6 Komentojonot (shellskriptit)

Komentojono on (ASCII)tiedosto, joka sisältää shellin ymmärtämiä komentoja. Komentojono suoritetaan kirjoittamalla tiedostonnimi. Tiedoston pitää alkaa rivillä (esim.)

```
#!/bin/sh
#!/bin/tcsh
#!/bin/ksh
#!/bin/bash
```

Kun tiedoston alussa on #! lukee kernel sen perästä komennon ja mahdolliset parametrit, joilla kyseinen komentojono suoritetaan.

Komentojono koostuu UNIX-komennoista, shellin komennoista ja kommenteista.

```
#!/bin/sh
# Yksinkertainen komentojono (hello.sh)
echo "Hello World"
```

Ennenkuin komentojono voidaan suorittaa on tiedostolle annettava suoritusoikeus `chmod` komennolla.

```
chmod a+x hello.sh
```

```
./hello.sh
Hello World
```

6.1 Aritmeettiset operaatiot

Bourne shellissä ei ole sisäänrakennettua aritmetiikkaa. Yksinkertaisiin kokonaislukulaskutoimituksiin voidaan kayttää Unixin `expr` komentoa. Liukulukuaritmetiikkaan voidaan käyttää esimerkiksi `awk` tai `bc` ohjelmia. Sisäänrakennetun aritmetiikan puuttuminen hidastaa shellin suorituskykyä esimerkiksi kasvatettaessa laskuria silmukassa, koska jokaisella iteraatiolla on käynnistettävä uusi prosessi suorittamaan aritmeettinen operaatio.

`Expr` komento ymmärtää seuraavat aritmeettiset operaattorit: + (yhteenlasku), − (vähennyslasku), ∗ (kertolasku), / (jakolasku) ja % (modulo). Jokainen `expr` komennon argumentti on erotettava välilyönnillä. Lisää tietoa `expr` komennon käytöstä löytyy komennolla `man expr`.

```
expr 2 \* 5
10
```

Korn shellissä on sisäinen komento `let`, jolla voidaan suorittaa yksinkertaisia aritmeettisia operaatiota. Esimerkiksi

```
let i=2\*5
echo $i
10
```

C shellissä käytetään symbolia @ laitettaessa aritmeettisen operaation tulos numeeriseen muuttujaan:

```
@ i = 2 * 5
echo $i
10
```

Kaikki edellä esitellyt tavat toimivat vain kokonaislukuaritmetiikassa. Kun halutaan suorittaa liukulukuoperaatiota on turvauduttava johonkin ulkoiseen ohjelmaan, esimerkiksi `bc`:hen.

```
x=`echo "scale=5; 234/66;"|bc`
echo $x
3.54545
```

6.2 Funktiot

Komentojonoissa funktioita voidaan käyttää rakenteen selkeyttämiseen (vrt. aliohjelmat). Funktiot on määriteltävä ennen käyttöä. Funktio suoritetaan, kuten sisäinen komentokin, nykyisessä ympäristössä, joten se jakaa komentojonossa määritellyt muuttujat. Mikäli kuitenkin funktion syöte tai tulostus uudelleensuunnataan tai käytetään komentosuoritusta (eli funktio laitetaan ‘ ‘ merkkien sisään) suoritetaan funktio alishellissä. Funktioiden määrittely säilyy vain nykyisessä ympäristössä (shellissä), eikä niitä voi "exportata". `Return` komennolla voidaan palauttaa viimeisen funktiossa suoritetun komennon arvo tai argumenttina annettu arvo (alle 255). Funktion palautusarvo voidaan lukea muuttujasta $?. Funktiot voidaan laittaa myös erilliseen tiedostoon ja ne voidaan ottaa käyttöön komentojonossa käyttäen `.` komentoa.

Funktiolle voidaan viedä argumenttejä käyttäen paikkasidonnaisia parametrejä, jotka ovat funktiokohtaisia, eli ne eivät vaikuta paikkasidonnaisiin parametreihin funktion ulkopuolella.

Funktio määritellään seuraavasti:

funktion_nimi () { komento1; ... komenton; **}**

Korn shellissä funktion määrittely menee seuraavasti:

function funktion_nimi { komento1; ... komenton; **}**

Esimerkki 1.

```
#!/bin/sh
listaus () {
  echo "$1 hakemiston listaus"
  ls $1
return 3
}

listaus /var
echo "Funktion palautusarvo on $?"
```

Esimerkki 2.

```
#!/bin/sh
listaus () {
  echo "$1 hakemiston listaus"
  ls $1
return 3
}

palaute=`listaus /var`
echo "Funktion palautusarvo on $?"
echo "Palaute 'nimessa' on:  $palaute"
```

6.3 Käyttäjän syötteen lukeminen (`read` komento)

Komentojonoissa (sh ja ksh) käyttäjän syöte luetaan shellin sisäisellä `read` komennolla. `Read` komento lukee syötettä kunnes vastaan tulee rivinvaihto. Komentojonoissa `read` komentoa voidaan käyttää myös suorituksen pysäyttämiseen kunnes rivinvaihto luetaan.

`read name`	Luetaan *stdin*:stä rivi ja laitetaan se muuttujan *name* arvoksi
`read name1 name2`	Luetaan *stdin*:stä muuttujaan *name1* kunnes tulee ensimmäinen välilyönti tai rivinvaihto ja loput luetaan muuttujaan *name2*.

```
#!/bin/sh
```

```
echo -n "Kijoita nimesi:"
read name
echo "Luettiin: $name"
```

C shellissä käyttäjän syöte voidaan lukea käyttäen muuttujaa `$<`
Esimerkiksi:

```
#!/bin/csh -f
echo -n "Kirjoita nimesi:"
set name = $<
echo "Luettiin: $name"
```

6.4 `Test` komento, testaus- ja loogiset operaattorit

6.4.1 `Test` komento

`Test` komentoa käytetään evaluoitaessa ehdollisia rakenteita, jotka palauttavat joko tosi tai epätosi. `Test` palauttaa nollan mikäli ehto on tosi ja nollasta poikkeavan mikäli ehto on epätosi.

Seuraavassa esitellään mitä keinoja merkkijonojen, tiedostojen yms. testaukseen eri shellit tarjoavat.

6.4.2 Bourne shell

Merkkijonojen testaus

jono1 = jojo2	jono1 on sama kuin jono2
jono1 != jojo2	jono1 on erisuuri kuin jono2
jono	jono on epätyhjä
-z jono	jonon pituus on nolla
-n jono	jonon pituus on erisuuri kuin nolla

Lukujen testaus

luku1 -eg luku2	luvut yhtäsuuret
luku1 -ne luku2	luvut erisuuret
luku1 -gt luku2	luku1 on suurempi kuin luku2
luku1 -ge luku2	luku1 on suurempi tai yhtäsuuri kuin luku2
luku1 -lt luku2	luku1 on pienempi kuin luku2
luku1 -le luku2	luku1 on pienempi tai yhtäsuuri kuin luku2

Tiedostojen testaus

-b tiedosto	Lohkolaite
-c tiedosto	Merkkilaite
-d tiedosto	Hakemisto
-f tiedosto	Tavallinen tiedosto ei hakemisto
-g tiedosto	set-group-id asetettu
-k tiedosto	sticky bitti asetettu
-p tiedosto	Nimetty putki
-r tiedosto	Tiedostoon on lukuoikeus
-s tiedosto	Tiedoston koko ei ole nolla
-u tiedosto	set-user-id asetettu
-w tiedosto	Tiedostoon on kirjoitusoikeus
-x tiedosto	Tiedostoon on suoritusoikeus

Loogiset operaattorit

!	not operaattori
-a	ja operaattori
-o	tai operaattori

Huomautus: -a operaattori sitoo vahvemmin kuin -o operaattori. Sitomisjärjestystä voidaan muuttaa suluilla.
Esim.

```
if [ \("$x" -gt "$y" -o "$z" -eq 10 \) -a "$f" -lt 100 ]
```

Sulut on suojattava \ merkillä koska ( ) aiheuttaa listan suorituksen alishellissä.

6.4.3 Korn shell

Ksh:ssa voidaan käyttää samoja testaus- ja loogisia operaattoreita kuin Bourne shellissä. Lisäksi ksh:ssa (ainakin "uusissa" versioissa) on käytössä [[...]] "testauskomennon" kanssa seuraavat operaattorit.

Merkkijonojen testaus

jono = hahmo	Jono vastaa hahmoa.
jono != hahmo	Jono ei vastaa hahmoa.
jono1 < jono2	Jonon *jono1* ASCII arvo on pienempi kuin jonon *jono2*.
jono1 > jono2	Jonon *jono1* ASCII arvo on suurempi kuin jonon *jono2*.
-z jono	Jonon *jono* pituus on nolla.
-n jono	Jonon *jono* pituus on eri kuin nolla.

Binaariset tiedostotestit ja loogiset operaattorit

tiedosto1 -nt tiedosto2	Tosi, jos tiedosto *tiedosto1* on uudempi kuin tiedosto *tiedosto2*.
tiedosto1 -ot tiedosto2	Tosi, jos tiedosto *tiedosto1* on vanhempi kuin tiedosto *tiedosto2*.
tiedosto1 -ef tiedosto2	Tosi, jos tiedosto *tiedosto1* on toinen nimi tiedostolle *tiedosto2*.
&&	Looginen ja operaattori (AND).
||	Looginen tai operaattori. (OR)

Lisäksi tiedostojen testaamiseen voidaan käyttää seuraavia operaattoreita.

-a tiedosto	Tiedosto *tiedosto* on olemassa.
-L tiedosto	Tiedosto *tiedosto* on olemassa ja on symbolinen linkki.
-O tiedosto	Omistat tiedoston *tiedosto*.
-G tiedosto	Tiedoston *tiedosto* ryhmänumero (GID) sama kuin sinulla.
-S tiedosto	Tiedosto *tiedosto* on olemassa ja on soketti.

6.4.4 C shell

Vertailu- ja loogiset operaattorit.

==	Yhtäsuuruus
!=	Erisuuruus
>	Suurempi kuin
>=	Suurempi tai yhtäsuuri kuin
<	Pienempi kuin
<=	Pienempi tai yhtäsuuri kuin
=~	Jono vastaa hahmoa
!~	Jono ei vastaa hahmoa
!	Looginen ei (NOT)
||	Looginen tai (OR)
&&	Looginen ja (AND)

Tiedostojen testaus

-r	Nykyisellä käyttäjällä lukuoikeus tiedostoon.
-w	Nykyisellä käyttäjällä kirjoitusoikeus tiedostoon.
-x	Nykyisellä käyttäjällä suoritusoikeus tiedostoon.
-e	Tiedosto on olemassa.
-o	Nykyinen käyttäjä omistaa tiedoston.
-z	Tiedosto nollan mittainen.
-d	Kyseessä on hakemisto.
-f	Kyseessä on tavallinen tiedosto.

Lisäksi `test` komentoa käytettäessä voidaan käyttää samoja testausoperaattoreita tiedostoille kuin Bourne shellissä.

6.5 Ehtorakenteet

6.5.1 `If`, `if/else` ja `if/elif/else` komennot

`If` komento on yksinkertaisin ehdolinen rakenne ja sen muoto on jokin seuraavista:

if komento
then
  komento1
  ...
  komenton
fi

if test expression
then
  komento1
  ...
  komenton
fi

if [expression]
then
  komento1
  ...
  komenton
fi

`If`:n perässä oleva komento suoritetaan ja mikäli komennon palautusarvo on nolla suoritetaan sanojen `then` ... `fi` välissä olevat komennot.

```
#!/bin/sh
echo -n "Anna kokonaisluku:"
read n
if test $n -lt 0
then
  echo "Antamasi luku on pienempi kuin nolla"
fi
```

`If/else` komento mahdollistaa halutut toimenpiteet siinä tapauksessa, että `if` komento ei toteudu. `If/else` komennon muoto on:

if komento
then

  komentoja

else

  komentoja

fi

```
#!/bin/sh
name=koe
if grep "$name" /etc/passwd > /dev/null 2>&1
then
   echo "Kayttaja $name loytyy"
else
   echo "Kayttajaa $name ei loydy"
fi
```

`If/elif/else` komento mahdollistaa moniasteisen testauksen. Mikäli ensimmäinen `if` ei toteudu tutkitaan seuraava `elif` rakenne jne. Jos jokin `if/elif` ehto touteutuu suoritetaan vastaavan `then`:n jälkeiset komennot ja jatketaan komentojonon suoritusta `fi`:n jälkeisestä osasta. `If/elif/else` komento on muotoa:

if komento
then

  komentoja
elif komento
then

  komentoja
else

  komentoja
fi

C-shellissä `if` komento on muotoa
if (expression) **then**

  komentoja
else if (expression2)**then**

  komentoja
else

  komentoja
endif

C shellissä `if`:n perässä tuleva komento on boolean tyyppinen looginen ilmaisu kuten C kielessä. Bourne ja Korn shellissä `if` jälkeinen osa on komento tai joukko komentoja. C-shellissä testattava lauseke sijoitetaan tavallisten sulkujen sisään.

Esimerkki 1.

```
#!/bin/csh -f
echo -n "Anna kokonaisluku:"
set n = $<
if ( $n < 0 ) then
  echo "Antamasi luku on pienempi kuin nolla"
endif
```

Esimerkki 2.

```
#!/bin/csh -f
set name=mjev
grep "$name" /etc/passwd >& /dev/null
```

```
if ($status == 0) then
   echo "kayttaja $name loytyy"
else
   echo "kayttajaa $name ei loydy"
endif
```

6.5.2 `Case` komento

`Case` mahdollistaa `if/else` tyyppisen muuttujan arvon tutkimisen. Muuttujan arvoa verrataan arvoihin arvo1, arvo2 jne. kunnes vastaavuus löydetään. Tämän jälkeen suoritetaan komennot, jotka seuraavat vastannutta arvoa. Komentoja suoritetaan kunnes vastaan tulee kaksinkertainen puolipiste (**;;**). Jos vastaavuutta ei löydy suoritetaan `*)` jälkeiset komennot (tämä vastaa siis `if/else` rakenteessa `else` osaa). `Case` komennon arvojen testauksessa voidaan käyttää jokerimerkkejä ja | symbolia (putken merkki), joka vastaa TAI sanaa.

`Case` komento on muotoa:
case muuttuja **in**
arvo1**)**

   komentoja **;;**
arvo2**)**

   komentoja **;;**
∗)

   komentoja **;;**
esac

```
#! /bin/sh

echo "Valitse editori:"
cat <<EOF
    1) emacs
    2) pico
    3) vi
EOF
read valinta
case "$valinta" in
1) /usr/bin/emacs -nw
   ;;
2) /usr/bin/pico
   ;;
```

```
3) /bin/vi
   ;;
esac
```

C-shellissä `case` rakennetta vastaa `switch` komento, jonka muoto on.

switch (muuttuja)
case arvo1:

komentoja

breaksw
case arvo2:

komentoja

breaksw
endsw

`switch` komennon perassä olevaa ilmaisua testataan `case` sanan perässä olevaan arvoon (label), joka voi sisältää myös jokerimerkkejä. `Breaksw` siirtää suorituksen `endsw` kohtaan. Mikäli `breaksw` jätetään laittamatta suoritetaan kaikki ne "`case` arvo" rivin jälkeiset komennot joihin ilmaisu on täsmännyt kunnes vastaan tulee `breaksw` tai `endsw`.

```
#! /bin/csh -f
echo "Valitse editori:"
cat <<EOF
    1) emacs
    2) pico
    3) vi
EOF
set valinta=$<
switch ("$valinta")
case 1:
   /usr/bin/emacs -nw
   breaksw
case 2:
   /usr/bin/pico
   breaksw
case 3:
   /bin/vi
   breaksw
endsw
```

6.6 Silmukkarakenteet (`for`, `while`, `until`)

Silmukkarakenteita käytetään suorittamaan ryhmä komentoja kunnes tietyt ehdot täyttyvät. Bourne shellissä on käytössä kolme silmukkarakennetta.

6.6.1 For

`For` komento lukee käyttäjän määrittämästä sanalistasta yhden sanan kerrallaan muuttujaan ja suorittaa parin `do done` (silmukan runko) välissä olevat komennot.

`For` komento on muotoa:
for muuttuja **in** sana lista
do

  komentoja
done

```
#!/bin/sh
IFS=:
for hak in $PATH
do
  if [ -d "$hak" ]
  then
    koko=‘du -s -k $hak 2> /dev/null | awk ’{print $1}’‘
    echo  "Hakemiston $hak koko on $koko kt"
  fi
done
```

C-shellissä `for` komentoa vastaa `foreach` komento, jonka muoto on

foreach muuttuja (sana_lista)

  komentoja
end

```
#!/bin/csh -f
foreach hak ( $path )
  if ( -d "$hak" ) then
    set koko=‘du -s -k $hak | awk ’{print $1}’‘
    echo  "Hakemiston $hak koko on $koko kt"
  endif
end
```

6.6.2 While

`While` silmukassa suoritetaan ensin `while`:n perässä oleva komento ja jos sen palautusarvo on nolla suoritetaan silmukan runko (`do done` välissä olevat komennot). Kun `done` osa saavutetaan palataan silmukan alkuun ja komennon palautusarvo tutkitaan uudestaan. Silmukan runkoa suoritetaan niin kauan kunnes silmukan alussa oleva komento palauttaa nollasta poikkeavan arvon. Silmukasta poistuttaessa jatketaan komentojonon suoritusta `done` osaa seuraavalla komennolla.

`While` komento on muotoa:
while komento
do

  komentoja
done

```
#!/bin/sh
num=0
while [ $num -lt 10 ]
do
  echo $num
  num=`expr $num + 1`
done
```

Csh:ssa `while` komento on muotoa:
while (expression)

  komentoja
end

```
#!/bin/csh -f
set num=0
while ( $num < 10 )
  echo $num
  @ num++
end
```

6.6.3 Until

`Until` on samankaltainen silmukkarakenne kuin `while`kin, mutta `until` silmikkaa suoritetaan kunnes alussa oleva komento palauttaa nollan.

`Until` komento on muotoa:
until komento
do

komentoja
done

```
#!/bin/sh
num=0
until [ $num -gt 10 ]
do
  echo $num
  num=`expr $num + 1`
done
```

6.6.4 Goto (csh)

Csh:ssa on käytössä myös `goto` komento, jolla voidaan hypätä komentojonon suorituksessa haluttuun paikaan. `Goto` komento on muotoa.

goto label

komentoja
label:

Tai

label:
komentoja

goto label

Jälkimmäisessä tapauksessa on syytä olla tarkkana ettei tule ikuista silmukkaa.

```
#!/bin/csh -f
uudestaan:
echo -n "Kuinka vanha olet?"
set ika=$<
if ( "$ika" < 0 || "$ika" > 130 ) then
  echo "En usko"
  goto uudestaan
endif
echo "Kerroit olevasi $ika vuotta vanha"
```

6.6.5 Break ja continue komennot

Shellin sisäistä `break` komentoa käytetään pakotettuun silmukasta postumiseen. Kun silmukassa tulee vastaan `break` komento hypätään komentojonon suorituksessa `done` (sh ja ksh) tai `end` (csh) sanan jälkeiseen osaan. Mikäli on useampia sisäisiä silmukoita `break` komennolle voidaan antaa argumenttina minkä silmukan alkuun hypätään. Tässä tapauksessa silmukoiden numerointi menee siten että sisimmän silmukan numero on yksi. Csh:ssa voidaan sama tehdä käyttäen komentoa `repeat 3 break.` (Tämä ei toimi `continue` komennolle.)

Sisäistä `continue` komentoa käytetään kun halutaan hypätä samassa silmukassa uudelle kierrokselle. Myös `continue` hyväksyy argumentin, jolla voidaan hypätä halutun silmukan alkuun. Sisäkkäisten silmukoiden numerointi menee samaan tapaan kuin `break` komennossakin.

6.7 Komentorivioptioiden parsinta (`getopts`) (sh ja ksh)

Komentojonolle voidaan välittää optiota paikkasidonnaisten parametrien avulla, mutta se ei ole aina tehokkain keino. Sisäinen komento `getopts` mahdollistaa komentorivioptioiden käsittelyn.

Esimerkki.

```
#!/bin/sh
exec 2>/dev/null
while getopts abc: optiot
do
  case $optiot in
    a)  echo "loytyi optio $optiot"
        echo " \$OPTARG = $OPTARG ja \$OPTINT = $OPTIND"
        echo "" ;;
    b)  echo "loytyi optio $optiot"
        echo "\$OPTARG = $OPTARG ja \$OPTINT = $OPTIND"
        echo "" ;;
    c)  echo "loytyi optio $optiot"
        echo "\$OPTARG = $OPTARG ja \$OPTINT = $OPTIND"
        echo ""  ;;
    \?) echo "Ei kelpaa! \$optiot= $optiot"
        echo "\$OPTARG = $OPTARG ja \$OPTINT = $OPTIND"
        echo ""  ;;
  esac
done
```

Edellä olevaa komentojonoa voidaan kutsua esim seuraavalla tavalla (komentojonon nimi opts.sh):

`opts.sh -abc 1000` tai `opts.sh -a -b -c 1000`

Edellä olevassa esimerkissä rivi `while getopts abc: optiot` lukee komentorivioptiot yksi kerrallaan muuttujaan *optiot*. `Case` rakenteessa tutkitaan mikä optio on kyseessä. Mikäli luettu optio ei ole mikään hyväksytyistä optiosta tulee muuttujan *optiot* arvoksi ? Kaksoispiste *c*:n perässä tarkoittaa että optioon *c* liittyy argumentti. Kyseinen argumentti luetaan muuttujaan `OPTARG`. Muuttujaa `OPTIND` lisätään aina yhdellä kun `getopts` on saanut luettua komentorivi option, muuttujan `OPTIND` arvo on aluksi yksi.

6.8 Signaalien käsittely (sh ja ksh) (trap)

`Trap` komennolla voidaan käsitellä ohjelman saamat signaalit halutulla tavalla. *Signaali* on asynkroninen viesti (numero) jonka prosessi (tai käyttöjärjestelmä) voi lähettää toiselle prosessille jos jotain näppäintä painetaan tai jotain poikkeuksellista tapahtuu.

`Trap` komennon muoto on:

trap 'komento1; … komento;' signaali_numero(t)

```
#!/bin/sh
trap 'echo "Not allowed"' 1 2 15
while :
do
  echo -n "Go ahead ...>"
  read reply
  if test "$reply" = "stop"
  then
    break
  fi
done
```

Csh:ssa voidaan keskeytys (ctrl-c) hoitaa komennolla `onintr`.

```
#!/bin/csh -f
onintr finish
while (1)
  echo -n "Kirjoita jotain ja paina enter:"
  set tmp = $<
  echo "Kirjoitit $tmp"
end
finish:
echo "Ulkona silmukasta"
```

Komento `onintr label` toimii siten, että keskeytyksen tullessa komentojonon suorituksessa hypätään `label`in osoittamaan kohtaan ja suoritetaan sen jälkeiset komennot. Komento `onintr -` jättää huomiotta kaikki tulevat keskeytyspyynnöt. `Onintr` komentoa voidaan käyttää esimerkiksi väliaikaistiedostojen siivoukseen mikäli komentojono keskeytetään.

6.9 Debuggaus

6.9.1 Bourne shell

Komentojonoja voidaan debugata käyttäen optioita `-n sh` komennolle. Optiolla `-n` shell tarkistaa komentojonon syntaksin suorittamatta yhtään komentoa. Virhetilanteissa shell tulostaa virheilmoitukset. Mikäli virheitä ei löydy mitään ei tulosteta.
Esimerkiksi: `sh -n script_file`

Yleisimmin käytetty debuggauskeino on käyttää `set` komentoa optiolla `-x`, tai antaa `-x sh` komennon optiona. `Set -x` aiheuttaa sen että jokainen komentojonon komento tulostetaan ruudulle sen jälkeen kun komennossa käytetyt laajennukset on tehty ja ennenkuin komento on suoritettu.

Käyttämällä komennon `sh` optiota `-v` jokainen komento komentojonossa tulostetaan kuten se on kirjoitettu.

Debuggaus optiot

Komento	**optio**	**Vaikutus**
`sh -x komentojono`	echo-optio	Tulostaa jokaisen rivin muuttujalaajennuksen jälkeen mutta ennen suoritusta.
`sh -v komentojono`	tulostus-optio	Tulostaa komentojonon rivit ennen suoritusta siten kuin ne on kirjoitettu.
`sh -n komentojono`	syntaksin tarkistus	Tarkistaa komentojono syntaksin suorittamatta yhtään komentoa.
`set -x`	echo-optio päälle	Komentojen suorituksen seuranta komentojonossa.
`set +x`	echo-optio pois päältä	Lopettaa komentojen suorituksen seurannan.

6.9.2 Korn shell

Korn shellissä toimii samat keinot kuin Bourne shellissä komnetojonojen debuggaukseen. Lisäksi ksh komentojonoja voidaa debugata seuraavilla keinoilla.

`Trap` komento tuntee seuraavat “vale” signaalit:

DEBUG	Suoritetaan `trap` komento jokaisen komentojonossa olevan komennon jälkeen.
ERR	Suoritetaan `trap` komento, jos jokin komnetojonon komento palauttaa nollasta poikkevan arvo.
0 tai EXIT	Suoritetaan `trap` komento, jos shellistä poistutaan.

```
#!/bin/ksh
trap 'print "num=$num rivilla $LINENO"' DEBUG
num=1
while [ "$num" -lt 5 ]
do
  let num=num+1
done
print "Summa on $num"
```

6.9.3 C shell

C shellissä komentojonoja voidaan debugata seuraavilla optioilla.

Debuggaus optiot

Komento	**Vaikutus**
`csh -x komentojono`	Tulostaa jokaisen rivin muuttujalaajennuksen jälkeen mutta ennen suoritusta.
`csh -v komentojono`	Tulostaa komentojonon rivit ennen suoritusta siten kuin ne on kirjoitettu.
`csh -n komentojono`	Tarkistaa komentojonosyntaksin suorittamatta yhtään komentoa.
`set echo`	Tulostaa jokaisen rivin muuttujalaajennuksen jälkeen mutta ennen suoritusta.
`set verbose`	Tulostaa komentojonon rivit ennen suoritusta siten kuin ne on kirjoitettu.
`#!/bin/csh -xv`	Laitetaan päälle sekä `echo` ja `verbose`.