# Betinget eksekvering
# og
# logiske tester i shell

# Betinget eksekvering*?

- Programmet utfører operasjon(er) bare hvis en logisk betingelse er *sann*
- Bash tilbyr to *kontrollstrukturer* for å kunne gjøre betinget eksekvering:

  `if` `case`

- Både `if` og `case` krever at vi kan gjøre logiske tester i shellet for å sjekke om noe er sant eller usant

*: Kalles også for *seleksjon* i programkoden

# Exit-status og variabelen `$?`

- For å forstå hvordan logiske tester håndteres i shell-programmer, må vi kjenne til begrepet “exit-status”
- Alle Linux-kommandoer returnerer en exit-status når de avslutter – en tallverdi som lagres i shellet
- Standard er at exit-status lik 0 (null) betyr “alt i orden”
- En exit-status > 0 brukes til å signalisere at noe er feil
- Vi kan sette exit-status for våre egne shellprogrammer med termineringskommandoen `exit` *`verdi`*
- Exit-status for siste utførte kommando ligger alltid lagret i shellet. Denne verdien kan hentes frem med `$?`

# `test` – Logiske tester i shellet

- Kommandoen `test`:
  - Beregner logiske utrykk
  - Returnerer med exit-status 0 hvis uttrykk er `true`
  - Returnerer med exit-status 1 hvis uttrykk er `false`
- Syntaks:

  `test` *`logisk-uttrykk`*
- For å få ryddigere kode, tillater Bash alternativt denne syntaksen for å kjøre kommandoen `test`:

  `[` *`logisk-uttrykk`* `]`

# Logiske uttrykk kan skrives på flere måter *

1. Uttrykket skrives inne i `[]` (enkle brackets):
   - Syntaksen som er angitt i `man test`
   - POSIX-standard
   - Vi skal *bare* bruke denne skrivemåten
2. Uttrykket skrives inne i `[[]]` (doble brackets):
   - Utvidet syntaks og funksjonalitet, med regulæruttrykk
3. Uttrykket skrives inne i `(())` (doble paraneser):
   - Egentlig beregning av aritmetiske uttrykk i Bash, som også kan beregne logiske true/false uttrykk

* Læreboken viser alle tre varianter og er litt uryddig i syntaksbruken

# Sammenligning og test av tekststrenger

| Logisk uttrykk | Uttrykket er *true* hvis: |
|---|---|
| `[ s1 = s2 ]` | De to strengene er like |
| `[ s1 != s2 ]` | De to strengene ikke er like |
| `[ -n s ]` | s har lengde større enn 0 |
| `[ s ]` | s har lengde større enn 0 |
| `[ -z s ]` | s har lengde lik 0 (tom) |

Merk at syntaksen her er "veldig streng", det *må* være en space mellom uttrykk og `[ ]`, og mellom operander og operatorer

# Sammenligning av to tallverdier

| Logisk uttrykk | Uttrykket er *true* hvis: |
|---|---|
| `[ n1 -eq n2 ]` | De to tallene er like |
| `[ n1 -ne n2 ]` | De to tallene ikke er like |
| `[ n1 -gt n2 ]` | `n1` er større enn `n2` |
| `[ n1 -ge n2 ]` | `n1` er større eller lik `n2` |
| `[ n1 -lt n2 ]` | `n1` er mindre enn `n2` |
| `[ n1 -le n2 ]` | `n1` er mindre eller lik `n2` |

# Logiske tester knyttet til filer

- Kan også teste egenskaper ved filer ved å bruke logiske uttrykk og kommandoen `test` i Linux
- Nyttig for unngå run-time feil i shellprogrammer:
  - Kan teste om filer eller kataloger finnes før vi prøver å gjøre noe med dem
  - Kan teste om filer er av riktig type
  - Kan sjekke om vi har rettigheter til filer før vi forsøker å lese, skrive eller eksekvere
- Gir også mulighet til å sammenligne metadataene for filer (tidsstempel, inodenummer etc.)

# Sammenligning og testing av filer *

| Logisk uttrykk | Uttrykket er *true* hvis filen: |
|---|---|
| `[ -e fil ]` | Eksisterer |
| `[ -f fil ]` | Er en vanlig (regulær) fil |
| `[ -d fil ]` | Er en katalog |
| `[ -r fil ]` | Er lesbar (tilsvarende for `-w` og `-x`) |
| `[ -O fil ]` | Eies av deg/bruker (`-G` for gruppe) |
| `[ fil1 -ef fil2 ]` | `fil1` og `fil2` er samme fil (inode) |
| `[ fil1 -nt fil2 ]` | `fil1` er nyere enn `fil2` |
| `[ fil1 -ot fil2 ]` | `fil1` er eldre enn `fil2` |

*: Det finnes flere logiske operatorer for filer, se læreboken og `man test`

# Sammensatte uttrykk: NOT, AND og OR

| Logisk uttrykk | Uttrykket er *true* hvis: |
|---|---|
| `[ ! u ]` | Det logiske uttrykket `u` er *false* |
| `[ u1 -a u2 ]` | Begge uttrykkene `u1` og `u2` er *true* |
| `[ u1 -o u2 ]` | Minst ett av `u1` og `u2` er *true* |
| `( )` | Brukes til gruppering/presedens |

Eksempel:

```
[ ! \( u1 -o u2 \) -a \( u3 -o u4 \) ]
```

# `if` - setningen

- Syntaks:

```
if TEST-KOMMANDO; then FLERE-KOMMANDOER; fi
```

- Virkemåte:
  - Hvis *TEST-KOMMANDO* returnerer med exit-status lik 0, så utføres *FLERE-KOMMANDOER* , ellers utføres de ikke

# Vanlig skrivemåte for bruk av `if` i shellprogrammer

- Dropper vanligvis semikolon...
- ...skriver i stedet hver del av `if` - setningen på hver sin linje, med indentering for lesbarhet
- Bruke oftest kommandoen `test` , skrevet med `[]` , som "test-kommando" ...
- ... men en *hvilken som helst* Linux-kommando kan brukes etter `if`

# `if` – eksempler

- Med en "vanlig" Linux-kommando i testen:

```
if who | grep -q 'janh'
then
   echo "Hacker alert!"
fi
```

- Med bruk av kommandoen test, skrevet med `[]` :

```
if [ $USER = 'janh' ]
then
   echo "Hacker alert!"
fi
```

# if – then – elif – else

```
#!/bin/bash
read -p "Og De heter? " guest_name

if [ $guest_name = Jan ]
then
  echo "Hjertelig velkommen"
elif [ $guest_name = Christian ]
then
  echo "Vennligst forlat lokalet umiddelbart"
  exit 1
else
  echo "Står De på gjestelisten?"
fi
exit 0
```

# Noen flere eksempler med `if`

- Skrive ut brukere med flest logins
- Interaktiv lesing og testing av verdier
- Skrive ut fil eller la bruker taste inn ny fil
- Bestemme det største av tre tall

# `case` - setningen i shellprogrammer

- `case`-setningen er alternativ måte å skrive en flergrenet `if`-test på
- Hvert "case" er en liste med verdier som matches mot et gitt uttrykk
- For hvert "case" angis det hvilke kommandoer som skal kjøres
- Tilsvarende konstruksjoner finnes i svært mange programmeringsspråk, gir ofte ryddigere kode

# `case` - syntaks

```
case expression in
  list1)
    action(s)
    ;;
  list2)
    action(s)
    ;;
  .
  .
  .
  *)
    action(s)
    ;;
esac
```

- Kan bruke regulæruttrykk i angivelsene av `list`
- `*)` angir det som utføres hvis ingen av de andre angitte case matcher `expression`
- Hvis hvert case avsluttes med `;&` i stedet for `;;`, vil eksekveringen fortsette med neste case, i stedet for å hoppe til `esac`

# case - eksempel

```
read -p "Do you agree to the licensing\
terms of this program? " answer

case $answer in
  [yY]|[yY][eE][sS])
    ./program
    ;;
  [nN]|[nN][oO])
    echo "I cannot run the program"
    exit 1
    ;;
  *)
    echo "Invalid response"
    exit 1
    ;;
esac
exit 0
```

# && og || : Betinget eksekvering

- Kan også bruke spesialtegnene && og || i Bash til å gjøre seleksjon i shellprogrammer
- Spesialtegnene settes *mellom* to kommandoer:

  && Utfør neste kommando *bare* hvis forrige OK

  || Utfør neste kommando *bare* hvis forrige *ikke* OK

- "OK" betyr at kommando terminerer med exit-kode lik 0
- && og || er egentlig bare "forkortede if-tester"

# Eksempler: Betinget eksekvering

```
grep -q Levon filnavn && echo ":-)"

if grep -q Levon filnavn
then
  echo ":-)"
fi


grep -q Levon filnavn || echo ":-("

if !grep -q Levon filnavn
then
  echo ":-("
fi
```

# Komprimering av kode med && og | |

```
if [ -f $filnavn ]
then
  rm filnavn
else
  echo "No regular file $filnavn found"
  exit 1
fi
```

```
[ -f $filnavn ] && rm $filnavn || { echo "No\
regular file $filnavn found"; exit 1; }
```