Obligatorisk oppgave 10: Programmering i Bourne Again Shell

Start oppgave

Forfall Lørdag av 23.59 **Poeng** 0 **Må leveres** en filopplasting **Filtyper** sh

I denne obligatoriske oppgaven skal du lage tre programmer som kan kjøres av Bourne Again Shell (*bash*). For (mye) mer lærestoff om shellprogrammering, se læringsmodulen i Canvas.

Oppgaveteksten består av to deler:

1. I den **første delen** gis det et lite "lynkurs" i shellprogrammering (aka "scripting") i *bash*. Her finner du alt(?) du trenger for å løse oppgavene i del 2. Det er også noen enkle øvinger som det ikke skal leveres noen besvarelse på.
2. Den **andre delen** inneholder selve programmeringsoppgavene som du skal svare på. Det skal leveres besvarelser i form av tre *bash*-script.

1 Lynkurs i shellprogrammering

Et shell i Linux er et C-program som implementerer et tekstbasert grensesnitt mot bruker. Shellet tilbyr en interaktiv linjebasert "kommandotolker" som kommuniserer med brukeren i et terminalvindu.

Det finnes mange ulike shell som kan brukes i Linux. Det vanligste og mest brukte heter Bourne Again Shell, tilgjengelig som et program med navn *bash*.

bash leser en og en linje med tekstlig input fra bruker/tastaturet. Shellet interpreterer (tolker) input, finner ut hva brukeren vil utføre og starter utføringen. Evt. gis det en feilmelding hvis det er feil/uklarheter i input.

Input til shellet er vanligvis Linux-kommandoer som skal utføres. En kommando er oftest en angivelse av hvor på disken OS finner et ferdig kompilert program. Shellet ber OS om å laste dette programmet inn i RAM og starte det som en prosess på maskinen.

1.1 Shellprogrammer

Shellet kan også lese input fra en fil. Hver linje på filen tolkes da som en Linux-kommando, og utføres på samme måte som om kommandoen var gitt fra tastaturet.

Men: Shellet tilbyr også et programmeringsspråk med bla. kontrollstrukturer og variabler, som kan brukes til å styre og kontrollere kjøringen av kommandoer.

En fil med shellkommandoer og programkode kalles for et shellprogram eller shellscript. Shellprogrammering, som også kalles for scripting, brukes bl.a. til effektivisering og automatisering av Linux. Det er også et standard verktøy for systemdrift og -administrasjon.

Det er viktig å merke seg at shellprogrammer ikke kompileres til eksekverbar kode som f.eks. programmer skrevet i C. I stedet interpreteres et script, og kjøres linje for linje av shellet. Interpretering er mye langsommere enn å kjøre ferdig kompilert kode. Scripting brukes derfor ikke for "vanlig" programmering der effektivitet gjerne er viktigst.

1.2 Hello world!

Her er et eksempel på et *bash* shellprogram:

```
#!/bin/bash
echo "Hello, world!"
```

Den første linjen i programmet er en standard for scripts, den forteller Linux hvilket program (*bash*) som skal kjøre/interpretere scriptet. Deretter kommer det en linje som gjør utskrift til skjermen med kommandoen *echo*.

Anta at dette programmet er lagret på en fil med navn *hello.sh* i stående katalog. Programmet kan kjøres på to ulike måter:

1. Gjør først filen eksekverbar med *chmod* og kjør den deretter bare ved å angi filnavnet i stående katalog:

   ```
   chmod u+x hello.sh
   ./hello.sh
   ```

2. Eller gi scriptet som input til programmet *bash*:

   ```
   bash hello.sh
   ```

Begge metodene vil kjøre programmet på samme måte. Last ned dette programmet selv og kjør ditt første(?) shellscript!

1.3 Shellvariabler: Tekst og heltall

Shellscripts bruker egne variabler til å lagre dataverdier. En shellvariabel refereres til med et variabelnavn, akkurat som i andre programmeringsspråk.

Merk at variable i shellet er uten datatype! De lagrer bare bytes, og tolkes som enten tekststrenger eller heltall.

Variable i shellet trenger ikke å deklareres. De opprettes automatisk av shellet første gangen det er en referanse til variabelen. For å få tilgang til dataverdien lagret i en variabel, brukes tegnet '$' (dollartegn)

foran variabelnavnet:

```
$variabelnavn
```

For å tilordne/legge inn en verdi i en variabel brukes likhetstegn (merk at en "space" før eller etter likhetstegnet vil gi feilmelding):

```
variabelnavn=verdi
```

Her kan *verdi* være:

- En konstant
- Verdien til en annen variabel, som hentes ut med å bruke $
- Resultatet fra f.eks. en regneoperasjon
- Output fra en Linux-kommando (aka "kommandosubstitusjon"), se eksempel nedenfor

Variabler kan også få en verdi som leses inn fra tastatur/standard input med kommandoen *read*. Her kan man bruke opsjonen *-p* for å gi en "prompt" til bruker.

I eksempelprogrammet gitt nedenfor vises:

- Enkel bruk av variabler som lagrer tall og tekst
- Innlesing av verdier fra bruker med *read*
- Tallregning, som kan skrives inne i doble paranteser i *bash*
- Kommandosubstitusjon, som kan angis ved å sette tegnet '$' foran en Linux-kommando som skrives inne i paranteser

Se kommentarene i programmet (som angis på linjer som starter med tegnet '#') for ytterligere forklaring:

```
#!/bin/bash

# Leser inn (tall)verdier for to variable X og Y
read -p "X og Y: " X Y

# Beregner summen av innleste tall, lagrer det i en ny variabel Z
# og skriver ut resultatet
((Z=X+Y))
echo "$X + $Y = $Z"

# Setter en variabel lik en tekststreng
farge="grønn"

# Skriver ut variabelen. Merk bruken av {} for å skille variabelnavnet
# fra annen tekst
echo "Livet er alltid skjønt, håpet er lyse${farge}t"

#Kommandosubstitusjon: Legger output fra kommandoer inn i en streng
echo "Hællæ, $(whoami), tidspunktet er nå $(date)"

# Kommandosubstitusjon: Kjører en kommando som teller antall filer i
# stående katalog. Lagrer resultatet av dette i en variabel som heter
# "antall_filer", og skriver det ut.
```

```
antall_filer=$(ls | wc -l)
echo "Antall filer: $antall_filer"
```

Her er en utskrift fra programmet, som er lagret på en eksekverbar fil med navn *eksempel_1.sh*, kjørt på egen maskin:

```
$ ./eksempel_1.sh
X og Y: 2 2
2 + 2 = 4
Livet er alltid skjønt, håpet er lysegrønnt
Hællæ, janh, tidspunktet er nå on. 11. okt. 09:26:39 +0200 2023
Antall filer: 5
```

Last ned shellprogrammet og kjør det selv. Skriv det deretter om slik at det i stedet beregner produktet av *X* og *Y*.

1.4 Parametre til shellprogrammer

Shellprogrammer kan ta parametre som input når de startes. Parameterverdier angis på kommandolinjen i shellet:

```
$./shellprogram param1 param2 param3
```

Verdier fra kommandolinjen er tilgjengelige inne i programmet gjennom et sett med spesialvariable i *bash*:

- *$1*, *$2*, *$3*... er verdiene gitt på kommandolinjen.
- *$0* er navnet på selve shellprogrammet.
- *$#* inneholder antallet parametre.
- *$@* og *$** angir begge mengden av alle parametre til programmet.

Eksemplet nedenfor viser hvordan disse mekanismene virker:

```
#!/bin/bash
echo "Hei, jeg er et script som heter $0"
echo "Antall parametre som jeg har fått er: $#"
echo "Verdiene av disse er: $@"
echo "Her er de tre første parametrene: \$1=$1, \$2=$2, \$3=$3"
```

Anta at dette programmet heter *parametre.sh*. En kjøring kan se slik ut:

```
$ ./parametre.sh Linux er verdens beste OS
Hei, jeg er et script som heter ./parametre.sh
Antall parametre som jeg har fått er: 5
Verdiene av disse er: Linux er verdens beste OS
Her er de tre første parametrene: $1=Linux, $2=er, $3=verdens
```

1.5 Logiske tester: Kommandoen *test*

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. Dette er en tallverdi som lagres i shellet.
- Exit-status lik 0 betyr “alt i orden”.
- En exit-status større enn 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 spesialvariabelen *$?*.

I shellprogrammering representeres boolske verdier som heltall. 0 (null) representerer *true*, alle andre verdier representerer *false*. Denne konvensjonen er basert på at Linux-kommandoer og -verktøy vanligvis returnerer en exit-status lik 0 for å indikere suksess, og en status som ikke er null for å indikere feil.

Kommandoen *test* kan brukes til å beregne verdien av logiske uttrykkk i Linux. *test* er et program som returnerer med exit-status lik 0 hvis det logiske uttrykket er *true*, og 1 hvis uttrykket er *false*.

Syntaksen for å bruke *test* er:

```
test logisk-uttrykk
```

For å få ryddigere kode i shellprogrammer, tillater Linux alternativt denne syntaksen for å kjøre kommandoen *test*:

```
[ logisk-uttrykk ]
```

test bruker ulike operatorer til å sammenligne og teste heltall og tekststrenger. Kommandoen kan også teste egenskaper ved filer, som er nyttig bl.a. for unngå run-time feil i shellprogrammer.

Her er noen operasjoner som *test* kan utføre:

Uttrykk:	*true* hvis:
`[ s1 = s2 ]`	De to strengene *s1* og *s2* er like
`[ s1 != s2 ]`	De to strengene *s1* og *s2* ikke er like
`[ -n s ]`	*s* har lengde større enn 0 (er ikke-tom)
`[ n1 -eq n2 ]`	De to heltallene *n1* og *n2* er like
`[ n1 -ne n2 ]`	De to heltallene *n1* og *n2* 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*
`[ -e fil ]`	Filen med navn *fil* eksisterer

- `[ -f fil ]` Filen er en vanlig (regulær) fil
- `[ -d fil ]` Filen er en katalog
- `[ -r fil ]` Filen kan leses
- `[ -w fil ]` Filen kan skrives
- `[ -x fil ]` Filen kan eksekveres
- `[ ! u ]` Det logiske uttrykket *u* er *false* (*NOT*)
- `[ u1 -a u2 ]` Begge uttrykkene *u1* og *u2* er *true* (*AND*)
- `[ u1 -o u2 ]` Minst ett av *u1* og *u2* er *true* (*OR*)

Paranteser (som må "escapes" med en backslash) kan brukes til gruppering/presedens i logiske uttrykk. Her er et eksempel på et sammensatt logisk uttrykk i *test*:

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

Merk at syntaksen til *test* er svært "streng". Bl.a. må det være en space mellom uttrykk og [], og mellom operander og operatorer. En manglende space kan gi "morsomme" feilmeldinger som ikke alltid er like lette å tolke. Se *man test* for mer informasjon om hvilke operasjoner *test* kan utføre.

1.6 Betinget eksekvering: *if*

Betinget eksekvering (aka "seleksjon") i et program gjør at vi kan utføre en del av koden bare hvis en logisk betingelse er sann. *Bash* tilbyr to kontrollstrukturer for å kunne gjøre betinget eksekvering, *if* og *case*. Vi skal her bare se på *if*-tester. *case*-setningen er beskrevet i læringsmodulen om shellprogrammering i Canvas.

Syntaksen for en *if*-test i shellet er:

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

Dette fungerer slik: Hvis *TEST-KOMMANDO* returnerer med exit-status lik 0, så utføres *FLERE-KOMMANDOER*, ellers utføres de ikke. Merk at en *if*-setning avsluttes med nøkkelordet *fi*.

I den første linjen i en *if*-setning bruker vi ofte kommandoen *test*, skrevet med [], for å evaluere et logisk uttrykk. Her er et eksempel som sjekker verdien på shellvariabelen *USER* (navn på innlogget bruker):

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

Men en hvilken som helst Linux-kommando kan brukes i en *if*-test, som i dette eksemplet:

```
if who | grep -q 'janh'
then
```

```
   echo "Hacker alert!"
fi
```

Forklaring på hvordan dette virker:

- Testen vil først kjøre kommandoen *who*, som gir informasjon om alle brukere som er logget på.
- Output fra *who* sendes til kommandoen *grep*, som finner linjer som inneholder tekststrengen *janh*. Opsjonen *-q* (quiet) til *grep* gjør at det ikke kommer noen utskrift til skjermen.
- *grep* vil returnere med exit-verdi lik 0 (*true*) hvis den finner tekststrengen, og *false* ellers.
- Testen derfor slå til bare hvis brukeren *janh* er logget på.

Her er et komplett shellprogram som viser flergrenet *if*-test med *elif* og *else*:

```
#!/bin/bash

if [ $# -ne 3 ]
then
    echo "usage: $0 num1 num2 num3"
    exit 1
fi

if [ $1 -gt $2 ]
then
   if [ $1 -gt $3 ]
   then
      L=$1
   else
      L=$3
   fi
elif [ $2 -gt $3 ]
then
   L=$2
 else
   L=$3
fi

echo "The largest number is $L"
```

Programmet ovenfor finner det største av tre tall som er gitt som parametre på kommandolinjen. Det skrives ut en "standard" feilmelding hvis det er feil antall parametre.

1.7 Iterasjon

Bash tilbyr flere kontrollstrukturer for å få til iterasjon, dvs. at en del av koden kjøres et antall ganger i en løkke. Vi skal her se på disse tre løkkestrukturene:

- *while*: Går så lenge en løkkebetingelse er *true*
- *until*: Går inntil en løkkebetingelse er *true*
- *for...in*: Itererer over hvert element i en angitt liste

1.7.1 *while*-løkker

Syntaksen for en *while*-løkke ser slik ut:

```
while TEST-KOMMANDO
do
   FLERE-KOMMANDOER
done
```

Her vil den logiske testen som kjøres av *TEST-KOMMANDO* håndteres nøyaktig på samme måte som for *if*-tester.

Det gjøres ofte tallregning i en *while*-løkke som endrer betingelsen. Her er et eksempel:

```
#!/bin/bash

if [ $# -ne 1 ]
then
   echo "usage: $0 number"
   exit 1
fi

i=0
sum=0

while [ $i -lt $1 ]
do
  ((i++))
  ((sum+=i))
done

echo 1+2+3+...+$1 = $sum
```

Dette programmet får et tall *n* som parameter på kommandolinjen, og beregner og skriver ut summen 1+2+3+...+*n*.

Det finnes også to egne kommandoer i *bash* som kan brukes til å styre eksekvering av løkker:

- *break*: Avbryt og hopp helt ut av løkken
- *continue*: Avbryt nåværende iterasjon og gå til toppen av løkken igjen

1.7.2 *until*-løkker

Syntaksen for en *until*-løkke ser slik ut:

```
until TEST-KOMMANDO
do
   FLERE-KOMMANDOER
done
```

Eksempelet nedenfor bruker en *until*-løkke til å "overvåke" systemet. Det gis en "alarm-melding" når en bestemt bruker, gitt som parameter til programmet, logger seg på:

```
#!/bin/bash

if [ $# -ne 1 ]
then
   echo "usage: $0 name"
```

```
   exit 1
fi

until who | grep -q $1
do
  echo "Waiting..."
  sleep 10
done

echo "$(date): Hacker alert! $1 has logged on!"
```

Merk bruken av Linux-kommandoen *sleep* for å få løkken til å ta en "pause" på 10 sekunder mellom hver gang systemet sjekkes.

1.7.3 *for*-løkker

Syntaksen til en *for*-løkke ser slik ut:

```
for VARIABEL in LISTE
do
   FLERE-KOMMANDOER
done
```

LISTE er her en liste med verdier, adskilt med whitespace. Listen kan angis eksplisitt eller med jokernotasjon. Den kan også lages med en Linux-kommando eller hentes fra en variabel. Løkken vil gå én gang per element i listen. I hver iterasjon lagres nåværende verdi i listen i *VARIABEL*.

Her er enkelt eksempel på en *for*-løkke i *bash* som bruker en løkkevariabel *i*:

```
for i in en to tre "fire fem seks"
do
   echo "$i"
done
```

Denne løkken vil gå fire ganger og skrive ut:

```
en
to
tre
fire fem seks
```

For å få en *for*-løkke til å "telle opp" et antall heltallsverdier, kan vi bruke Linux-kommandoen *seq* som lager en sekvens med heltall. I eksemplet nedenfor lages det en sekvens som starter med 1 og går i steg på 2 opp til 20:

```
for number in $(seq 1 2 20)
do
   echo -n "$number "
done
```

Utskriften fra denne løkken blir:

```
1 3 5 7 9 11 13 15 17 19
```

Dette eksemplet bruker output fra Linux-kommandoen *ls* til å lage listen med verdier som løkken skal gå gjennom:

```
n_files=0
for file in $(ls)
do
  ((n_files++))
done
echo $n_files
```

Programmet ovenfor vil telle opp og skrive ut antall filer i stående katalog. Det vil gi samme resultat som kommandoen *ls* | *wc -l*.

Til slutt et eksempel som bruker jokernotasjon ("file name generation") i shellet:

```
for filename in *.txt
do
  if [ -w $filename ]
    then echo $filename
  fi
done
```

Løkken vil skrive ut navnet på alle tekstfiler i stående katalog som vi har skrivetilgang til.

2 Oppgaver

Det er gitt tre (forhåpentligvis) enkle oppgaver som skal besvares med *bash*-kode.

Oppgave 1

Skriv et shellprogram som ber brukeren om å taste inn et fornavn. Programmet skal skrive ut hvor mange registrerte brukere på systemet som har dette fornavnet.

Hvis du kjører programmet (som ligger på filen *oppgave_1.sh*) på maskinen *itstud.hiof.no*, kan utskriften se slik ut:

```
$ ./oppgave_1.sh
Fornavn? Wilhelm
Antall brukere som har navnet Wilhelm: 7
```

Du finner informasjon om alle brukerne på serveren i systemfilen */etc/passwd*.

Oppgave 2

Skriv et shellprogram som har to parametere. Begge parametrene må være eksisterende kataloger der vi har skrivetilgang, ellers skal programmet avbrytes med feilmelding.

Programmet skal flytte alle filer i den første katalogen, som har et filnavn som slutter på *.mp3*, over i den andre katalogen. Det skal også skrive ut hvor mange filer som er flyttet.

Utskriften fra en kjøring av programmet (som ligger på filen *oppgave_2.sh*) kan se slik ut:

```
$ ./oppgave_2.sh tempdir musikkarkiv
Flyttet 2 mp3-filer fra tempdir til musikkarkiv
```

Oppgave 3

Skriv et shellprogram som har et filnavn som parameter på kommandolinjen. Programmet skal terminere med feilmelding hvis den angitte filen ikke finnes.

Deretter skal programmet gå i en "evig løkke" som "sover" i ett minutt mellom hvert gjennomløp. I hver iterasjon skal det sjekkes om filen gitt som parameter har endret *størrelse*. Hvis filstørrelsen forandrer seg, skal programmet avslutte med en melding om at filen er endret.

Størrelsen til en fil kan finnes på flere måter i Linux. I denne opgaven kan du bruke filverktøyet *stat*. Følgende Linux-kommando vil finne størrelsen på filen *filename* i antall bytes og lagre den i en variabel med navn *size*:

```
size=$(stat -c "%s" filename)
```

Hvis du bruker MacOS, som er basert på BSD Unix og ikke på Linux, vil kommandoen ovenfor gi en feilmelding. Dette skjer fordi *stat* er implementert litt anderledes og med andre opsjoner i MacOs enn i standard Linux. Følgende kommando i terminalen i MacOS lagrer størrelsen på filen *filename* i variabelen *size*:

```
size=$(stat -f "%z" filename)
```

Hva skal du levere?

Svarene på oppgave 1, 2 og 3 skal legges inn i tre filer med navn *oppgave_1.sh*, *oppgave_2.sh* og *oppgave_3.sh*. Filene skal lastes opp i Canvas.