name%TYPE;
table_name      user_indexes.table_name%TYPE;
ini_trans       user_indexes.ini_trans%TYPE;
max_trans       user_indexes.max_trans%TYPE;
pct_free        user_indexes.pct_free%TYPE;
col_name        user_ind_columns.column_name%TYPE;
sql_line        varchar2(200);
first_line      boolean;

cursor  get_indexes     is
        select  uniqueness, index_name, table_name, ini_trans, max_trans,
                pct_free
        from    user_indexes
        where   tablespace_name = '&From_Tb_Space'
        ;

cursor  get_ind_cols    is
        select  column_name
        from    user_ind_columns
        where   index_name = idx_name
        order   by column_position;

begin

dbms_output.enable( 100000 );
dbms_output.put_line('PROMPT Start Moving Indexes ....');
dbms_output.put_line('PROMPT');

open    get_indexes;
LOOP
        fetch get_indexes into uniqueness, idx_name, table_name, ini_trans,
                               max_trans, pct_free;
        EXIT when get_indexes%NOTFOUND;

        dbms_output.put_line('PROMPT Moving index:'||idx_name);
        sql_line := 'Create';
        IF   uniqueness = 'UNIQUE'
        THEN sql_line := sql_line || ' UNIQUE';
        END IF;
        sql_line := sql_line || ' index ' || idx_name ||
                                ' on ' || table_name;
        dbms_output.put_line(sql_line);
```

```
        first_line := TRUE;
        open    get_ind_cols;
        LOOP
                fetch get_ind_cols into col_name;
                EXIT when get_ind_cols%NOTFOUND;
                IF   first_line
                then sql_line := '     (' || col_name;
                     first_line := FALSE;
                else sql_line := sql_line || ',' || col_name;
                end if;
        END LOOP;
        close   get_ind_cols;
        sql_line := sql_line || ')';
        dbms_output.put_line(sql_line);

        dbms_output.put_line('     INITRANS ' || to_char(ini_trans));
        dbms_output.put_line('     MAXTRANS ' || to_char(max_trans));
        dbms_output.put_line('     TABLESPACE &To_Tb_Space');
        dbms_output.put_line('     PCTFREE ' || to_char(pct_free));
        dbms_output.put_line('     ;');

END LOOP;
close   get_indexes;

end;
/

spool off;

ed move_idx.sql
```

11 Baggrundsprocesser og signal-håndtering

I UNIX, der er et multiproces styresystem, er der mulighed for afvikling af mange processer på samme tid. En bruger kan således starte en masse processer op i baggrunden ved at angive et "&" efter programkaldene.

Dette er dog kun interessant for de brugere, der har adgang til shell.

Man kan desuden afvikle programmer på fastlagte tidspunkter via **cron**.

11.1 Baggrundsprocesser

Da man kan starte mange baggrundsprocesser op giver UNIX en lavere prioritet til disse job. Den tildelte prioritet kan eventuelt ændres med kommandoen **nice** inden start af kommandoen, men det er kun superbrugeren, der kan sætte prioriteten op for en kommando.

```
$ nice 19 <kommando> &          # lav prioritet

# nice -19 <kommando> &         # høj prioritet
```

Når man benytter baggrundsprocesser startet fra shell, skal man være opmærksom på at der sendes et hangup signal til ens baggrundsprocesser, når man logger ud.

For at undgå dette, kan de startes op med programmet **nohup** (no hangup). Dette kan specielt være væsentligt, når man kører over modem, så en baggrundsproces ikke stoppes, hvis forbindelsen skulle gå tabt.

```
$ nohup <kommando> &
```

Hvis man ikke i kommandolinien omdirigerer standard output og standard error vil **nohup** sende dette til filen **nohup.out** eller **$HOME/nohup.out**, hvis der ikke kan skrives i det pågældende katalog.

11.2 Afvikling af programmer med cron

UNIX brugere har mulighed for at få afviklet jobs på bestemte tidspunkter. Den ene er via kommandoen **at** og den anden med **crontab**.

Begge kommandoer kræver at den pågældende bruger er autoriseret til at benytte kommandoerne. Dette bliver man ved at skrive en linie med sit login i henholdsvis filen **/usr/lib/cron/at.allow** og **/usr/lib/cron/cron.allow**.

Når man benytter **at**, bevares brugerens environment variable, nuværende katalog, samt umask og ulimit til senere afvikling af de anførte kommandoer.

Hvis man afvikler jobs ved hjælp at **cron**, skal man huske selv at sørge for disse ting.

```
$ at now + 10 minutes
write lise <<STOP
Kaffen er færdig.
STOP
<Ctrl+D>
warning: commands will be executed using /bin/sh
job 637158600.a at Mon Mar 12 10:10:00 1990
```

11.3 Signalhåndtering

Signaler er hændelser, der sendes til processer, enten fra operativsystemet, eller ved hjælp af kommandoen **kill**. Disse signaler fortæller om forskellige hændelser, og er beskrevet i *System V, Reference Manual, section 2* under **signal**. Nedenfor er angivet en liste over nogle af disse signaler:

SIGHUP	**1**	**hangup**	**Forbindelsen er tabt til terminalen**
SIGINT	**2**	**interrupt**	**Svarer til at taste <Control+C>**
SIGQUIT	**3**	**quit**	**Svarer til at taste <Control+D>**
SIGKILL	**9**	**kill**	**Kan ikke fanges med trap**

De omtalte signaler sendes til processer fra shell med kommandoen **kill**.

```
$ kill -2 3045
```

svarer f.eks. til at sende et interrupt til processen med procesid 3045.

Systemadministratoren vil oftest sende disse signaler til brugerprocesser, der er kørt fast. Hvordan programmet vil reagere på signalet afhænger af programmøren.

Sædvanligvis sender de fleste blot signal 9 til programmet, for at få det til at stoppe, men dette behøver ikke være den bedste løsning.

Sender man f.eks. signal 2 til en Supermax Tekst redigering, vil man få afbrudt denne proces, og brugeren kan få fat i den redigerede tekst ved at vælge "rediger sidste". Sender man derimod signal 9, vil teksten i RAM-lageret, og dermed måske adskillige timers arbejde, gå tabt.

11.4 Trap kommandoen

Hvis man, i et script, ønsker at reagere mod de signaler, der sendes til shell, kan dette gøres med kommandoen **trap**.

Kommandoen tager to argumenter: en kommando der udføres, hvis det pågældende signal modtages, samt de signaler, der skal fanges.

```
$ cat script

:
:
trap "echo Der er tastet Control C" 2
:
:

$ script
^C
Der er tastet Control C
```

Hvis man laver et lille program, som man ikke ønsker, at en bruger skal kunne afbryde, kan man fange de signaler 2 og 3, der sendes, ved at skrive:

```
trap "" 2 3
```

i starten af programmet, og

```
trap 2 3
```

på det sted i programmet, hvor brugeren skal kunne afbryde igen.

Opgave 10

Formål: - at starte baggrundsprocesser
- at arbejde med signaler

1. Opret schellscriptet "langsomt" som:

   find / -name core 2>/dev/null

   Start scriptet op i baggrunden, og find ud af hvilken prioritet, processen får.

   Mens processen arbejder, logges ud og ind. Prøv både med og uden nohup.

   Start "langsomt" op med en lavere prioritet end default.

2. Sørg for at scriptet, der undersøger, hvorvidt der kører to lpsched processer, bliver startet op ved boot.

3. Lav et script, der registrerer, når der logges ind og ud på en bestemt terminal, f.eks. en modem linie, og start scriptet ved boot.

4. Undersøg indholdet af /usr/spool/cron/atjobs

5. Fremstil et brugervenligt script, så brugerne kan få udskrevet en meddelelse på deres skærm på et bestemt tidspunkt.

..opgaven fortsættes

6. Eksperimenter med f.eks. Supermax Tekst redigering og diverse signaler.

7. Fremstil et script, der registrerer, hvornår personer logger ind og ud på systemet.

   Du kan f.eks. bruge trap kommando 0 til at registrere, når der logges ud.

   Hvor bør scriptet kaldes for at fungere.

12 Oprydning i filsystemet

For at udnytte diskpladsen effektivt, er det nødvendigt, at man har nogle rutiner til at fjerne filer, der ikke længere er brug for.

Endvidere er der visse systemfiler, der skal vedligeholdes, idet de ellers vil vokse til en størrelse, hvor de bliver uhåndterlige.

12.1 Gamle filer

Kommandoen **find** har 3 options til specificering af tid i filsystemet:

-atime	**access time**	**tidspunkt for sidste læsning/skrivning/udførelse af filen**
-mtime	**modify time**	**tidspunkt for sidste ændring af filen**
-ctime	**change time**	**tidspunkt for ændring i i-noden. F.eks. oprettelse eller ændring af ejerforhold**

Følgende vil finde alle filer i brugerens hjemmekatalog, der ikke er blevet ændret i de sidste 180 dage:

```
$ find $HOME -mtime +180 -print

/bruger2/jst/oracle/afiedt.buf
/bruger2/jst/oracle/demobld
/bruger2/jst/oracle/format.sql
/bruger2/jst/oracle/login.sql
/bruger2/jst/oracle/ora
/bruger2/jst/oracle/rap.sql
```

Her betyder +180, at der skal søges efter filer, hvor det er mere end 180 dage siden, de blev ændret.

Hvis man ønsker en oversigt over filer, hvor det er mindre end 180 dage siden, de blev ændret, skal man i stedet benytte -180.

12.2 Systemfiler

Unix benytter en række filer til at holde rede på visse hændelser i systemet.

Disse filer vokser, og der skal derfor gøres noget ved dem, da de ellers vil nå grænsen på 1 Mb, hvor der ikke længere kan skrives til dem.

De filer, det drejer sig om, er følgende:

/usr/lib/cron/log
/usr/lib/errlog/log
/usr/adm/sulog
/etc/wtmp
/usr/mail - kataloget
/usr/spool/lp/log

Ud over disse filer, vil nogle programmer også skrive til diverse logfiler. Det drejer sig bl.a. om Tar Backupsystemet (TBS), Oracle, PC-Term og UUCP.

Disse logfiler skal systemadministratoren selv vide hvor ligger.

En meget benyttet måde til at begrænse systemfilernes størrelse er med kommandoen **tail**, der kan skære de sidste linier af en fil:

```
# tail -30 /usr/adm/sulog> /tmp/sulog$$
# mv /tmp/sulog$$ /usr/adm/sulog
```

Disse kommandoer vil forkorte suloggen til de sidste 30 linier.

Opgave 11

Formål: - lave scripts, der rydder op i filsystemet

1. Lav et oprydningsscript, der finder de filer i brugernes hjemmekataloger, der ikke er blevet brugt i en angiven periode.

   Scriptet skal tage en backup af disse filer, sende mail til brugeren og systemadministratoren om hvilke filer, det drejer sig om.

2. Find alle de filer i /dev - kataloget, der ikke er specialfiler eller kataloger.

   Find alle filer uden for /dev/kataloget, der er specialfiler.

3. Lav et script, der afkorter de omtalte systemfiler.

   Lad scriptet tage en sikkerhedskopi på diskette eller på streamer inden afkortningen.

   Bemærk at filen /etc/wtmp automatisk vil blive håndteret ordentligt af accounting, hvis denne pakke er installeret.

4. Sørg for at disse scripts bliver kørt i cron én gang om måneden.

13 Introduktion til awk

Awk er et programmeringssprog, der indeholder kontrolstrukturer, løkker etc., som alle andre programmeringssprog.

I modsætning til andre programmeringssprog, er **awk** først og fremmest beregnet til at håndtere filer, specielt med henblik på fremstilling af rapporter ud fra filernes indhold.

Navnet **awk** er et akronym for Aho, Weinberg og Kernighan, som er de tre udviklere bag sproget.

13.1 Opbygning af et awk-program

Et program til awk kan bestå af op til tre dele:

- en del der udføres inden **awk** går i gang med at behandle filerne (**BEGIN**)
- en del der behandler filen
- en del der udføres efter behandlingen er færdig (**END**)

Alle tre dele er valgfri, og de enkelte dele anføres i {..}:

```
$ awk 'BEGIN {print "OVERSKRIFT"; exit}'
OVERSKRIFT
```

13.2 Kald af awk

Brug af **awk** til behandling af filer, kan foregå enten ved at angive direkte på kommandolinien, hvad **awk** skal foretage sig, eller ved brug af option **-f**, efterfulgt af en kommandofil, der indeholder de kommandoer **awk** skal udføre på filerne.

```
$ awk -f skrivlogin /etc/passwd
```

vil læse kommandoerne i filen skrivlogin og udføre dem på linierne i /etc/passwd.

13.3 Udskrift af bestemte felter

Udvælgelse af bestemte felter i et datasæt sker ved at benytte betegnelsen $*nr*, hvor *nr* er nummeret på feltet i datasættet.

Følgende vil udskrive felterne 1 og 6 i /etc/passwd (login og hjemmekatalog):

```
$ awk -F: '{print$1,$6}' /etc/passwd
```

Her angives med **-F:** at feltseparatoren er ændret fra defaultværdien blanktegn til ":". Denne kan også ændres inde i programmet ved at tildele den specielle variabel **FS** (Field Separator) den ønskede værdi.

13.4 Variable i awk

Som nævnt i forrige afsnit, benytter **awk** nogle specielle variable:

FS	**Felt separator**
RS	**Datasæt separator**
OFS	**Udskrift felt separator**
ORS	**Udskrift datasæt separator**
OFMT	**Udskrift format for tal**
FILENAME	**Navn på den fil, der behandles**
NF	**Antal felter i datasættet**
NR	**Datasættets nummer**
FNR	**Feltets nummer**

Default for datasætseparatoren er ny-linie, men sættes den f.eks. til tom linie, har man mulighed for at lade flere linier optræde som et datasæt.

```
$ cat whatdo
whodo ^ awk 'BEGIN {RS=""}
             /'$1'/ {print}'

$ whatdo mette
tty01B   mette     8:47
    tty01   24619    0:01 ds
    tty01   18485    0:01 menu
    tty01    2201    0:00 dsh
    tty01   24664    0:01 menu
    tty01    6188    0:01 stfile
```

Bemærk hvorledes det i **awk** er muligt at søge efter et bestemt regulært udtryk omgivet af /../

For at der ikke skal søges efter tegnene $1, er det nødvendigt at omslutte $1 med '..', så udtrykket fortolkes som en positionel parameter til scriptet.

13.5 Søgemønstre

Som anført kan der placeres et søgemønster mellem /../. Disse mønstre er regulære udtryk, som beskrevet tidligere, men udover disse kan man også angive søgekriterier for bestemte felter.

Ønsker man f.eks. at finde alle superbruger-logins i /etc/passwd (brugerid = 0) kan dette gøres med:

```
$ awk -F: '$3~/^0$/ {print}' /etc/passwd
```

eller

```
$ awk -F: '$3==0 {print}' /etc/passwd
```

Hvor der i det første eksempel benyttes et regulært udtryk, benyttes der logisk sammeligning af feltet i det andet eksempel.

Logisk sammenligning kan dannes af følgende relationer:

<	**mindre end**
<=	**mindre end eller lig med**
==	**identisk lig med**
!=	**forskellig fra**
>=	**større end eller lig med**
>	**større end**

Disse relationer kan kombineres med følgende logiske operatorer:

&&	**logisk og**
||	**logisk eller**
!	**negation**

Man kan finde de superbrugere, der ikke har hjemmekatalog i /, og udskrive deres login samt hjemmekatalog:

```
$ awk -F: '$3==0 && $6!~/^\/$/ {print $1,$6}' /etc/passwd
```

Ønsker man at angive et område, indenfor hvilket man vil operere, kan det gøres ved at skrive komma mellem søgekriterierne:

```
$ awk 'NR==5, NR==10 {print NR,$0}' filnavn
```

vil udskrive linierne mellem 5 og 10 i den angivne fil med linienumre.

$0 er en variabel, der indeholder hele datasættet (linien).

13.6 Beregninger i awk

Awk har indbygget en række regneoperationer. Således kan man direkte benytte de almindelige regneoperationer på variable:

```
ls -l ^ awk '{bytes+=$5};
        END {print "Antal bytes: ", bytes}'

ls -l ^ awk '{bytes+=$5};
        END {print "Antal filer: ", NR,
                   "Antal bytes/fil: ", bytes/NR}'
```

Bemærk hvor let det er at lægge til en variabel ved at benytte +=.

Awk kender regneoperationerne:

+	addition
-	subtraktion
*	multiplikation
/	division
%	modulus beregning

Desuden kan man, som allerede vist, tildele variable værdi, samtidig med, at man udfører en beregning på disse:

=	tildeler variable værdi
+=	adderer til en variabel
-=	subtraherer fra en variabel
*=	multiplicerer en variabel
/=	dividerer op i en variabel
%=	beregner modulus på en variabel
++	adderer 1 til en variabel
--	trækker 1 fra en variabel

13.7 Kontrolstrukturer i awk

I **awk** har man, ligesom i shell, mulighed for at styre hvorledes kommandoer skal udføres. Man skal blot være opmærksom på at strukturerne ligner dem, man benytter i C-programmering.

if (*betingelse*) *kommando*; [**else** *kommando*]

Her kan *kommando* strække sig over flere linier ved at omslutte disse med {..}.

```
ls -l ^ awk '{if ($5 >= 100000) print "Stor: ", $9;
              else print $9}'
```

13.8 Løkker

Der er i **awk** mulighed for at konstruere løkker af typerne:

```
while (betingelse) kommando
```

```
for (udtryk1; betingelse; udtryk2) kommando
```

Følgende to udtryk vil udføre præcis det samme:

```
$ awk 'BEGIN {i=0; while (i<=10) {print i; i++}}'

$ awk 'BEGIN {for (i=0; i<=10; i++) print i}'
```

Bemærk at en **awk**-kommando kan bestå af flere kommandoer, hvis disse står i {..}

13.9 Arrays

Et array (eller en dimensioneret variabel) er en mængde af variable, som opfattes som en helhed. F.eks. et array **VAR** kan bestå af 5 variable:

VAR[1], VAR[2], VAR[3], VAR[4], VAR[5]

De 5 variable kan være nummereret 1,2,3,4,5 eller på anden måde.

I **awk** behøver man ikke definere arrays, inden man anvender disse. Dette kan være en fordel, hvis man ikke på forhånd ved hvilken længde array'et skal have.

En anvendelse af arrays er til at gemme variable, der så senere skal behandles, måske udskrives i **END** delen.

```
$ awk '{linie[NR] = $0};
  END {for (i=NR; i!=0; i--) print linie[i] }' /etc/passwd
```

Dette lille eksempel udskriver linierne i /etc/passwd i omvendt rækkefølge.

En speciel snedig form for arrays i **awk**, er de associative arrays, hvor man ikke benytter tal, men f.eks. navn for at referere til arrayets elementer.

Ønsker man således en liste over antallet af processer, som de enkelte brugere har i gang, kan dette gøres meget simpelt ved at benytte et array, hvor elementerne refererer til brugernavnet (felt 1 i en ps -ef):

```
$ ps -ef ^ awk '{brug[$1]++};
           END {for (i in brug) print i, brug[i]}'
```

Bemærk, hvorledes man kan få **for**-løkken til at gennemløbe arrayets elementer. Rækkenfølgen, de gennemløbes i, er rimeligt uorganiseret.

13.10 Indbyggede funktioner

Awk indeholder en lang række indbyggede funktioner, hvoraf nogle blot skal omtales i det følgende:

length(*variabel*)	**returnerer længden af variablen**
substr(*variabel,n,m*)	**tager en delstreng af variablen udfra n,m angivelsen**
split(*variabel,array,separator*)	**opdeler variablen i arrayet udfra separatoren**

13.10.1 length

Funktionen **length** returnerer længden af variablen, som nedenstående eksempel viser:

```
$ awk -F: '{if length($1) > 6)
                print "Lang login:", $1}' /etc/passwd
```

13.10.2 Substr

Med **substr** er det muligt at udvælge bestemte dele af en variable.

Den fuldstændige syntaks er:

substr(*variabel,n,m***)**

hvor *n* angiver, hvorfra der skal startes, og *m* angiver det antal karakterer, der skal udvælges. Hvis ikke *m* er angivet, udvælges til slutningen af variablen.

```
$ awk '{print substr($0, 1, 40)}' /etc/gettydefs
```

udskriver de første 40 tegn i hver linie af /etc/gettydefs.

13.10.3 Split

Split benyttes til at splitte variablen op i elementer efter hver forekomst af separatoren, og placere dem i det angivne array.

split(*variabel,array,separator*)

Hvis ikke, der angives en separator, benyttes værdien af **FS**, felt-separatoren.

Værdien af funktionen er antallet af elementer.

```
$ awk -F: '{ant=split($6,katalog,"/");
            print "Hoved: ", katalog[2], "Hjem: ",katalog[ant]}'
            /etc/passwd
```

Bemærk her, at der benyttes **katalog[2]**, da **katalog[1]** vil være inden den første forekomst af separatoren, og det vil i dette tilfælde sige, at den er tom.

13.11 Formateret udskrift

For at opnå en pænere udskrift end den man får ved at benytte **print** kommandoen, har man i **awk** mulighed for at benytte en kommando **printf**, der fungerer på samme måde som i **C**. Når denne kommando benyttes, er det nødvendigt at angive et udskrivningsformat efter syntaksen:

printf *format, udtryk1, udtryk2, ...*

hvor der i *format*-specifikationen står, hvorledes man ønsker de efterfølgende udtryk udskrevet.

Nedenfor følger et par eksempler til illustration af udskrivning.

En ting, man altid bør huske, når man benytter **printf**, er at den ikke selv udskriver et linieskift, men at dette skal specificeres med **\n** i formatet.

13.11.1 Udskrift af tekst

Kommandolinien:

```
$ awk -F: '{printf "%s \t %s \n", $1, $6}' /etc/passwd
```

vil udskrive login og hjemmekatalog i 2 kolonner med en tabulering mellem, mens kommandolinien

```
$ awk -F: '{printf "%10s \t %s \n", $1, $6}' /etc/passwd
```

vil udskrive det samme, hvor login højrestilles i et felt, der er nøjagtigt 10 karakterer langt.

```
$ awk -F: '{printf "%-10s \t %s \n", $1, $6}' /etc/passwd
```

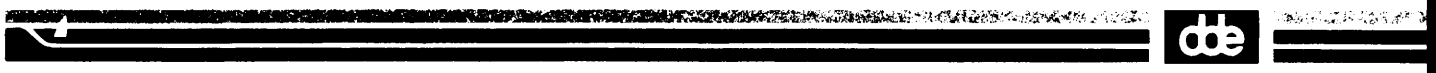
vil udskrive det samme igen, men denne gang med login venstrestillet i et felt af længde 10.

Man kan udskrive en ledetekst på følgende måde:

```
$ awk -F: '{printf "Her er login: %10s \t %s \n", $1, $6}'
          /etc/passwd
```

Og antallet af tegn i det første felt (login) kan begrænses til 3 ved at skrive:

```
$ awk -F: '{printf "%.3s \t %s \n", $1, $6}' /etc/passwd
```



13.11.2 Udskrift af tal

Når der skal udskrives tal, kan dette foregå med angivelse af feltets længde, præcision samt for heltal, hvilket talsystem, der skal udskrives i.

Talsystemer:

d	**decimal**
o	**oktalt**
x	**hexadecimalt**
f	**flydende tal (f.eks. 37.3902)**
e	**eksponentiel notation (f.eks. 7E3 svarer til 7000)**

Formatet for taludskrivning kan angives med konstruktioner af typen:

%m.n*talsystem*

Her specificerer **m** den minimale feltbredde, hvor der lige som ved tekststrenge kan angives et "-" for venstrestilling.

Tilsvarende betegner **n** den præcision, hvormed tallet skal udskrives. Dette er for heltalssystemer det minimale antal tegn, der skal udskrives, og for flydende tal og eksponentiel notation, antal tegn efter decimalpunktum.

Nedenstående tjener som eksempler på taludskrivning:

```
$ awk 'BEGIN {printf "%d \t %f \n", 30, sqrt(200)}'
30       14.142136

$ awk 'BEGIN {printf "%.4d \t %-10.2f \n", 30, sqrt(200)}'
0030     14.14

$ awk 'BEGIN {printf "%o \t %e \n", 30, sqrt(200)}'
36       1.414214e+01

$ awk 'BEGIN {printf "%.3x \t %1.2f \n", 30, sqrt(200)}'
01e      14.14

$ awk 'BEGIN {printf "%-x \t %1.2e \n", 30, sqrt(200)}'
1e       1.41e+01

$ awk 'BEGIN {printf "%f \t %d \n", 30, sqrt(200)}'
30.000000        14

$ awk 'BEGIN {printf "%e \t %20.15f \n", 30, sqrt(200)}'
3.000000e+01       14.142135623730951
```

Reference:
Aho, Weinberg & Kernighan: Awk, The Programming Language.
System V, Programmers Guide, kapitel 4.

Opgave 12

Formål: - at blive fortrolig med sproget awk

1. Prøv de anførte eksempler.

2. Fremstil et program, der udskriver login, hjemmekatalog samt diskpladsforbrug for de brugere, der ikke har system-administrativt funktion. (Userid>100)

   En unix-kommando udføres fra awk ved at skrive:

   system(*kommandolinie*)

3. Lav et program, der kan udskrive filer med linier længere end 80 tegn, således at de foldes.

   Lav evt. programmet, så det ikke skiller midt i et ord.

4. Lav et program, "imellem", der kan udskrive et antal linier før og efter et søgemønster.

   Husk at der måske kan være flere liner, der indeholder mønsteret.

Opgaven fortsættes...

5. Lav et program, der udskriver en statistik over de fejl, der er rapporteret i errloggen.

6. Lav tilsvarende statistik programmer til de øvrige administrative logfiler (cronlog, sulog, ...)

7. Fremstil et samle/usamle programpar.

   Det ene program skal samle en række tekstfiler sammen til en fil, og tilsvarende skal det andet program kunne adskille den samlede file i sine bestanddele.

   Tips: Leg lidt med variablen FILENAME.

Dansk Data Elektronik A/S
Herlev Hovedgade 199
DK 2730 Herlev
Tel.: (- 45) 42 84 50 11
Fax.: (- 45) 42 84 52 20