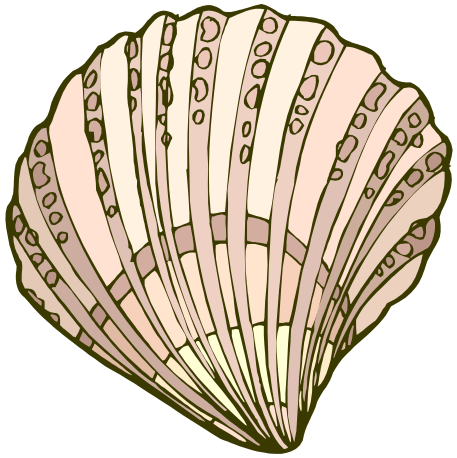
Shellen og dens funktion



10 Shellen og dens funktion

Hvad er en shell?

- ***Shellen*** er et program, der ***kommunikerer med brugerne*** på ethvert UNIX/Linux-system.
  Den er også kaldet ***kommandofortolkeren***
- ***Shellen*** startes automatisk når man er logget ind
  og den bliver der indtil man logger ud igen
- ***Shellen*** er også et ***programmeringssprog***
- ***Shellen*** indeholder diverse ***kommandoer og mekanismer***
- ***Shellen*** er vores ven
  Den giver os lov til at ***udtrykke os kort*** – og så skal den nok gøre ***resten af arbejdet for os***
- ***Shellen*** bruges til at ***udføre kommandoer interaktivt*** eller som ***batch-jobs***

10 Shellen og dens funktion

Shellen udfører opgaver gennem :

Interaktiv kommando-afvikling:

- ***Brugeren indtaster*** kommandoer på kommando-linjen (ud for prompten), hvorefter ***shellen læser og fortolker*** de enkelte dele inden den ***udfører kommandoen***.

Batch kommando-afvikling via shell-scripts:

- Brugerens ***kommandoer*** indtastes ***i en fil***.
- Når shellen bedes om at udføre filen (***shell-scriptet***), udføres kommandoerne.

10 Shellen og dens funktion

Forskellige shells

- **Bourne shell** (sh) blev skabt af Stephen Bourne (AT&T) og findes i en eller anden version på *enhver UNIX-platform*.
- **C shell** (csh) er skabt af Bill Joy (Berkley, SUN), fordi der var mange ting, han savnede i Bourne Shell.
  - Er ikke bagudkompatibel imod Bourne Shell syntaxen. AT&T var derfor meget utilfredse med C shell.
- **Korn shell** (ksh) er skabt af David Korn (AT&T) ud fra hans erfaringer med fordele/ulemper ved C shell.
  - Korn shell er 100% bagudkompatibel med Bourne shell.
- **Bash** (bash) "Bourne Again Shell" er **standard shell i Linux** og er GNU-versionen af *Bourne shell*.
- **TC shell** (tcsh) er en udvidet **C shell**

10 Shellen og dens funktion

Shellens indbyggede mekanismer:

- **Wildcards**
- **Redirigering**
- **Pipes**
- **Baggrundsprocesser**

(Mekanismerne gennemgås i de følgende slides)

Shellens wildcards til filnavne

*	Repræsenterer enhver sekvens af tegn – inkl. "nulstrengen".
?	Ét og netop ét tegn.
[a-z]	Repræsenterer ét vilkårligt tegn indeholdt i den angivne mængde (a til z).
[AYay]	Repræsenterer ét vilkårligt tegn indeholdt i den angivne mængde {AYay}.
\	Får shellen til at undlade fortolkning af det umiddelbart efterfølgende tegn.

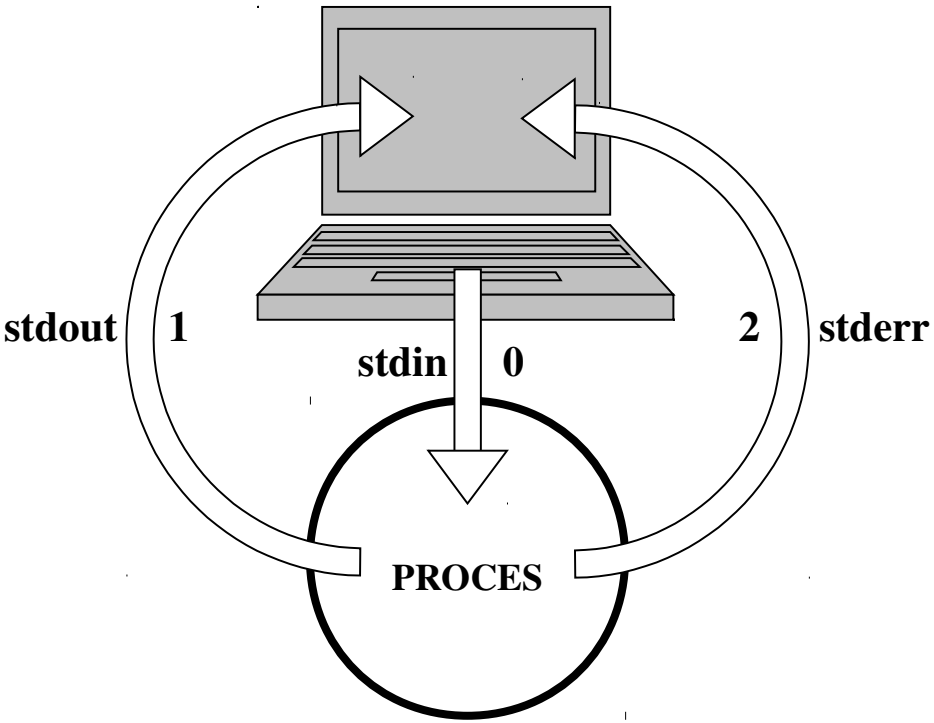
```
$ ls -l fil*  ; ls -l fil? ; ls -l fil[1-3] ; ls -l fil[x1]
$ ls -l alf* \; fil*           $ ls -l alf* ; fil*
```

Lav øvelser 10.1

10 Shellen og dens funktion

Standard redirigering

Default bliver såvel
standard **output** (*stdout*) som
standard **error** (*stderr*)
returneret til skærmen




```
$ ls -l alf* \; fil*      #(Læg mærke til at denne kommando
                               får returneret såvel stdout
                               som stderr til skærmen)
```

Redirigering - andre muligheder

- Det er muligt at redirigere output til et andet terminalvindue med write kommandoen:

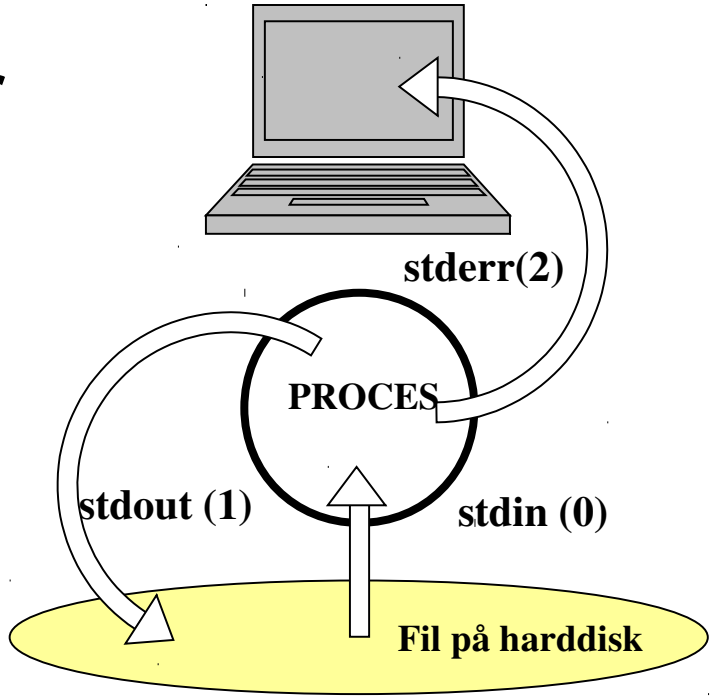
```
$ tty
  /dev/pts/1
 (Åbn et nyt terminalvindue)
$ who
       bruger1  pts/0        2018-10-08 13:19
       bruger1  pts/1        2018-10-08 13:19

$ echo 'hej med dig' > nyfil #(Redirigering af stdout til nyfil
                filen oprettes eller indhold slettes først)

$ write bruger1 pts/0  < nyfil
    #(Input fra nyfil skrives til "bruger i teminalvindue pts/0")
```

10 Shellen og dens funktion

Redirigering - andre muligheder



Her læses **input fra en fil**

og

output skrives til en fil,

filen oprettes eller tømmes først!

```
$ ls -l > liste_over_filer   #(fil oprettes med en liste)
$ grep kat < liste_over_filer > katfil
  # (Filen "liste_over_filer" er input til grep kommandoen. -
     Output skrives til filen "katfil"
```

Redirigering – med tilføjelse til en fil

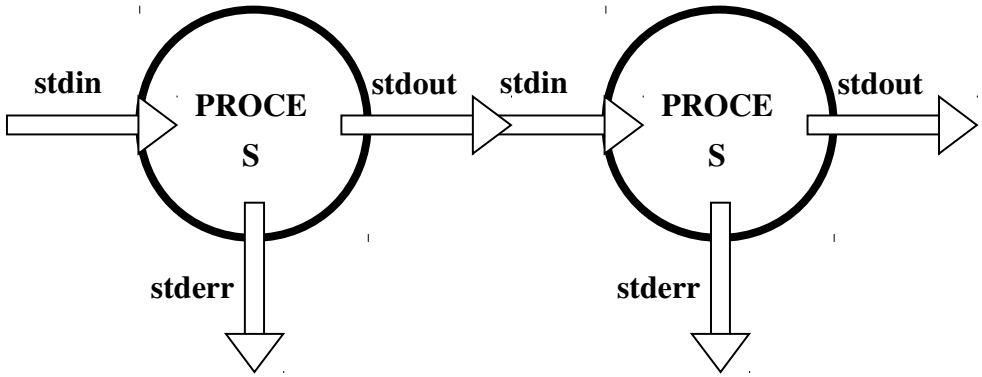
- Ved almindelig redirigering af **stdout (>)** oprettes den ønskede fil, eller hvis filen eksisterer i forvejen, slettes det oprindelige indhold (*trunkeres til 0 bytes*)
- I stedet kan man vælge at bevare filens oprindelige indhold, og tilføje output **stdout (>>)**
  i *forlængelse af filens eksisterende data*.

```
$ ls > fil #(Redirigering til stdout)
$ cat fil
$ echo 'Her er noget ekstra indhold' >> fil #(tilføjer indhold)
$ cat fil
```

10 Shellen og dens funktion

Pipelines

- En ***pipe*** er en måde at redirigere stdout fra én proces til stdin hos en anden proces
- Symbolet for en ***pipe*** er **|**



Eksempel – Sortér filliste efter størrelse og vis listen sidevis.

```
$ ls -l | sort -nk5 | more
```

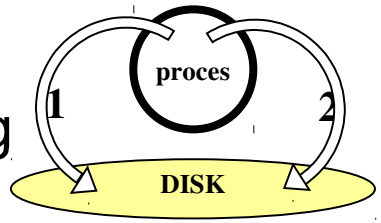
10 Shellen og dens funktion

Redirigering af fejlmeddelelser

- Alle UNIX/Linux kommandoer skriver deres ***fejlmeddelser*** til ***stderr (2)***

```
$ ls -l ingenfil 2> errfil #(Redirigerer af fejl til errfil)
$ cat errfil
```

Redirigering af både stdout og stderr



- Man kan også lade både ***almindeligt output(1)*** og ***fejlmeddelser(2)*** blive skrevet ***til hver sin fil***:

```
$ find /home/bruger -name txt -print > findfil 2> errfil
$ cat findfil  #(stdout(1) er læst ind i filen "findfil")
  (Man behøver ikke skrive 1 for at redirigere stdout)
$ cat errfil  #(stderr(2) er læst ind i filen "errfil")
```

Lav øvelser 10.2 og 10.3

Baggrundsprocesser

- Almindeligvis, når en proces (et program) startes, kan man ikke benytte terminalen, før dette job er afsluttet.
- Når en kommando startes som en baggrundsproces, vender shellen straks tilbage med en ny prompt.
- Baggrundsprocessen udføres parallelt med, at shellen kan udføre en ny kommando.

  Eksempel:

```
$ sort kassebon > sorteret_kassebon &  #(Det afsluttende &
                                         sørger for baggrundsproces)
      [1] 15440                       #(jobnummer og procesnummer)
$                                     #(promten vender starks tilbage)
$ tail sorteret_kassebon
```

Nohup – Baggrundsprocesser

(Gælder kun ældre Linux og Unix versioner)

- Kommandoen ***nohup*** kan benyttes i forbindelse med baggrundsprocesser.
- **nohup** benyttes til at starte en baggrunds-kommando, der *ikke* må terminere, når brugeren logger ud (NOHangUP).

Eksempel:

```
$ nohup sort fil > sortfil &
```

- Nu kan der logges ud, uden at sorteringen forstyrres.
- Hvis brugeren logger ud, har baggrundsprocessen ingen forældreproces mere. Proces 1 (***init***) går da ind og overtager forældrerollen.

Afbrydelse af baggrundskommandoer

- Kommandoen ***kill*** kan bl.a. benyttes til at afbryde et baggrundsjob.

  Syntax:

  - kill [-signalnummer] processnummer *eller* kill [-signalnummer] %jobnummer

```
$ sleep 100 &          #(Process startes i baggrunden)
      [1] 15824
$ sleep 200 &          #(Process startes i baggrunden)
      [2] 15829
$ ps                   #(list egne processer)
$ kill 15824           #(kill processnummer 15824)
$ kill %2              #(kill jobnummer 2)
$ kill -L              #(Viser de kill signaler man kan sende)
```

Hvis en proces *ikke* terminerer på ovenstående mordforsøg må skrappere midler tages i brug. Signal 9 er "*stærkt dødeligt*" og bør kun bruges som *sidste* udvej. Eksempel `$ kill -9 7913`

10 Shellen og dens funktion

Variabler

Værdien af shell-variablerne vises med kommandoen ***echo $[navnet på variablen]***.
Eksempler:

```
$ echo $HOME          #(Viser stien til dit hjemmekatalog)
$ echo $PATH          #(Viser de kataloger shellen først
                        søger i, når et program skal køres)
$ echo $PS1           #(Titlen på Shell-promt 1)
```

Du kan oprette og ændre variabler på følgende måde:

```
$ STED='Ringsted'   #(Opretter den midlertidige variabel STED)
$ export STED       #(Variablen STED kopieres til environment)
$ echo $STED
$ env               #(Viser invironment variabler)
```

10 Shellen og dens funktion

Alias

Man kan give et alias til en kommando:
Eksempler:

```
$ alias CD='cd ~/Desktop #(CD med stort leder dig til desktop)
$ alias gx='chmod +x'    #(gx = gør eksekverbar)
$ type ls              #ls is aliased to `ls -color=auto'
                         Standard i Ubuntu Mate
```

unalias:

```
$ unalias CD  #(fjerner alias for CD)
```

Lav øvelser 10.4

10 Shellen og dens funktion

Shellscript

- Et ***shellscript*** er en fil med UNIX/Linux-kommandoer også kaldet en *(**batch-fil**)*.
  Alt, hvad man kan skrive på kommandolinjen, kan man skrive i et shellscript.
- Eksekverbare ***shell-scripts*** bør indledes med en linje, der indeholder oplysning om, hvilken shell der skal anvendes til fortolkningen af det.

Eksempel:

```
#!/bin/sh     Hvis det er skrevet til Bourne-shell
#!/bin/csh    Hvis det er skrevet til C-shell
#!/bin/ksh    Hvis det er skrevet til Korn-shell
#!/bin/bash   Hvis det er skrevet til Bash
```

10 Shellen og dens funktion

Shellscript

Eksempel $ vi mit_script.sh :

```
#!/bin/bash
 # Dette er et shell-script
 # ved navn goddag.bash
 # Scriptet udføres ved at
 # skrive navnet og trykke
 # RETUR.
date
echo
echo Kære $LOGNAME
echo
echo Indloggede personer:
echo
who
```

Sådan udføres et shellscript

Et ***shellscript*** kan udføres på 3 måder:

Eksempler:

```
$ bash mit_script.sh                #(Manuelt fra barne-shell)
$ . /home/bruger1/mit_script.sh     #(Med fuld stiangivelse)
$ . ./mit_script.sh                 #(Med relativ stiangivelse)
```

Hvis du får : ”*Permission denied*” når du kører scriptet med stiangivelse,
er det nødvendigt at gøre scriptet eksekverbart:
(forklares senere)

```
$ chmod +x mit_script.sh
```

Shell quoting

Quoting betyder anvendelse af ***citationstegn***. Citationstegn i UNIX kaldes normalt ***plinger***

" " ***Dobbeltplinger*.**
Beskytter *alt* mod shellens ekspansion ***undtagen* \ , $-udtryk og bagplinger**.
Eksempelvis er wildcards og alle mulige andre specialtegn beskyttet og vil derfor *ikke* blive fortolket på sædvanlig vis.

' ' ***Enkeltplinger*.**
Beskytter ***alt*** mod shellens ekspansion uden undtagelse
dvs. også $-tegn, "-tegn, \ m.m.

**** ***Backslash*** eller *escape*-tegn
Beskytter efterfølgende tegn mod fortolkning af shell

$( ) ***Kommandosubstitution***
Bevirker, at shellen udfører den kommando, der står imellem parenteserne og derefter erstatter *hele udtrykket* med outputtet fra den udførte kommando.

` ` **Bagplinger** kan anvendes til at gemme kommandoer i variabler ex:
LISTE=`ls` prøv herefter: **echo $liste**

Shell quoting

Eksempler på shell quoting:

```
$ echo "Dette er en liste:  $(ls fil.*)"
  Dette er en liste:  fil.a
  fil.b

$ echo 'Dette er en liste:  $(ls fil.*)'
  Dette er en liste:  $(ls fil.*)

$ echo 'Backslash med enkeltplinger  \' "og med dobbeltplinger: \"
  >
```

Lav øvelse 10.5 og 10.6