Shellprogrammer og -variabler

Innhold

- Hva er et shellprogram?
- Kjøring av shellprogrammer
- Feil, feilsøking og feilmeldinger
- Input og output
- Shellvariable
- Kommandosubstitusjon
- Tekststrenger
- Tallregning
- Parametre til shellprogrammer

Shellprogrammer

- Shellet kan lese kommandoer fra en fil i stedet for tastatur, linje for linje inntil filen er slutt
- Hver linje på filen *interpreteres* som en Linux-kommando og eksekveres på vanlig måte
- Shellet tilbyr også kontrollstrukturer og variable, som i et vanlig programmeringsspråk
- En fil med shellkommandoer og shellkode kalles for et shellprogram (eller shellscript)
- Shellprogrammering = "scripting"

Hello, world!

```
#!/bin/bash
echo "Hello, world!"
exit 0
```

Kjøring av shellprogrammer

- Enten:

```
chmod u+x filnavn
./filnavn
```

- Eller:

```
bash filnavn
```

Feil og feilsøking i shellprogrammer

- Shellprogrammer kompileres ikke, vi ser først feilene i *runtime*, når scriptet kjøres

- Feilsøking i shellprogrammer:
  - Finnes ingen ordentlig "debugger" *
  - Vanlig å teste shellscripts en og en linje om gangen, etterhvert som programmet skrives

- Viktig å teste scripts for feil før "seriøs bruk":
  - Lett å overskrive filer med `>`
  - «Sterke» kommandoer som `rm` og `find` kan ødelegge mye data

* Bash-opsjonene `-x` og `-v` kan brukes til "debugging"

Feil og feilmeldinger

- Feil i script vil kunne generere feilmeldinger fra shellet eller fra kommandoene som kjøres
- Feilmeldingene er ofte kryptiske og kortfattede
- Ikke alltid lett å se ut i fra feilmeldingene hva som *egentlig* er feil i shellprogrammet
- Feilmeldinger er ofte resultat av små syntaksfeil, som f.eks. en manglende quote eller space
- Shellprogrammerere lærer seg fort å bli relativt nøye med syntaksen...

Eksempel:
Feilmelding fra shellprogram

- Shellprogrammet ligger på filen `feil` :

```
#!/bin/bash
navn=Jonas Fjeld
echo $navn
```

- Kjøring:

```
janh@HP:~/osml$ ./feil
./feil: line 2: Fjeld: command not found
```

Innlesing og utskrift av data

`read` – Interaktiv innlesing av tekstlige data

Opsjoner:

`-p` *`string`*	Bruk *`string`* som ledetekst/prompt
`-n` *`number`*	Ikke les flere enn *`number`* antall tegn
`-t` *`secs`*	Time-out etter *`secs`* sekunder

`echo` – Utskrift av linjer med tekst

Opsjoner:

`-e`	Tillat escape-koder i teksten (`\t`: TAB, `\n`: NEWLINE)
`-n`	Ikke gjør linjeskift etter utskrift av teksten

Skallvariable / Shell variables

- Brukes til å lagre verdier i shellprogrammer
- Refereres til med *variabelnavn*:
  - Variabelnavn kan inneholde `a-z` `A-Z` `0-9` og `_`
  - Må begynne med en bokstav eller `_`
- Variable i shell er default *uten* datatype:
  - Lagrer bare bytes (tegn)
  - *Tolkes* som tekststrenger, heltall eller arrays

Opprettelse av variabler

- Variable i shell *trenger* ikke deklareres
- Opprettes ved første referanse til variabelen:

```
VAR=verdi
```

- For å bruke variabelen/se på verdien:

```
$VAR
```

eller

```
${VAR}
```

- `unset` vil *fjerne* innholdet i en variabel

Eksempel:
Tilordning og bruk av variabel

```
FARGE=Rød
echo $FARGE
echo $FARGEaktig  (*)
echo ${FARGE}aktig
FARGE_2=Grønn; FARGE_3=$FARGE
echo FARGE $FARGE $FARGE_2 $FARGE_3
```

(*): Gir ikke feilmelding, men oppretter en ny variabel med verdien NULL

Variable *kan* deklareres med `declare`

```
declare [options] name[=value]
```

- Opsjoner: *

`-i`	heltallsvariabel (regning "håndteres riktig")
`-r`	read-only(!) variabel
`-l/u`	alltid lower-/upper-case
`-x`	*eksporteres* til subshell
`-a`	array som kan indekseres
`-f`	deklarerer en *funksjon*

- Eksempel:

```
declare -rx Pi=3.14
```

*: + i stedet for - slår *av* opsjonen for denne variabelen

Tilordning av verdier til variabler

- Syntaks*:

```
navn=verdi
```

- `verdi` kan være:
  - Konstant tekststreng, kan innesluttes i `""` eller `''`
  - Verdien til en annen variabel, hentes ut med `$`
  - Verdi som returneres fra en Linux-kommando
  - Resultatet av en regne- eller streng-operasjon

*: Space før eller etter likhetstegnet vil gi feilmelding

Tekststrenger: Forskjellen på ' ', "" og ``

- Single quotes ' ' :

  Shellet ekspanderer ingen spesialtegn, hele strengen brukes akkurat slik den er skrevet

- Double quotes "" :

  Alle tegn beholdes uendret, unntatt $, ` og \

  $ brukes til å hente ut verdier av variable

  \ kan brukes som "escape-tegn" foran $, ` eller "

- Back quotes `` :

  Brukes til å lage en streng med *kommandosubstitusjon*

  Inneholder *output* fra Linux-kommandoen angitt mellom ``

Kommandosubstitusjon

- *Output* fra en Linux-kommando tilordnes som *verdi* til en variabel
- Kan også bruke output fra en kommando som input/ argument(er) til en annen kommando
- Syntaks: Kommandoen innesluttes i “back quotes” *:

```
var=`kommando`
```

- Bash tilbyr også en nyere syntaks-variant:

```
var=$(kommando)
```

*: aka. “back ticks”, oftest oppe til høyre på norske tastatur

Eksempler på kommandosubstitusjon

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

echo "Hællæ, $(whoami), tidspunktet er nå\
$(date)"

rm `cat filer_som_skal_fjernes.txt`

echo -e "Filer med navn som inneholder 'd'\
eller 'o':\n`ls -d *[od]*`"

tekstfil_liste=$(ls *.txt)
echo $tekstfil_liste
echo "$tekstfil_liste"
```

Enkle operasjoner på tegnstrenger *

- Kommandoen `expr` kan bl.a. håndtere tegnstrenger:

```
var=`expr kommando streng [parameter(e)]`
```

- Lengde av tekststreng:

```
lengde=`expr length streng`
```

- Substreng:

```
substreng=`expr substr streng start lengde`
```

- Indeks til første tegn i `str1` som også er i `str2`:

```
indeks=`expr index str1 str2`
```

*: Vi skal lære mer avansert strenghåndtering i Bash senere i kurset

Eksempel: Enkel strenghåndtering

```
#!/bin/bash
S1="Levon Helm"
echo "S1: \"$S1\""

L1=`expr length "$S1"`
echo "L1: $L1"

s1=`expr substr "$S1" 3 5`
echo "S1( 3,5): $s1"

S2="Robbie Robertson"
i=`expr index "$S2" "$S1"`
t=`expr substr "$S2" $i 1`
echo "Første felles tegn i \"$S2\" og \"$S1\": '$t'"
```

Tallregning i shellprogrammer

- Bash kan utføre enkle regneoperasjoner
- Støtter bare regning med heltall *
- Kommandoen `expr` *kan* også brukes til regning:
  - "Utdatert"
  - "Pirkete" og lite fleksibel syntaks
- Vi skal i stedet lære å bruke mekanismene for tallregning som er *innebygget* i Bash

*: Programmet `bc` kan brukes til å regne med floating point fra kommandolinjen

Beregning av aritmetiske uttrykk i Bash

- Regnestykker skrives inne i *doble paranteser:*

  ```
  ((A = A + 1))
  ```

- Raskere og mer fleksibel syntaks enn `expr`

- Alternativt kan kommandoen `let` brukes:

  `let` *`regneuttrykk`*

  gjør akkurat det samme som:

  `((`*`regneuttrykk`*`))`

Operatorene i Bash-aritmetikk

+	Addisjon
-	Subtraksjon
++	Postivt inkrement
- -	Negativt inkremet
*	Multiplikasjon
/	Divisjon
%	Rest ved heltallsdivisjon (mod)
**	Eksponential

Eksempel: Tallregning

```
#!/bin/bash
read -p "X og Y: " X Y
((Z=X+Y)); echo -e "X + Y\t= $Z"
((Z=X-Y)); echo -e "X - Y\t= $Z"
echo -e "X++\t= $((X++))"
echo -e "++X\t= $((++X))"
echo -e "X--\t= $((X--))"
echo -e "--X\t= $((--X))"
echo -e "X * Y\t= $((X*Y))"
echo -e "X / Y\t= $((X/Y))"
echo -e "X%Y\t= $((Z=X%Y))"
echo -e "X^Y\t= $((Z=X**Y))"
```

Parametre til shellprogrammer

- Shellprogrammer kan ta parametre som input
- Parameterverdier angis på kommandolinjen:

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

- Verdiene er tilgjengelige som `$1`, `$2`, `$3`
- `$0` er navnet på selve shellprogrammet
- `$#` er antallet parametre angitt på kommandolinjen
- `$@` og `$*` angir begge *alle* parametre til programmet

Eksempel: Parametre

```
#!/bin/bash
echo "Hei, jeg er et script som heter\
\"`basename $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"
```