UNIX-käyttöjärjestelmä

Luento 5

Timo Savola

<timo.savola@evtek.fi>

28. huhtikuuta 2006

Osa I

Shell-ohjelmointi

Ehtolause

- Lausekkeet suoritetaan jos ehtolausekkeen paluuarvo on 0

```
if ehtolauseke
then lauseke
 lauseke
 ...
fi
```

- Halutessa voidaan määritellä vaihtoehtoisesti suoritettavia lausekkeita

```
if ehtolauseke
then lausekkeet
else vaihtoehtoiset lausekkeet
fi
```

[

- Ohjelma, jolla voi kokeilla ehdon paikkansapitävyyttä
- Suunniteltu käytettäväksi ehtolauseissa:

  `if [ testi ]`

- Merkkijonojen vertailu:

  `[ a = b ]`
  `[ a != b ]`
  ...
- Numeroarvojen vertailu:

  `[ a -lt b ]`
  `[ a -gt b ]`
  ...
- Tiedostojen tarkastelu:

  `[ -e polku ]` – onko olemassa
  `[ -L polku ]` – onko kyseessä symbolinen linkki
  `[ -r polku ]` – onko luettavissa
  ...

Kontrollirakenteet: `for`

- `for` -silmukan lausekkeet suoritetaan kerran jokaista listan alkiota kohden

```
for muuttuja in lista
do lausekkeet
done
```

- Lista sisältää yhden tai useamman välilyönnillä erotetun merkkijonon

- Esimerkkejä:

```
for NIMI in pekka paavo liisa
do
 mkdir kayttajat/$NIMI
done
```

```
for TIEDOSTO in ‘ls dokumentit‘
do
 cp $TIEDOSTO varmuuskopiot/$TIEDOSTO.bak
done
```

Kontrollirakenteet: `while`

- `while` -silmukan lausekkeita suoritetaan kunnes ehto ei enää ole tosi

```
while ehto
do lausekkeet
done
```

- Toimintaperiaate vastaa `if` -ehtolauseketta

- Esimerkkejä:

```
while [ -e /proc/$PID ]
do
 sleep 5s
done
echo "Prosessi $PID kuoli"
```

Kontrollirakenteet: case

- case -rakenne suorittaa merkkijonoa vastaavat lausekkeet

```
case merkkijono in
 vaihtoehto) lausekkeet ;;
 ...
esac
```

- Vaihtoehto voi olla *glob pattern*:

```
*.txt)
abc[12])
```

- Vaihtoehtoja voi yhdistää "|" -merkillä:

```
foo|bar)
```

- Esimerkki:

```
case $TIEDOSTO in
*.txt|README)
 less $TIEDOSTO
 ;;
*)
 echo "Tuntematon tiedosto"
 ;;
esac
```

Kontrollirakenteet: `break`, `continue`

- `for` ja `while` -rakenteissa voidaan käyttää seuraavia komentoja:
  - `continue` siirtää suorituksen silmukan seuraavan iteraation alkuun
  - `break` keskeyttää silmukan suorituksen kokonaan
- Esimerkki:

```
for VUOSI in ‘sort vuosiluvut.txt‘
do
 if [ $VUOSI -lt 1900 ] ; then continue ; fi
 if [ $VUOSI -gt 2006 ] ; then break ; fi
 echo $VUOSI
done
```

true, false

- `true` -komento palaa aina paluuarvolla 0
  - Onnistui $\Rightarrow$ tosi
- `false` -komento palaa aina nollasta poikkeavalla paluuarvolla
  - Virhe $\Rightarrow$ epätosi
- Esimerkki käytöstä kontrollirakenteessa:

```
while true
do
 ls
 sleep 5s
done
```

- Esimerkki käytöstä ehtolauseessa:

```
ECHO=true

if $ECHO ; then echo 'Suoritan stat-komennon' ; fi
stat /tmp/foo
```

read

- `read` -komento on shellin sisäänrakennettu toiminto
- Lukee rivin syötettä muuttujiin
- Esimerkki:

```
while read RIVI
do
 echo $RIVI
done
```

```
read A B C
echo "Sana 1: $A"
echo "Sana 2: $B"
echo "Loput: $C"
```

Funktiot

- Funktio määrittelee uuden komennon

```
nimi () {
 lausekket
}
```

- Funktiossa voidaan käyttää parametreja (kuten scripteissä):

```
kaiuta() {
 echo $1
 echo $2
 echo $3
}
```

- Paluuarvo voidaan määritellä `return` -komennolla:

```
tee-jotain() {
 if [ $1 = yes ]; then return 0; fi
 if [ $1 = no ]; then return 1; fi
 return 50
}
```

Osa II

Tekstinkäsittely

Regular expressions (regex, RE)

- Tapa valita tekstiä
- Vastaa glob patterneja, mutta on paljon monipuolisempi
- Käytetään useissa ohjelmissa

- Glob-erikoismerkkien regex-vastineet:
  - `.` vastaa mitä tahansa yhtä merkkiä
  - `.*` vastaa mitä tahansa merkkijonoa (“yksi merkki monta kertaa”)
  - `[...]` vastaa mitä tahansa sulkeiden sisällä lueteltua merkkiä

- Lisätietoa:
  - `regex(7)`
  - `http://en.wikipedia.org/wiki/Regular_expression`

grep

- *Global Regular Expression Print*
- Etsii syötteestä (tai tiedostoista) rivejä jotka vastaavat regexiä ja tulostaa ne
- Käyttö:

```
grep regex
grep regex tiedostot
```

- grep -komento käyttää yksinkertaistettua regex-syntaksia
- egrep -komento hyväksyy monimutkaisemmat regexit
- Esimerkkejä:

```
grep Pekka nimet.txt
cat *.txt | grep 'P(ekka|aavo)' | sort
grep ^Nimi: kurssit.txt
```

Ed-komennot

- `ed` on UNIXin alkuperäinen tekstieditori
- Toi mukanaan edelleen käytössä olevat editointikomennot

- Komento kohdistetaan yhteen, moneen tai kaikkiin riveihin kerrallaan (osoite)
- Yleisiä komentoja:
  - `a` Lisää tekstiä rivin jälkeen (append)
  - `i` Lisää tekstiä ennen riviä (insert)
  - `c` Vaihtaa rivien sisällön (change)
  - `d` Poistaa rivit (delete)
  - `s` Korvaa `regexin` valitseman tekstin (substitute)

Ed-komennot: s

- Korvauskomento on kätevin ja käytetyin ed-komento
- Syntaksi:

  `s/`*regex*`/`*teksti*`/`
  `s/`*regex*`/`*teksti*`/g`
  `s/`*regex*`/`*teksti*`/`*n*

  `g` tarkoittaa että kaikki rivin osumat korvataan
  n on korvattavan osuman järjestysnumero
- Osoite (kohderivit) voidaan määritellä komennon alussa:

  osoite`s/`*regex*`/`*teksti*`/`
  - Alkuperäinen ed-ohjelma vaatii että komento alkaa pilkulla jos osoitetta ei määritellä

- Esimerkkejä:

```
s/foo/bar/
s/George.*Bush/Amerikan presidentti/
s/vuonna 19../viime vuosisadalla/g
15s/./x/g
```

sed

- *Stream EDitor*
- Käytetään syötteen (tai tiedostojen) ei-interaktiiviseen editointiin ed-komentojen avulla
- Lopputulos tulostetaan (tai kirjoitetaan takaisin lähdetiedostoihin)

- Käyttö:

```
sed komento
sed komento tiedostot
sed komento -i tiedostot
sed -e komento1 -e komento2 ...
```

- Esimerkkejä:

```
sed s/pekka/Pekka/g <nimet.txt >nimet2.txt
echo ABC DEF ABC|sed -e s/ABC/GHI/2 -e 's/ /, /g'
sed -i 's/ *$//' tiedosto.txt
```

vi

- *VIsual editor*
- Visuaalinen korvike ed:ille

- Vi:n käyttö tapahtuu pääasiassa kahdessa eri tilaa (mode):
  - Oletuksena ollaan komentotilassa
  - Syöttötilassa voidaan kirjoittaa tekstiä normaalisti
  - Komentotilaan palataan painamalla ESC

- Joitain komentoja:

`i`	Siirry syöttötilaan (insert)
`:e`	Avaa tiedosto (edit)
`:w`	Tallenna tiedosto (write)
`:q`	Poistu ohjelmasta (quit)
`:q!`	Poistu ohjelmasta tallentamatta muutoksia

- Komentotilassa voidaan myös antaa ed-tyylisiä komentoja:

  `:%s/`*`regex`*`/`*`teksti`*`/`