UNIX/LINUX-PERUSKURSSI, MUUTTUJAT JA KOMENTOKIELI

Tämä dokumentti kuvaa Unixin komentotulkkien ominaisuuksia sekä johdatuksen Unixin komentokieleen ja komentoskriptien luomisen. Skriptikielenä esitetään Bourne-komentotulkin skriptikieli. Dokumentin lopussa ovat tähän osuuteen kuuluvat tehtävät sekä ohjeistus sille, miten tehtävät tulee palauttaa.

Lisää aiheeseen löydät muun muassa seuraavista materiaaleista:

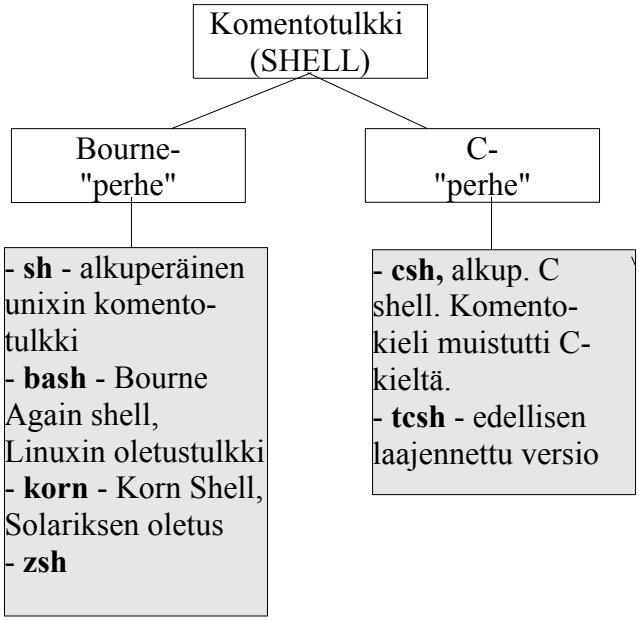
http://users.metropolia.fi/~kuivi/linux/kom.php, Opettajan oma komentotulkkiohje.

http://koti.welho.com/jkajaste/bash.html – Jaakko Kajasteen Bash-ohje

http://linux.fi/wiki/Bash-skriptaus – Linux.fi-wikistä löytyy myös tietoa asiasta

1. Komentotulkki

Komentotulkki (Shell) on käyttöliittymä käyttäjän ja kernelin välillä. Unixissa komentotulkkeja on useita erilaisia, nämä jaetaan usein kahteen erilliseen osaan, Bourne- ja C-osaan, puhutaan myös komentotulkkiperheistä:



Komentotulkki miellletään usein pelkäksi tekstipohjaiseksi komentoriviksi, mutta käytännössä myös graafinen liittymä on myös komentotulkki, tosin sitä kutsutaan yleensä GUI:ksi, eli Graphical User Interface.

Komentotulkki on ensimmäinen ohjelma, joka ladataan, kun järjestelmään kirjaudutaan. Komentotulkki tulkitsee käyttäjän antamat komennot ja käynnistää ohjelmat taikka antaa virheilmoituksen, jos se ei ole ymmärtänyt mitä käyttäjä on tekemässä. Se myöskin huolehtii jokerimerkkien tulkinnasta, syötön ja tulostuksen ohjaamisesta, putkittamisesta ym. Yhteisellä sopimuksella jokainen tulkki käyttää samoja merkkejä näihin.

Komentotulkit yleensä sisältävät myös ohjelmointikielen. Puhutaan usein ns. pienistä ohjelmointikielistä. Unixin komentotulkkien ohjelmointikielet ovat yleensä varsin ilmaisuvoimaisia ja niillä voi luoda hyvinkin vaativia ohjelmia, jäljempänä komentoskriptejä.

Alkuperäinen Bourne-komentotulkki on ominaisuuksiltaan varsin rajoittunut. Rivin korjaaminen taikka jälkikäteen käyttäminen olivat täysin mahdottomia asioita. Riviä saattoi korjata vain Delete- taikka backspace -näppäimillä. Bournen nimi on *sh* ja se on */bin*-hakemistossa.

Huom! Koska Filesystem Hierarchy Standard (FHS) edelltyttää, on kaikissa Unix-tyyppisissä käyttäjärjestelmissä oltava sh-niminen ohjelma /bin-hakemistossa. Siksi Linuxissakin tämä on ja se on symbolinen linkki bash-komentotulkkiin. Sen takia Linuxin sh saattaa näyttää siltä, että se osaa paljon sellaisia asioita, joita sh:n ei kuuluisi osata.

2. Alias-ominaisuus

Aliasta voisi kutsua eräänlaiseksi komennon "lempinimeksi". Aliasten avulla voidaan määritellä omia nimiä taikka lyhenteitä komennoille. Varsinkin näin voidaan helpottaa usein käytettyjen komentojen käyttöä: lyhyellä aliaksella hoidetaan monimutkaisen komennon kirjoittaminen.

Aliakset ovat voimassa täsmälleen siinä komentotulkissa, jossa ne on määritelty. Jos halutaan niiden olevan voimassa aina, pitää ne määritellä jossakin komentotulkin asetustiedostossa. Bournessa tällainen tiedosto on nimeltään *.profile*, Bash:ssa taas *.bashrc* on oikea paikka määritellä aliaksia.

Komennolla *alias* saadaan näkymään järjestelmään luodut aliakset. Lopputulos voi näyttää esim. tältä:

```
[kuivanen@chimay test]$ alias
alias l.='ls -d .* --color=tty'
alias ll='ls -l --color=tty'
alias ls='ls --color=tty'
alias mc='. /usr/share/mc/bin/mc-wrapper.sh'
alias vi='vim'
alias which='alias | /usr/bin/which --tty-only --read-alias --show-dot --show-ti
lde'
[kuivanen@chimay test]$
```

HUOM! Jotkut distribuutiot – esim. SuSE saattavat sisältää huomattavasti enemmän erilaisia valmiiksi määriteltyjä aliaksia kuin jotkut toiset. Tämä saattaa olla jossain määrin hämääväkin, koska esimerkiksi yhteenkirjoitettu *cd..* on määritelty aliakseksi komennolle *cd ..* Näin ollen voi helposti tulla tunne, että Linux onkin varsin joustava siinä, miten komento ja sen parametrit kirjoitetaan ja tämä ei oikeasti pidäkään paikkaansa...

Aliaksia voi itse määritellä seuraaavasti:

```
alias l='ls'
```

joka määrittää komennolle ls aliaksen l.

Aliaksia voi poistaa komentamalla *unalias*:

```
unalias l
```

poistaa aliaksen l.

Jos on määritetty alias, joka on samanniminen kuin jokin komento (vaikkapa cp), saadaan alkuperäinen komento suoritettua kirjoittamalla komennon eteen kenoviiva (esim. *cp*).

3. History

History-ominaisuus tallettaa vanhoja komentoja uudelleenkäyttöä varten. Komentamalla *history* näkee, mitä aiemmin on tehty ja bash-komentotulkissa voi komennon numeron perusteella suorittaa saman komennon uudelleen. History-komennon tuloste voi näyttää esimerkiksi tältä:

```
 998  man tar
 999  cd Tyomaa/Unix/coursefiles
1000  ls -l
1001  cd test
1002  mkdir test
1003  vi kirje1
1004  vi kirje2
```

Esimerkiksi rivillä 999 olevan komennon voisi toistaa kirjoittamalla komentoriville

```
!999
```

Eli huutomerkki ja sen jälkeen komennon numero. Historyyn liittyy seuraavia ominaisuuksia:

Komento	***Merkitys***
!!	Suorittaa edellisen komennon uudelleen
!*	Suorittaa komennon edellisen komennon argumenteilla
!$	Suorittaa komennon edellisen komennon viimeisen argumentin kanssa
!nro	Suorittaa komennon numero nro uudelleenkäyttöä
!nro:p	Näyttää listalta komennon numero nro muttei suorita sitä.

Korn-komentotulkissa on vastaava toiminto. Sisäinen komento *r* toistaa vanhoja operaatioita.

4. Komentotulkin muuttujat

Komentotulkit tarvitsevat joukon määrityksiä toimiakseen. Pääosa näistä määrityksistä on talletettu komentotulkille määriteltyihin muuttujiin. Täyden komentotulkin muuttujalistan löydät kirjoittamalla komentoriville komennon

```
set
```

Eräitä komentotulkin muuttujia:

HOME - käyttäjän kotihakemisto on talletettu tähän muuttujaan. Muista myös tämä: **~**

PATH - hakupolku. Määrittää ne hakemistot, mistä Unix hakee suoritettavaa ohjelmaa

PS1 - käytettävä ensisijainen komentokehote

LOGNAME - käyttäjän tunnus, jolla hän on kirjautunut sisään

SHELL - käytössä oleva komentotulkki.

Bash-komentotulkissa (ja yleisesti sh-pohjaisisa) nämä komentotulkin omat muuttujat kirjoitetaan isoilla kirjaimilla. Csh-pohjaisissa taas pienellä. Tässä dokumentissa keskitytään sh-pohjaisiin muuttujamäärityksiin.

Muuttujia voi myös määritellä itse. Pääsääntöisesti näiden tarve tulee esille komentoskriptien käytössä, mutta on tilanteita, joissa joku ohjelma toimiakseen tarvitsee jonkin oman määrityksen. Bashissa komentotulkin muuttuja määritetään seuraavasti:

```
muuttuja=arvo
```

Eli muuttujan nimi, =-merkki ja muuttujan sisältö kirjoitetaan yhteen ilman välilyöntejä. Esimerkkejä:

```
luku=1
hei="Terve"
```

Muuttujaan viitataan $-merkillä:

```
echo $hei
```

tulostaa muuttujan hei sisällön.

Muuttuja voidaan poistaa komennolla

```
unset muuttujannimi
```

Muuttujat ovat voimassa vain siinä komentotulkissa, missä ne on määritelty. Jos halutaan laajentaa muuttujan vaikutusaluetta, se on määriteltävä export-komennolla:

```
export muuttuja
```

laajentaa muuttujan näkyvyyttä siten, että se on olemassa kaikissa tästä komentotulkista käynnistetyissä komentotulkeissa. Jos muuttujasta halutaan tehdä sellainen, että se on olemassa jokaisessa komentotulkeissa mitä käynnistetään, on määritys kirjoitettava komentotulkin asetustiedostoon. Esimerkiksi Bash:ssa sopiva paikka tälle on *.bashrc*.

5. Kolmenlaisia lainausmerkkejä

Komentotulkit ymmärtävät erilaiset lainausmerkit eri tavoin. Tämän hahmottaminen on tärkeää, koska näiden ero näkyy muuttujien tulkintana.

Lainausmerkit (") aiheuttavat muuttujien tulkinnan:

```
echo "Kotihakemistosi on $HOME"
```

Heittomerkit (') taas käsittelevät sisäänsä jäänyttä merkkijonoa pelkkänä merkkijonona:

```
echo 'Kotihakemisto on talletettuna $HOME-muuttujaan'
```

Kolmas merkintä on ns. gravis (`, Shift-` ja välilyönti). Se aiheuttaa sisäänkirjoitetun komennon suorittamisen ja komennon tulosteen sijoittamisen siihen paikkaan:

```
echo "Today is `date +%A`"
```

Kokeile edellisiä komentoja ja ymmärrä niiden toiminta!

6. Komentotulkkien asetustiedostot

Jossakin on komentotulkin perusasetukset tehtävä. Tätä varten on olemassa erilaisia asetustiedostoja. Näiden nimet ja toiminta vaihtelee riippuen komentotulkista. Bourne-komentotulkki käyttää seuraavia asetustiedostoja:

/etc/profile - järjestelmän kaikille käyttäjille yhteiset määritykset

~/.profile - jokaisen henkilökohtaiset asetukset. Tiedosto suoritetaan aina kun järjestelmään kirjaudutaan.

Bash-komentotulkin asetustiedostojen valikoima on hieman laajempi kuin Bournen:

/etc/profile - kuten edellä

~/.bash_profile- tehdään kun kirjaudutaan sisään

~/.bashrc - käynnistetään joka kerta, kun käynnistetään bash-komentotulkki

~/.bash_logout - uloskirjautuessa tehtävät toimenpiteet.

7. Tehtäviä

Nämä tehtävät esitetään tunnilla tuntitehtävinä. Huomaa, että dokumentin lopussa on vielä lisää tehtäviä liittyen komentoriviohjelmointiin.

1. Tee alias seuraaville operaatioille. Suluissa tehtävän aliaksen nimi:
   a) clear (c)
   b) cd; ls (home)
   c) cp -i (copy)
   d) ps -ef | sort | less (p)
2. Mitä pitää tehdä, että ko. aliakset olisivat olemassa joka kerta, kun avaat uuden pääteikkunan?
3. Poista alias copy.
4. Olet määrittänyt aliaksen '*ls*', joka tulostaa pitkän listauksen pelkkien tiedostonimien sijaan. Tarvitset kuitenkin erääseen tehtävään alkuperäistä *ls*-komentoa. Miten saat käytettyä sitä aliaksen sijaan poistamatta aliasta *unalias*-komennolla?
5. Suoritit muutama komento sitten komennon *find / -name core -exec rm {} \;* Et viitsisi millään kirjoittaa komentoa uudelleen. Mainitse vähintään kaksi tapaa, jolla saat toistettua komennon kirjoittamatta sitä kokonaan uudelleen:
6. Määritä muuttuja nimeltään PII ja aseta sille arvoksi 3.14. Mitä tekevät seuraavat komennot:
   a) echo PII
   b) echo $PII
   c) PII=6.28
   d) unset PII
7. Määritä uudelleen muuttuja PII ja anna sille arvo kuten edellisessäkin tehtävässä. Käynnistä sen jälkeen samasta komentotulkista uusi komentotulkki bash-komennolla. (sinulla on siis ikäänkuin kaksi komentotulkkia päällekkäin) Näkyykö muuttuja tässä komentotulkissa? (kokeile echo $PII). Jos ei näy, mitä pitäisi muuttujan määrittelyn lisäksi tehdä, jotta muuttuja näkyisi myös täällä? (ratkaisu ei ole se, että määritellään muuttuja uudelleen...)
8. Avaa valikosta kaksi erillistä komentotulkkia. Miten saat tällaisessa tapauksessa saman muuttujan näkymään molemmissa?
9. Ota selvää esimerkiksi Googlea apuna käyttäen, mitä ovat korn-shellin bash-komentulkin asetustiedostoja vastaavat asetustiedostot.

Linuxin mukana tulevat komentotulkit (mm. bash, tcsh, ksh, jne…) sisältävät ohjelmointikielen, joka on varsin tehokas ja ilmaisuvoimainen. Tähän yhdistettynä unix-maailmasta tutut tehokkaat apuohjelmat, voi sanoa, ettei Linuxissa jonkun pienen ohjelmointiongelman ratkaisemiseen tarvita mitään muuta ulkoista ohjelmointikieltä.

HUOM! Työtilassa on pdf nimeltään *1-bash-skriptit.pdf.* Sieltä löydät myös arvokasta tietoa skriptien toiminnasta. Suosittelen lukemaan ajatuksen kanssa!

1. Ohjelmien suoritus unix-pohjaisissa käyttöympäristöissä

Unix poikkeaa tässä Windows-maailmasta ratkaisevasti. Kun Windows-maailmassa etsitään suoritettavaa ohjelmaa ensiksi työhakemistosta ja mennään vasta sitten hakemaan ns. hakemistopolusta (PATH), unix-pohjaisissa käyttöjärjestelmissä mennään suoraan hakemaan ohjelmaa hakemistopolusta. Tällä estetään ns. ohjelmien kaappaukset.

Myöskään tiedoston nimen pääte ei määrää sitä, minkä tyyppinen ohjelma on kyseessä. Jos unixissa laittaa ohjelman (skriptin) nimeksi ohjelma.sh, on kirjoitettava näkyviin koko ohjelman nimi, viimeistä kirjainta myöten.

Skriptille tulee aina ennen suoritusta antaa suoritusoikeudet. Suoritusoikeus annetaan seuraavalla komennolla:

```
chmod u+x ohjelma.sh
```

Esimerkki antaa omistajalle (u – user) lisää (+) suoritusoikeuden (eXecute)

2. Miten skripti käynnistetään unixissa?

Jos työhakemisto on polussa, voi skriptin käynnistää aivan samoin kuin Windowsin komentoriviltä, eli näin:

```
ohjelma.sh
```

Jos työhakemisto ei ole polussa, pitää ohjelman sijainti kertoa. Yksinkertaisimmillaan se voidaan sitoa työhakemistoon näin:

```
./ohjelma.sh
```

Piste viittaa aina työhakemistoon. Näin ollen yllä olevassa esimerkissä kerrotaan, että ”suorita työhakemistossa oleva ohjelma.sh-tiedosto”

Kolmas tapa on käynnistää skripti näin:

```
. ohjelma.sh
```

Tässä siis laitetaan ensiksi piste, sitten välilyönti ja lopuksi skriptin nimi. Tämä ajaa skriptin tässä komentotulkissa eikä siis käynnistä uutta kuten nuo edelliset tekevät.

Neljäs tapa on käynnistää skripti komentotulkin parametrinä:

```
bash ohjelma.sh
```

Tässä tavassa voi komentotulkin optioita käyttäen suorittaa paremmin debuggausta siitä, mitä skripti oikein tekee. Eli virheen etsinnässä hyödyllinen tapa.

3. Mitkä hakemistot ovat polussa (PATH)?

Perinteisesti bin-nimiset hakemistot ovat tarkoitettu ohjelmille (*/bin, /usr/bin, /usr/local/bin*, jne). Myös käyttäjän kotihakemiston alle tehty bin-hakemisto on olemassa valmiina polussa, vaikkei itse hakemistoa vielä olekaan. Näin ollen jos haluat tehdä sellaiseen paikkaan skriptisi, josta voit suorittaa ne missä vain, tee kotihakemistoosi

hakemisto bin, jonne talletat kaikki skriptisi. Seuraavat tehtävätkin voit toteuttaa siellä.

4. Ensimmäinen skripti

Kaikki itseään kunnioittavat ohjelmointioppaat aloittavat seuraavalla esimerkillä. Kirjoita se haluamallasi editorilla ja anna sille nimeksi vaikkapa hello:

```
#!/bin/sh
echo ”Hello world”
```

Anna suoritusoikeudet sille ja testaa se.

Ensimmäinen rivi määrittää käytettävän komentotulkin. Sh on Unixien perus-komentotulkki, Bourne Shell. Sitä käytetään yleisimmin, koska se on mukana kaikissa unixeissa. Perus-linux-komentotulkit, kuten bash ja zsh, perustuvat tähän komentotulkkiin ja niiden komentokieli on jokseenkin samanlainen. C-komentotulkkihaaran kieli on taas näistä poikkeavaa. Tähän haaraan ei tällä kertaa puututa. Yleensäkin suurin osa komentoriviohjelmointioppaista keskittyy ns. Bourne-haaran kieliin.

5. Oppilaitoksen shell-koneet

Jos haluat kokeilla skriptiohjelmointia oppilaitosten koneilla, käytä konetta nimeltään *edunix.metropolia.fi*. Tällä koneella skriptien suorittaminen on sallittu. Koneessa *shell.metropolia.fi*:llä ei skriptien suorittaminen ole sallittua. Tosin asian voi kiertää ajamalla skriptit bash-komentotulkin parametreinä (ks yllä).

6. Kokeile!

HUOM! Palautettavien tehtävien osuus alkaa näiden jälkeen! Tee kuitenkin nämäkin, sillä niissä käsitellään samoja asioita kuin palautusosiossa.

1. Lisää aiemmin esillä olleeseen hello-skriptiin kahden rivin väliin rivi, jolle kirjoitat komennon clear. Testaa skriptiä ja katso, miten tulos muuttui.
2. Kirjoita seuraava skripti:

   ```
   #!/bin/bash
   echo ”Moi, mikä sinun nimesi on?”
   read nimi
   echo ”Hei, minusta $nimi on kaunis nimi.”
   ```

   Mitä se tekee?

   HUOM! Vaikka tässä kysellääänkin read-ohjelmalla käyttäjältä muuttujalle sisältöä, Unix-maailmassa on tavallista, että asiat annetaan komentorivin parametreinä! Tätä tapaa ei siis käytetä paljoa todellisuudessa.
3. Vaihda jälkimmäiseen echo-lauseeseen kaksinkertaisten lainausmerkkien tilalle yksinkertaiset (heittomerkki – ’). Kokeile nyt samaa skriptiä. Mitä havaitsit?
4. Lisää samaan skriptiin seuraava rivi loppuun:

   ```
   echo ”Tiesitkö, että tänään on vuoden `date +%j`.s päivä?”
   ```

Tuo ”hipsu”-merkki löytyy näppäimistöltä kysymysmerkin vierestä. Saatat tarvita välilyöntinäppäimen painallusta merkin jälkeen. Mitä tulostui?

5. Toteuta seuraava skripti. Ole tarkkana kirjoitusasun kanssa!

   ```
   #!/bin/bash
   echo ”Mietin yhtä lukua, yritä arvata se:”
   declare i arvaus
   ```

```
read arvaus
luku=arvaus + 1
echo ”Ajattelin lukua $luku. Hävisit niukasti!”
```

6. Ja nyt lopetamme read-komennon käyttämisen. Tee skripti, joka saa parametrinään kaksi lukua ja kertoo niiden summan. Eli jotain tällaista:

```
$ summa 4 5
Lukujen 4 ja 5 summa on 9.
```

   Vihje: muuttujat $1 ja $2 ovat rivin kaksi ensimmäistä parametriä (ks teht. 7)

7. Jos muutat edellisen tehtävän laskemaan summan sijasta tulon, mitä tapahtuu?

8. Seuraava skripti kertoo nimensä sekä parametrinsä:

```
#!/bin/bash
echo ”Tämän skriptin nimi on $0”
echo ”Ensimmäinen parametri on $1 ja toinen $2”
echo ”Annoit yhteensä $# parametriä, jotka olivat: $*”
```

Kokeile skriptiä antamalla komennon perään muutama parametri, vaikkapa näin:

```
param.sh eka toka kolmas neljas
```

Numerot 0 – 9 ovat siis varattu komentorivin parametreille. Normaalisti tieto välitetäänkin skripteille komentorivin parametreillä eikä millään read-lauseella.

9. Mitä seuraava tekee:

```
#!/bin/bash
if [ $1 –gt 10 ]
then
   echo ”parametri oli suurempi kuin 10”
else
   echo ”parametri oli 10 tai pienempi”
fi
```

10. Entä seuraava:

```
#!/bin/bash
declare –i summa=0
for luku in 1 2 3 4 5 6
do
  summa=summa + luku
done
echo "Lukujen 1 - 6 summa on $summa."
```

11. Miten saisit yo. skriptin lukemaan luvut parametreinä?

PALAUTETTAVAT SKRIPTIT

Toimintaohje: Skriptit pakataan *tar.gz*-paketiksi ja lähetetään *TUUBIN kautta*. Skriptit on palautettava **tenttipäivään mennessä.** Kaikki tehtävät on muutenkin palautettava siihen mennessä.

1. Tee skripti nimeltään info, joka tekee seuraavat asiat:
   - määrittelee aluksi käytettävän komentotulkin (bash, sh, joku muu)
   - tyhjentää kuvaruudun
   - tervehtii käyttäjää tämän nimellä (login name)
   - kertoo käyttäjän työhakemiston
2. Tee skripti, joka tulostaa allekkain kaikki sille annetut parametrit sekä kertoo lopuksi niiden lukumäärän. Testaa skriptin toimivuus.

   Vihje: tässä tehtävässä voi käyttää joko for- taikka while-lausetta. Ensimmäisen kanssa asia menee vähän suoraviivaisemmin...
3. Skripti tulostaa seuraavanlaisen valikon:

```
Mitä haluat tehdä?
1) katsoa pitkän tiedostolistauksen
2) katsoa tiedostolistauksen tiedostotyyppien kanssa
3) tulostaa päiväyksen
Anna valintasi:
```

   Toteuta nyt skripti siten, että se toteuteuttaa myös ko. asiat. (esim. ykkösestä pitkä tiedostolistaus.) Väärästä valinnasta pitää tulla virheilmoitus.

   Vihje: case-lause. If:llä onnistuu myös, muttei ole järkevä.
4. Toteuta esimerkkien tehtävä 5 siten, että arvattava luku annetaan parametrinä.

   Vihje: ensimmäinen parametri komentoriviltä on $1. Määrittele kokonaisluku, joka saa arvokseen tämän parametrin arvon lisättynä yhdellä.
5. Tee skripti, joka laskee silmukassa yhteen luvut 1 - 10.

   Vihje: ks for-silmukan muoto aiemmin mainitusta pdf:stä diasta 42....
6. Muuta skriptiä siten, että käyttäjä voi antaa komentoriviltä parametrinä luvun, mihin asti lasketaan. Esim. laske 23 laskisi yhteen luvut väliltä 1 - 23. Voit olettaa, että annetut luvut ovat positiivisia.
7. Toteuta lopuksi sellainen versio skriptistä, joka ei hyväksy kuin yhden parametrin. Eri määrällä parametrejä tulee tulostaa virheilmoitus eikä tehdä yhtään mitään.

   Vihje: Testaa ensiksi parametrien lukumäärä. Senhän löytää muuttujasta $#....