# DAT 103
# Datamaskiner og operativsystemer (Computers and Operating Systems)

## UNIX – Deler av denne er obligatoriske

Dette er en øvelse hvor de siste oppgavene er obligatoriske med innlevering. Hjelp til øvingen gis på laben i uke 35 til 37. De første 14 oppgavene gir en god trening i bruk av bash. Krav til de obligatoriske delene av øvelsen:

- Dere kan arbeide sammen i grupper på inntil 4 personer.
- Gruppen leverer samlet på Canvas.
- Innleveringsfrist er **kl 12:00, 14 september**.
- Dere må klare å løse alle de obligatoriske oppgavene.
- For hver av oppgavene skal følgende leveres:
  - Tekstfil med kildekode til skallprogram.
  - Resultat av kjøring.

Merk at det er oppgitt en liste av nyttige ting (tips) for hver oppgave. Prøv å finne ut av ting, enten ved å slå opp i lærebok, slå opp i man, eller søk på verdensveven i denne rekkefølgen, før dere spør studentassistentene eller foreleser.

UNIX-pensum i dette kurset er bruk av BSD UNIX og bash-kommandoer selv om de fleste operasjoner kan gjøres fra skrivebordet og grafiske programmer. Grunnen er at UNIX og Linux ofte benyttes på tjenere. For en tjener kan det være unødvendig å bruke disk-plass og hukommelse på å kjøre et vindussystem.

En annen grunn for å skrive kommandoer er å kunne automatisere systemoppgaver. Da er ikke grafiske system-program egnet. Vi skal derfor arbeide i terminalvinduer og utføre alle operasjoner derfra.

# Innhold

## Oppgave 1 – Enkle oppgaver om kommandoer og litt forklaring (ikke obligatorisk)

I terminalvindu, skriv `ls` og så linjeskift.

Hent fram igjen kommandoen i terminalvindu ved å trykke opp-piltasten. Vent med å utføre kommandoen. Legg først til teksten `-l` (bokstaven `l` er en liten L) slik at teksten i terminalvindu blir `ls -l`. Utfør nå kommandoen.

Utfør `man ls` og se hva «`-l`» betyr. Vi avslutter lesing av manualsider ved å trykke «`q`».

Utfør følgende to kommandoer og sammenlign resultatet

```
ls -a -l -h
ls -alh
```

En opsjon kan ta verdier som må følge like etter bryteren, f.eks. `ls -lw40 -a`. Utfør denne og virkningen

Vi skal nå opprette en mappe *tmp* under hjemmekatalog. Det gjør vi ved å utføre:

```
mkdir tmp
```

Her er noen kommandoer som oppretter filer:

```
echo "En fil" > aa.txt
echo "Enda en fil" > bb.txt
echo "Og enda en fil" > AA.txt
```

Setningene over lager tre filer. Sjekk innholdet med:

```
cat aa.txt
cat bb.txt
cat AA.txt
```

Utfør nå følgende kommandoer og forsøk å finne ut hva som skjer:

```
ls aa.txt bb.txt
ls -l aa.txt bb.txt    # l er en liten L
ls -l *.txt
```

Tegnet # angir kommentar. All tekst fra # og ut linjen blir ignorert av kommandotolker.

Vi ser at UNIX skiller på store og små bokstaver. Det gjelder også for kommandoer. Forsøk å skrive `LS` i stedet for `ls`.

Du har sikkert lagt merke til at mappeskillemerket i UNIX og Linux skiller seg fra Windows. UNIX og Linux bruker symbolet «/», det samme som brukes i URL-er for å spesifisere mappestruktur på webtjener.

Tabulatortasten kan brukes til å fylle ut siste del av kommandoer og filnavn. Dette fungerer hvis det vi har skrevet entydig kan identifisere en kommando eller fil. Dersom flere muligheter finnes vil et nytt klikk på tabulatortasten liste alle muligheter.

For mange programmer vil bash vite hvilke type filer et program kan åpne. Fullføring av filnavn vil kun velge blant filene som programmet kan åpne. Tabulatorfullføring vil ofte også fungere på opsjoner til program. Tabulatorfullføring av filnavn fungerer kun hvis kommandotolker forventer et filnavn.

Fyll inn følgende i et terminalvindu:

```
ged<TAB>
```

Hva skjer? Her er «`TAB`» tastaturet sin tabulatortast.

I mappen der vi har laget de 3 filene, utføre følgende:

```
ged<TAB> a<TAB>
man<TAB><TAB>
```

Dersom vi skriver feil på kommandolinjen kan den redigeres. Vi kan navigere fram og tilbake på linjen, nye tegn kan legges til og tegn kan slettes både med *delete* og *backspace* knappene på tastaturet. Vi kan også legge til tekst kopiert med musen.

Kommandolinjen kan også redigeres og navigeres i ved å benytte `Ctrl`-kombinasjoner. Disse er ikke pensum, men kan gjøre editering enklere. De interessert kan selv sjekke manualsiden til *readline*. Finn avnsittet *EDITING COMMANDS*.

På UNIX har vi flere hjelpesystemer. Vi skal nå se på manualsystemet.

Dersom vi ønsker hjelp for en kommando kan vi skrive `man «kommando»`. I starten kan det kanskje være vanskelig å forstå manualsidene. De har alle samme struktur, så etterhvert blir de nok mer forståelige.

Manualsider er oppdelt i seksjoner, normalt med et nummer som navn. Dersom ikke seksjon oppgis vil avsnittet med lavest nummer vises. Vi kan angi en annen seksjon som argument til *man*, f.eks. vil `man 3 exec` gi 3 avsnitt i manualsidene for *exec*. Avsnitt 1 viser hvordan kommandoer skrives i terminalvinduet. Avsnitt 2 og 3 gir hjelp om systemkall (mer om dette i OS-delen av kurset).

For å navigere til neste side i en manualside kan vi trykke mellomrom-tasten. Programmet som benyttes på lab-en for å vise innholdet i manualsider heter *less* (men vi kan velge andre program, f.eks. programmet *more*).

Kommandotolkerspråket vi benytter heter *bash*. Det finnes mange alternativer, men *bash* er standard valg for CentOS, Ubuntu og Fedora.

Kommandoene vi skriver er stort sett navn på systemprogrammer. Disse er uavhengige av valg av skall. Systemprogrammene finner vi normalt i mappene `/bin` og `/usr/bin`.

Hvilke skall som er tilgjengelige finner vi listet i filen `/etc/shells`. Vi kan finne ut hvilket skall vi benytter ved å lese innholdet av variabelen `SHELL`.

```
echo $SHELL
```

Variabelen `SHELL` viser hva som er standardvalget ditt. Vi har nå sett på grunnleggende bruk av UNIX. Vi er nå klar for å studere litt flere kommandoer.

```
Date
```

Bruk manualsystemet og les om *date*-kommandoen.

Utfør følgende kommandolinje:

```
date +"%A"
```

Bruk *cal* og list ut dagene i oktober i år.

Bruk manualsystemet og les om cd. Hjelp om *pwd*-kommandoen finner vi ved å skrive `help pwd`. Det finnes hjelpeside om *pwd* også i manualsystemet, men denne hjelpen stemmer ikke. Grunnen til det skal vi undersøke i en senere øvelse.

Finn ut hva følgende kommandolinjer gjør

```
cd ..
pwd
cd ../..
pwd
cd .././..
pwd
cd /usr/local
pwd
ls
```

Når vi senere skal lage skallprogram, må vi benytte et verktøy for å skrive og lagre tekst. Med skrivebordet GNOME følger teksteditoren *gedit*. Mer avanserte teksteditorer er bla. *emacs* og *vi*, men disse kan være vel kompliserte for nye brukere. Enklere teksteditorerer er f.eks. *nano*, *joe* og *pico*.

## Oppgave 2 – Kommandoer for å få hjelp (ikke obligatorisk)

UNIX har har flere kommandoer for å få hjelp:

- *man* gir hjelp om kommandoer, f.eks. `man ls` vil gi hjelp om *ls*.
- *info* er et hjelpesystem som gir utfyllende hjelp til mange UNIX kommandoer. F.eks. vil kommando `info` gi en oversikt over all hjelp som *info* kan tilby, mens `info ls` gir hjelp om *ls*.
- Kommandoen *apropos* returnerer alle man-sider der ordet vi oppgir eksisterer. Med man-side menes hjelpedokumentet som kommandoen man returnerer. F.eks. vil `apropos file` returnere alle man sider der ordet file opptrer i beskrivelsen av man siden.
- Kommandoene *which*, *whereis* og *type* er nyttige for å finne ut hvor/hvilke programmer en kommando utfører. Du må sammenholde informasjonen som *which* gir med *type*. Hvis *type* forteller at en kommando er «a shell builtin» betyr det at kommandoen ikke kjører et program, men at det er en kommando som er en del av bash.
- For kommandoer innebygget i skallprogrammet finner vi hjelp med kommandoen *help*, f.eks.:

  ```
  help echo | less
  ```

  Det kan være nyttig å sende utskriften fra help gjennom *less* for å få listet ut en side om gangen.
- Vi kan ofte spørre programmer og kommandoer om hjelp ved å bruke brytere som `-?`, `--help`, `-help` eller `-h`. Forsøk gjerne først `-?`.

Bruk informasjonen over til å finne svar på følgende spørsmål:

1. Når vi skriver kommando *pwd*, hva er det da vi utfører? (Er *pwd* et program?)
2. I et terminalvindu med bash utfører vi følgende kommando:

   ```
   pwd -P
   ```

   Hva angir bryter `-P` i kommandoen over? Hvordan fant du hjelp om bryteren `-P`?
3. Bruk info-systemet og finn hjelp om *ls*.

   - Hjelpesystemet *man* gir av og til mer kortfattet og kompakt informasjon enn *info*.
   - Hvis informajonen fra *man* er vanskelig å forstå kan det være lurt å sjekke hjelpeteksten fra *info*.
   - I dette kurset vil fokus være på *man*. Hjelp til kommandoer på eksamen vil være utskrift fra kommandoer sine *man* sier.
4. Hvilke manualsider omhandler *hardware*?
5. Hva gjør bash sin innebygde kommando *echo*? Finnes det et program som har omtrent samme oppgave? Hva er forskjellene på programmet og kommandoen innebygget i bash?
6. Skriv ut følgende tekst med en enkelt *echo*:

   ```
   Kurset heter DAT103
   ```

   Mellomrommene skal lages ved å bruke tabulatortegn.

## Oppgave 3 – Arbeide med filer (ikke obligatorisk)

Vi skal nå arbeide med kommandoene for å håndtere filer og navigere i filsystemet.

1. Sjekk bruken av kommandoene *mkdir* og *rmdir*. Lag følgende mappestruktur under din hjemmekatalog:

   ```
   hjemmekatalog
   +--dat103
   ```

```
+--diverse
+--ovinger
+--skallprogram
```

Naviger til mappen *diverse* og opprett en fil der, f.eks. med *touch*. Forsøk så å slette mappen ved å benytte *rmdir*. Hva skjer? Hvordan vil du gå fram for å slette mappen *diverse*. Tips: Sjekk ut kommandoen *rm*.

2. Lag en fil *ov2.txt* i mappen *ovinger*. Naviger til mappen *skallprogram*. Angi stien fram til *ov2.txt* både som et relativt- og et absolutt stinavn. Test ut svaret, f.eks. som argument til *ls*.
3. Enkleste metode for å navigere til din hjemmekatalog er å utføre kommando *cd* uten argumenter. Vi har andre alternativer, f.eks. variabelen `HOME` og også konstruksjonen ∼.

   Angi stien fram til filen *ov2.txt* ved alle de ulike metodene. Obs.: For å referere innholdet i en bash-variabel må vi benytte tegnet $.
4. Hva gjør kommandoen *tree*? Test ut resultatet fra din hjemmekatalog. Vi kan få listet de samme filene og mappene med kommandoen *ls*. Hvordan?
5. Kommando *mv* endrer navn på filer og mapper, eller flytter filer og mapper. Kommando *mv* tar som argument filer og mapper. Dersom siste argument er navnet på en eksisterende mappe flyttes de andre filene og mappene gitt som argument dit.

   I din hjemmekatalog, opprett mappen *nyttig*. I denne mappen oppretter du filene *fil1.txt*, *fil2.txt*, *fil3.txt*, *fila.txt*, *du1*, *du2*, *dua*, *duc* og *duc.txt*. Flytt så hele mappen *nyttig* inn i din mappe dat103 med kommandoen *mv*.
6. Opprett en mappe *dat103.copy*. Kopier innholdet i mappen *dat103* til *dat103.copy* ved å bruke kommandoen *cp*.
7. Ved å bruke kommandoen *mv*, endre navnet på filen *duc* til *dub*. Endre også navnet på mappen *dat103.copy* til *dat103.kopi*.
8. Lag en tekstfil med noe innhold. Tell antall ord i filen ved å benytte kommandoen *wc*. Vis innholdet i filen med kommandoene *cat*, *less* og *more*.
9. Bruk kommandoene *ls* og *wc* for å telle antall filer i mappen *nyttig*.

## Oppgave 4 – Jokertegn (ikke obligatorisk)

Jokertegn benyttes for å referere vilkårlige tegn i filnavn:

- `?`: Matcher ett enkelt vilkårlig tegn.
- `*`: Matcher ett vilkårlig antall vilkårlige tegn.
- `[]`: Matcher ett enkelt tegn som må være listet, f.eks. `[a-z]` som matcher ett enkelt tegn som kan være en bokstav fra a–z.

Konstruksjonene `?` og `*` kan også matche et punktum, hvis punktum ikke er første tegn i filnavnet.

I konstruksjonen `[]` kan vi benytte – for å angi en sekvens med tegn. Dersom vi ønsker å matche tegnet – angir vi det som første tegn i tegnlisten.

I konstruksjonen `[]` kan vi starte med tegnet ^ som angir negasjon. Dersom vi ønsker å matche tegnet ^ gjør vi det ved å ikke bruke det som første tegn.

Naviger til mappen *nyttig*. Test ut og forklar konstruksjonene under:

```
ls fil?.txt
ls -- *.txt
ls -- *.??
ls fil[a13,].txt
ls fil[a-z].txt
```

```
ls fil[^23].txt
ls fil[0-12].txt
echo Hva skal du?
```

Opprett en fil *-l* (liten L) i arbeidsmappen. Forklar forskjellen i resultatet av følgende kommandoer:

```
ls *
ls -- *
```

Resultatet av setningen med *echo* kan virke uforståelig. Når vi benytter jokertegn i argumentene til en kommando blir jokertegnet erstattet med matchende filer *før* kommandoen utføres. Kommandoen ser altså ikke jokertegnene, men resultatet etter at skallet har funnet matchende filer. Dersom ingen filer matcher vil jokertegnkonstruksjonen bli gitt som argument til kommandoen.

## Oppgave 5 – Tegn i filnavn (ikke obligatorisk)

Vi skal nå arbeide med filnavn og tegn i filnavn.

1. Opprett to filer i samme mappe, kall dem *Fil.txt* og *fil.txt*. List mappeinnholdet med *ls*. Hva blir konklusjonen mhp. UNIX og skille mellom store og små bokstaver?
2. Bortsett fra tegnet / kan alle tegn benyttes i filnavn. Mange tegn lager likevel problemer. Hvorfor bør vi unngå filnavn som inneholder tegn som `*` og `??`.
3. Vi kan unngå spesialbetydningen av et tegn ved å skrive tegnet \ foran tegnet. Bruk dette og opprett en fil med navn `*?`. Slett så filen. Hvordan gjør du det?
4. Mellom to hermetegn `"` mister de fleste spesialtegn sin betydning, unntatt tegnene «\», «`.» og «$» . Forsøk følgende kommando i mappen *nyttig*:

   ```
   echo "Hei du?  Hvor er $HOME?"
   ```
5. Erstatt tegnene `"` med `'`. Utfør igjen kommandoen over. Hva skjer nå?
6. **Ikke** lag filnavn med norske bokstaver eller mellomrom. Det vil lage problemer. Norske bokstaver er ikke entydige, men avhenger av tegnsettet som er i bruk. I epost, på web ol. spesifiseres tegnsett i metainformasjon som sendes med, men filsystem lagrer normalt ikke info om hvilket tegnsett som er benyttet. På Linux benyttes i dag ofte tegnsettet UTF-8. Windows er på vei fra ISO-8859-1 (i Vest-Europa) til UCS-2. Tidligere har Windows basert på DOS benyttet CP-865 (i Norden). I alle disse tegnsettene har norske bokstaver forskjellige koder. Et filnavn med en UTF-8 Å vil ikke vises på en maskin som benytter UCS-2.

   Selv om vi unngår å bruke problematiske tegn i filnavn kan andre sende oss slike filer. Vi må derfor kunne håndtere dem. Opprett filene under (bruk gjerne *touch* og kopier med musen):

   (a) `foreløpig.txt`
   (b) `okf il.txt`

   Hvis du ser en boks med et nummer i eller et tegn `?` i noen av filnavnene betyr det at du på din maskin mangler skrifttype med dette tegnet. Tegn 3 i filnavnet i punkt (b). er inkludert blant annet i skrifttypen *gnu-free-mono-fonts*.

   Fra et terminalvindu, åpne et nytt terminalvindu ved følgende kommando:

   ```
   LANG=no_NO.iso8859-1 gnome-terminal&
   ```

   I det nye terminalvinduet, list mappeinnholdet med *ls*. Forsøk så å slette filene ved:

   ```
   rm forel?pig.txt
   rm ok?il.txt
   ```

   Hva ble resultatet? Hvorfor resultatet ble som det ble?

7. Forsøk også å slette filene ved:

   ```
   rm forel??pig.txt
   rm ok???il.txt
   ```

   Forklar det du ser. Spør foreleser hvis oppgaven var uklar eller resultatet var uforståelig.
8. Mellomrom lager problemer da mellomrom normalt benyttes for å skille argumenter til programmer. Et filnavn med mellomrom vil da oppfattes som to ulike argumenter. Ved å være omhyggelig når vi senere skal lage skallprogrammer kan problemet omgås, men ikke alle systemprogrammer vi vil benytte er like påpasselige. Opprett en fil med navn *en liten fil.txt*. Slett så filen. Hvilke kommandoer bruker du?
9. Linjeslutt er ofte representert forskjellige på UNIX og Windows. UNIX benytter vanligvis tegnet `LF`, mens Windows vanligvis benytter `CR+LF`.

   Opprett en tekstfil over flere linjer i Linux. Forsøk så å se innholdet på Windows med programmet *notepad*. Har vi et problem her?

## Oppgave 6 – Omdirigering (ikke obligatorisk)

Utfør kommandoen under og se hva som skjer

```
ls -l > lsut
```

Sjekk at det ikke finnes en fil ikkefil. Utfør så kommandoene under og se hva som skjer

```
ls ikkefil > lsut
```

Sjekk innholdet av filen *lsut*. Hvorfor kom feilmeldingen til skjermen?

Skriv om kommandoen over slik at både `stdout` og `stderr` omdirigeres til filen *lsut*.

Opprett en fil innfil med noe tekst. Hva gjør kommandoen under?

```
cat < innfil > utfil
```

Kjenner du en annen kommando for å oppnå samme resultat?

Ved å omdirigere filen *innfil* til kommando *tr*, erstatt alle mellomrom med tabulator.

## Oppgave 7 – Kanaler (ikke obligatorisk)

Ofte kan utskrift fra kommandoer bli uhåndterlig stor. Test ut resultatet av følgende kommando:

```
help test | less
```

Hva skjedde? Forsøk selv å sende utskriften fra kommando *ls* til programmet *less*.

Utfør kommandoen under. Sjekk manualsidene til de involverte kommandoene for å forstå resultatet.

```
ls -l | tr -s " " | tr " " "\t" | sort -gk5
```

Sjekk manualsiden til kommando *cut*. Skriv så en kommando for å vise kun eier, størrelse og filnavn i listingen fra kommando `ls -l` .

## Oppgave 8 – Oppstartsfiler for bash (ikke obligatorisk)

Legg inn følgende i filen `~/.bash_profile`

```
export navn="<Sett inn ditt navn>"
```

Logg ut og inn igjen. Har variabelen fått verdi? For å referere innholdet av variabelen må vi referere den som «`$navn`».

Legg inn følgende linje i filen `~/.bashrc`

```
alias finnSiste="ls -ltr | tail -n1"
```

Start et nytt teminalvindu. Eksisterer aliaset? Hva gjør det? List ut alle alias.

## Oppgave 9 – Variabler i bash (ikke obligatorisk)

Utfør setningene under:

```
tall="23"
gnome-terminal&
```

I det nye terminalvinduet, sjekk om variabelen finnes. Hva blir resultatet?

Endre koden over slik at variabelen tall vil finnes når et nytt terminalvindu opprettes.

Endre verdien på tall i et av de to vinduene. Hva skjer med verdien i det andre vinduet?

## Oppgave 10 – Bash og miljøvariabler (ikke obligatorisk)

Rediger din ~/.bash_profile og inkluder arbeidsmappen og mappen ${HOME}/bin i søkestien for programmer. Arbeidsmappen angis med ett punktum.

Bruk kommando *ls* og list ut en eksisterende fil. Hva er returverdien? Hva blir returverdien når kommandoen *ls* brukes på en ikke-eksisterende fil?

## Oppgave 11 – Kommandosubstitusjon (ikke obligatorisk)

Gi variabelen med navn filer verdien som tilsvarer navnene på alle filene i arbeidsmappen. Bruk så programmet *less* til å bla i innholdet.

Les om kommandoen *grep*. Bruk *grep* til å liste ut kun filene som inneholder en bokstav «a-c» i filnavnet. Bruk variabelen filer.

## Oppgave 12 – Prosesser (ikke obligatorisk)

Start *gedit* i forgrunnen. Flytt så prosessen til bakgrunnen.

Lag en fil ~/bin/prog med følgende innhold:

```
#!/bin/bash
frukt="Banan"
echo $frukt
```

Gjør programmet kjørbart og utfør det i et kommando-vindu. Hva blir verdien av variabelen frukt i kommando-vinduet?

Utfør programmet på nytt, men nå slik at variabelen frukt også blir tilgjengelig i kommando-vinduet.

## Oppgave 13 – Bash og aritmetikk (ikke obligatorisk)

Gjør følgende tilordning til en ikke-eksisterende variabel:

```
tall=4+76
echo $tall
```

Hva blir konklusjonen? Utfør setningene på nytt, men deklarer først variabelen tall som en heltallsvariabel.

Test ut koden under:

```
resultat="Kvadratet av $(( n = 2 )) er $(( n**2 ))."
echo $resultat
```

Vi ser at tegnet $ brukes for å hente ut verdien fra en `(( ... ))` .

Opprett en ny variabel `res`. Variabeltilordningen er som følger:

```
res="Absoluttverdien av $n er $(( ... ))."
```

Skriv koden for siste `(( ... ))` slik at absoluttverdien til `n` blir returnert.

## Oppgave 14 – Kommando test (ikke obligatorisk)

Skriv kode for [ ... ] nedenfor slik at svarene stemmer:

```
[...] || echo "Filen \"$fil\" er ikke lesbar for deg."
[...] || echo "Det finnes ingen fil \"$fil\"."
[...] && echo "Teksten \"$tekst\" er tom."
[...] && echo "Tekstene \"$tekst1\" og \"$tekst2\" er like."
[...] && echo "Et av tallene \"$tall1\" eller \"$tall2\" er større enn 0."
```

## Oppgave 15 – Skallprogrammering (obligatorisk)

Skriv et skallprogram **repeter.sh** som ved hjelp av en løkke viser en tekst på skjermen et gitt antall ganger. Antall repetisjoner og teksten som skal repeteres skal oppgis som argument til programmet.

Nedenfor vises et eksempel på kjøring av programmet:

```
repeter.sh 4 "Denne teksten skal repeteres"
Denne teksten skal repeteres
Denne teksten skal repeteres
Denne teksten skal repeteres
Denne teksten skal repeteres
```

***Inndata:***

- Ingen

***Argument:***

1. Et heltall som er antall ganger teksten i det andre argumentet skal repeteres
2. En tekst som skal repeteres

***Utdata:***

- Teksten som repeteres

***Nyttige ting:***

- `test` eller `[]`
- `if`
- `declare`
- `for`
- overføring av argument til skallprogram

## Oppgave 16 – Skallprogrammering (obligatorisk)

Lag et skallprogram **filkontroll.sh** som tar navnet på en fil som første argument og et tidsintervall som andre argument. Programmet skal regelmessig undersøke filen som ble gitt som argument. Tidsintervallet angir hvor mange sekunder det skal gå mellom hver gang filen undersøkes.

Programmet skal gi beskjed i terminalen ved følgende hendelser:

- **Filen ble opprettet.** Filnavnet gitt som argument til programmet var navnet til en fil som ikke eksisterte når programmet ble startet. Nå er en fil med dette navnet opprettet. Utdata til skjerm skal være Filen `<filnavn>` ble opprettet. `<filnavn>` er filnavnet som er gitt som argument.
- **Filen ble slettet.** Filnavnet gitt som argument var navnet til en eksisterende fil som nå er blitt slettet. Utdata til skjerm skal være Filen `<filnavn>` ble slettet. `<filnavn>` er filnavnet som er gitt som argument.
- **Filen ble endret.** Filnavnet gitt som argument var navnet til en eksisterende fil hvis innhold nå er blitt endret i følge tidsstempel (`mtime`) på filen. Utdata til skjerm skal være Filen `<filnavn>` ble endret. `<filnavn>` er filnavnet som er gitt som argument.

Etter at en av hendelsene over har skjedd og etter at melding er skrevet til terminal, skal programmet avsluttes.

Et datoformat egnet for å sammenligne tider er et UNIX timestamp.

Testkjøring av dette programmet må skje i to kommandovinduer. Ett som kjører programmet og ett hvor du endrer filen sin status.

***Inndata:***

- Ingen

***Argument:***

1. Et heltall som angir tidsintervall i sekunder
2. Et filnavn

***Utdata:***

- En av meldingene som er vist over

***Nyttige ting:***

- `stat`
- `date`
- `while`
- `if`
- `test`
- `sleep`
- `true`
- overføring av argument til skallprogram

## Oppgave 17 – Skallprogrammering (obligatorisk)

Lag et skallprogram **kontrollflerefiler.sh** som tar en mengde filnavn som argument. For hver fil skal programmet sjekke status en gang i minuttet ved å benytte programmet **filkontroll.sh** fra forrige oppgave.

***Inndata:***

- Ingen

***Argument:***

- En mengde filnavn

***Utdata:***

- Meldinger som i forrige oppgave, en for hver fil som overvåkes

## Oppgave 18 – Skallprogrammering (obligatorisk)

Lag et program **summer.sh** som spør bruker om tall og deretter leser inn tallet som gis. Når bruker er ferdig med å skrive inn tall trykker han tastekombinasjonen `ctrl+d` og programmet skal skrive ut summen av alle tallene brukeren har oppgitt.

***Inndata (skrevet inn av bruker):***

- En sekvens av tall avsluttet med tastekombinasjonen `ctrl+d`

***Utdata:***

- Summen av alle tallene som er gitt

Nedenfor vises et eksempel på kjøring av programmet (<enter> forteller at returtasten trykkes, <ctrl+d> forteller at tastekombinasjonen ctrl og d trykkes):

```
summer.h
1 <enter>
2 <enter>
3 <enter>
4 <enter>
<ctrl+d>
10
```

***Nyttige ting:***

- `while`
- `read`
- aritmetiske oprasjoner

## Oppgave 19 – Skallprogrammering (obligatorisk)

Vi har som driftsavdelingen i bedriften, blitt bedt om å analysere loggfilen til et program. Loggfilen har følgende format for hver linje (<tab> representerer et tabulatortegn)

navnet på hendelse<tab>kjøretiden for hendelse

Nedenfor vises et utdrag fra loggfilen **hendelse.logg**:

```
CallTilEdb   8
CallCustomer   9
CallTilEdb   4
CustomerChk  10
CustomerChk  15
CallTilEdb   16
```

Lag et skallprogram **vistid.sh** som tar loggfilen som argument og leser inn navnet på en hendelse fra bruker og skriver ut den totale kjøretiden til denne hendelsen. Merk at hendelsen kan forekomme flere ganger.

***Ledetekst til brukeren:***

- Hva er hendelsen?

***Inndata:***

- Navn på hendelse

***Argument:***

- Loggfil

***Utdata:***

- Kjøretiden for hendelsen

Nedenfor vises et eksempel på kjøring av programmet (<enter> forteller at returtasten trykkes):

```
vistid.sh hendelse.logg
Hva er hendelsen?  CallTilEdb <enter>
28
```

***Nyttige ting:***

- `grep`
- `cut`
- omdirigering av `stdin`