linux: komennoista

linux

- Komentojen käyttö
  `komento -opt1 -opt2 argumentti`
- Esimerkiksi `ls -s *.dat`
  tulostaa työtiedoston `.dat`
  loppuiset tiedostot ja niiden koon
- Esimerkiksi `ls -l *.dat`
  tulostaa samojen tiedostojen, koon,
  viimeisen version ajan, yms
- Molemmat yhtä aikaa `ls -l -s *.dat`
  on sama kuin `ls -ls *.dat`
  on sama kuin `ls -sl *.dat`
- `whatis arg` lyhyt `arg` kuvaus
- `man arg` näyttää `arg` manuaalisivut
- `info arg` näyttää `arg` infosivut
- `arg --help` komennon opaste
- Kokeile `whatis cp`, `man cp`, `info cp`,
  `cp --help` (Poistuminen tarvittaessa: `q`)

linux

- Luodaan kaksi tyhjää tiedostoa
  `touch aaa` ja `touch bbb`
- Annetaan komento `cp -i aaa bbb`
  tulostuu kysymys
  `cp: overwrite 'bbb'?`
  johon voi vastata `y` tai `n`
- Sarkain $\longrightarrow\!|$ täydentää komentoja
- Esimerkiksi `ho` $\longrightarrow\!|$ tuottaa `host`
- $\longrightarrow\!|$ täydentää myös tiedostojen tai hakemistojen nimiä
- Annetut komennot tallentuvat komentopinoon, jota voi selata nuolinäppäimillä $\uparrow$ ja $\downarrow$
- Selattujen komentojen sisällä voi liikkua nuolinäppäimillä $\leftarrow$ ja $\rightarrow$, sekä samalla edioida uusia merkkejä

linux: komennoista

linux

- Esimerkiksi `history 10` tulostaa viimeiset 10 annettua komentoa
- Komennon ulosanti voidaan ohjata tiedostoon tai toiseen komentoon
- Esimerkki komennosta tiedostoon: `ls > aaa` tulostaa työhakemiston sisällön tiedostoon `aaa`
- Näin luodun `aaa` tiedoston voi tulostaa komennolla `cat aaa`
- Esimerkki komennosta komentoon: `grep 'ab' ~username/* |sort -` etsii hakemistosta `~username/` tiedostot `*`, joissa esiintyy merkkirivi `ab` Komento `sort -` tulostaa nämä tiedostot aakkosjärjestyksessä
- Useita komentoja voi siis "putkittaa" (engl. pipe) `|` merkin avulla
- "Pidempiä putkia" EnOlKoTa

linux

- Komennoilla on "oletuskäyttäytyminen", jota voi muuttaa ja luoda eri muunnelmia
- `cp file1 file2` kopioi tiedoston `file1` sisällön tiedostoon `file2`, `cp` oletuskäyttäytyminen on, että `file2`:n päälle kirjoitetaan mitään kysymättä
- Komento `cp -i file1 file2` kysyy `cp: overwrite 'file2'`
- `alias cp='cp -i'` määrittelee uuden komennon `cp`, joka tekee kuten `cp -i`
- Editoidaan tiedosto `emacs .Myalias`

```
alias c='clear'
alias cp='cp -i'
alias rm='rm -i'
```

- Ajetaan komennolla `source .Myalias`
- Tämän jälkeen `c` tyhjentää näytön, sekä `cp` tai `rm` eivät kopioi tai tuhoa kysymättä
- `ls -a` listaa työtiedoston sisällön, mukaan lukien merkillä `.` alkavat tiedostot

linux: Alias

linux: Alias

- Pitkät usein käytetyt komennot
  `alias nimi='pitkä komento'`
  kannattaa lyhentää
- Esimerkki 1: `alias` lyhennys

`alias Pr='lpr -PHP-Laserjet-1200'`

  mahdollistaa tiedoston `file`
  printtauksen jonoon `HP-Laserjet-1200`
  komennolla `Pr file`
- Muuten pitää jokaista tiedostoa
  `*` printatessa kirjoittaa
  `lpr -PHP-Laserjet-1200 *`
- Esimerkki 2: valinnalla
  `alias python='python3'`
  varmistetaan, että komennolla
  `python` käytetään uusinta
  (vuonna 2015) **python** versiota
- Toimivaa `alias` komentoa ei kannata "keksiä/määritellä/etsiä" uudestaan

linux: alias

- Valitut `alias` komennot kannattaa tallentaa tiedostoon
  `/home/username/.bashrc`
- Tämä alustustiedosto luetaan aina istunnon alkaessa tai uuden **linux** komentotulkin käynnistyessä
- Tapa 1: editoidaan tiedostoon `/home/username/.bashrc` rivit

```
alias c='clear'
alias cp='cp -i'
alias rm='rm -i'
alias python='python3'
```

- Tapa 2: editoidaan samat 4 riviä tiedostoon `/home/username/.Myalias` ja 1 rivi

```
source .Myalias
```

  tiedostoon `/home/username/.bashrc`
- Tapa 2 parempi: Miksi?
- Ensin `cp .bashrc .bashrcOLD`
  Sitten `emacs .bashrc &` Miksi?

LaTeX: Matemaattiset kaavat

LaTeX:Kaavat

- Matemaattiset kaavat rajataan merkillä **$**
- Tekstin sisään (in line math) tarkoitetut kaavat rajataan kummaltakin puolelta **$**:llä
- Esim: tekstisyöte

```
Mitä on $a_1 \sin{y},$
jos $a_1=7$ ja $y=89$.
```

tuottaa

Mitä on $a_1 \sin y$, **jos** $a_1 = 7$ **ja** $y = 89$

- Symbolit kursiivia: kts a_1, y yllä
- Luvut ja funktiot ei kursiivia: kts 7, $\sin$ yllä
- Kursiivin voi estää (esim: luonnon vakiot)
- Esim: `valon nopeus $\mathrm{c}$`
  tuottaa **valon nopeus** c, mutta

```
valon nopeus $c$
```

tuottaisi **valon nopeus** c

LaTeX: Kaavat

- Tekstistä erottumaan (display math) tarkoitetut kaavat rajataan **$$**:llä
- Esim: tekstisyöte

```
Mitä on $$a_1 \sin{y},$$
jos $a_1=7$ ja $y=89$.
```

tuottaa

Mitä on

$$a_1 \sin y,$$

jos $a_1 = 7$ **ja** $y = 89$.

- Merkki _ tuottaa alaviitteen
- Esimi: `$a_{b+c}$` tuottaa a_{b+c}, missä { ja } rajaavat alaviitteen
- Esim: `$a^{b+c}$` tuottaa yläviitteen a^{b+c}
- Esim: `$a^{b+c}_{d+e}$` tuottaa molemmat a^{b+c}_{d+e}

LaTeX: Matemaattiset kaavat

LaTeX: Kaavat (kuva:@www.spreadshirt.co.uk)

- **emacs** on LaTeX "ystävällinen"
- Kursori pomppii editoidessa

```
${
{ \sum_{i=1}^{n+1}\cos{i\pi\alpha} }
\over
{ \sum_{i=1}^{n+1}\sin{i\pi\alpha} }
}$
```

  joka tuottaa $\frac{\sum_{i=1}^{n+1} \cos i\pi\alpha}{\sum_{i=1}^{n+1} \sin i\pi\alpha}$
- Kreikkalaisille kirjaimille omat komennot: esim `\alpha` ja `\pi` yllä
- Matemaattisille funktiolle omat komennot: esim `\cos` ja `\cos` yllä
- Matemaattisille merkinnöille omat komennot: esim `\sum` ja `\over` yllä
- Mitään ei tarvitse/kannata opetella ulkoa, koska löytyvät esimerkiksi täältä | www |

LaTeX: Kaavat

- Lähes minkä tahansa symbolin komento löytyy täältä | www |

LaTeX: Matemaattiset kaavat

LaTeX: Kaavat

- $$ rajatuista kaavoista puuttuu numerointi
  ⇒ Viittaaminen tekstin sisällä vaikeaa
- LaTeX tekstisyöte

```
Tähän kaavaan
\begin{equation}
E = m \mathrm{c}^2
\label{Ejuttu}
\end{equation}
voi viitata (Kaava \ref{Ejuttu})
```

tuottaa

Tähän kaavaan

$$E = mc^2 \tag{1}$$

voi viitata (Kaava 1)

- Komento **\ref{Ejuttu}** viittaa tunnisteeseen **\label{Ejuttu}**
- Kaavan numero pysyy oikeana, vaikka lisätään uusia numeroituja kaavoja

LaTeX: Kaavat

- LaTeX tekstisyöte

```
\begin{eqnarray}
a_x     & = & \int_a^b x dx \\
\sin{x}& = & a_x \nonumber \\
d       & = & x
\end{eqnarray}
```

tuottaa

$$\begin{aligned} a_x &= \int_a^b x dx && (2) \\ \sin x &= a_x \\ d &= x && (3) \end{aligned}$$

- Sarake vaihtuu = **&**, Rivi vaihtuu = ****
- Ei numeroa komennolla **\nonumber**
- **latex tiedosto** ajettava kahdesti ⇒ Numerointi kunnossa ⇒ Ei tulostu kaavan ja viitenumeron kohdalla **??**

python

python

- **python**: paljon erilaisia valmiita moduleja, joita ladataan “tarpeen mukaan”
- Matemaattiset funktiot: Numpy | www | importoidaan komennolla `import numpy`
- Importoinnin jälkeen esimerkiksi tämän modulin sisällön voi aina tarkistaa komennolla `dir(numpy)`
- Nimeä voi muuttaa esimerkiksi

```
python
import numpy as np
a=np.arange(3)
print(a)
```

tulostaa `[0 1 2]`

- Tieteellinen laskenta: Scipy | www | importointi `import scipy`
- Tulosten visualisointi: Pylab | www | importointi `import pylab`

python

- Pythonista on olemassa interaktiivinen versio `ipython` | www |
- Kokeile **linux** komentotulkissa `ipython -pylab` ⇒ `In [1]` ilmestyy `a=numpy.ones(3) ; print(a)` tulostaa `[ 1. 1. 1.]`
  Anna sen jälkeen komento `a.` →|
  ja `ipython` kertoo kaiken, mitä voit tehdä muuttujalle `a`, jos aloitat muodolla `a.`
- `ipython`:ssa voi vaikkapa kokeilla erilaisia komentoja ennen kuin kirjoittaa ne editoimansa ohjelman sisään
- Kokeile **linux** komentoa `pwd`. Se toimii `ipython`:ssa, mutta ei tavallisessa `python`:ssa
- **octave**: kaikki “ladattu valmiiksi” ja **linux** komennot toimivat

python ja octave

python

- Numeeriset muuttujatyypit pitävät sisällään lukuarvoja. Esimerkkejä **python**:n numeerisia muuttujatyyppejä

  `int` = kokonaisluku ≡ esim **`a=4`**

  `float` = reaaliluku ≡ esim **`a=3.45`**

  `complex` = kompleksiluku ≡ esim **`a=complex(3,4)`** tai **`a=3.+4j`**
- Tyhjän muuttujan luominen: **`a=int()`**, **`a=float()`**, **`a=complex()`**
- Kokeillaan **python**:ssa

```
a=1 ; b=1 ; c=a+b
print(type(a),type(b),type(c))
```

  ⇒ tyyppi ei muutu
- Kokeillaan **python**:ssa

```
a=1 ; b=2.2 ; c=a+b
print(type(a),type(b),type(c))
```

  ⇒ tyyppi muuttuu tarkkuuden mukaan

octave

- Kokeillaan **octave**:ssa `a=1 ; class(a)` tulostaa **`ans = double`** eli ei integer
- Kompleksiluvun luominen **octave**:ssa `z=3+4*i ; abs(z)` tulostaa **`ans = 5`** eli **`|z|`** arvon
- Kokeillaan **octave**:ssa `a=1 ; isnumeric(a)` tulostaa **`ans = 1`** eli kertoo, että **`a`** on numeroarvoinen muuttuja (s.o. joko kokonais-, reaali- tai kompleksiluku)
- Kokeillaan **octave**:ssa `a=1 ; islogical(isnumeric(a))` tulostaa **`ans = 1`** eli kertoo, että **`isnumeric(a)`** on “looginen” muuttuja (s.o. **`1`**=True, **`0`**=False)
- **octave**: tietoa numeerisista (myös muista) muuttujatyypeistä löytyy täältä | www |
- Muuttujatyyppejä ei tarvitse opetella ulkoa

python ja octave

python ja octave

- Peruslaskutoimitukset

python	octave	
+	+	summa
-	-	erotus
*	*	kertolasku
**	** tai ^	potenssiin korotus
/	/	jakolasku

python: lisää löytyy täältä | www |

octave: lisää löytyy täältä | www |

- Vertailuoperaattorit samat

python	octave	
==	==	yhtä suuri
<	<	pienempi
>	>	suurempi
>=	>=	suurempi tai yhtä suuri
<=	<=	pienempi tai yhtä suuri
!=	!=	ei yhtä suuri

python ja octave

- Peruslaskutoimituksissa on eroja
- Kokeillaan **python**:ssa

```
import numpy
a=numpy.arange(2) ; b=2.+a
```

`print(a)` tulostaa `[0 1]`
`print(b)` tulostaa `[2. 3.]`
`print(a*b)` tulostaa `[0. 3.]`
Tulos: Vastaavien alkioiden tulot

- Kokeillaan **octave**:ssa

```
a=[0 1] ; b=2.+a ;
```

`disp(a)` tulostaa `0 1`
`disp(b)` tulostaa `2 3`
`disp(a*b)` tulostaa `error ...`
koska molemmat ovat vaakavektoreita
`disp(a*b')` tulostaa `3`
koska `b'` pystyvektori, saadaan pistetulo
`disp(a.*b)` tulostaa `0 3`
eli vastaavien alkioiden tulot

python ja octave

python ja octave: funktiot

- **python**: Matemaattiset funktiot saadaan käyttöön `import numpy` ⇒ käskymuoto `numpy.cos(3.14)` toimii
- **python**: Vain valitun funktion importointi
  Esim `from numpy import cos`
  ⇒ käskymuoto `cos(3.14)` toimii
- **octave**: Funktiot valmiiksi ladattu
- Tavallisimmat funktiot molemmissa lähes samanlaisia

python	octave	
`cos(x)`	`cos(x)`	$\cos x$
`sin(x)`	`sin(x)`	$\sin x$
`tan(x)`	`tan(x)`	$\tan x$
`power(x,y)`	`realpow(x,y)`	x^y
`sqrt(x)`	`sqrt(x)`	$\sqrt{x}$
`exp(x)`	`exp(x)`	e^x
`log(x)`	`log(x)`	$\ln x$

- Trigonometriset funktiot: $[x]$ = radiaani

python ja octave: funktiot

- Harjoitus
  Laadi **octave** tai **python** ohjelma joka laskee ja tulostaa muuttujat
  $a = \pi$, $b = sin(a)$, $c = cos(a)$, $d = tan(a)$, $e = 1$,
  $f = \mathrm{asin}(e)$, $g = \mathrm{acos}(e)$, $h = \mathrm{atan}(e)$,
  $i = 9$, $j = \sqrt{i}$,
  $k = i^2$ (missä i on imaginaariluku $i^2 = -1$),
  $m = 1 + i$, kompleksiluku,
  $n = 2 + 3i$, kompleksiluku,
  $o = m + n$, $p = |o|$,
  $q = e = 2.71828... =$ Neperin luku,
  $r = \ln(q)$ (luonnollinen logaritmi),
  $s = \log(q)$ (10 kantainen logaritmi),
  $t = -3$, $u = |t|$ ja $v = t^4$
- Ohjelmat monimutkaisempia
  ⇒ Sitä vaikeampi tehdä yksikäsitteinen **python** ja **octave** versio
  ⇒ Useita erilaisia “oikeita” vastauksia

python ja octave

Muuttujan tunnuksista

- Tunnus toimii osoittimena muuttujaan eli muuttujan nimenä
- Yhteen muuttujaan voi osoittaa monta tunnusta
- Tunnuksen voi kesken ohjelman vaihtaa osoittamaan johonkin toiseen muuttujaan
- Toisin kuin monissa kielessä, **python**:ssa tunnus ei ole sidottu mihinkään tiettyyn muuttujatyyppiin
- Kopioidaan kotisivulta **`Pmalli2.py`** ja **`Omalli2.m`** ja vertaillaan niitä

```
# Kommenttirivi: Tama on python ohjelmani Pmalli2.py
import os ; os.system('clear')  # Tyhjennetaan naytto
a = [1,2] ; b = a # Kaksi python kaskya samalla rivilla
print('Vaihe I ..................................')
# Merkilla "\" laaditaan kahden rivin pituinen kasky
print('a[0]=',a[0],',a[1]=',a[1],\
      ',b[0]=',b[0],',b[1]=',b[1])
a[1]=0 # Muutetaan a:n toisen alkion a[1] arvo
print('Vaihe II: b[1] muuttunut!')
print('a[0]=',a[0],',a[1]=',a[1],\
',b[0]=',b[0],',b[1]=',b[1])
# Ratkaisu 1: tuple tyyppista muuttujaa d ei voi muuttaa
c = [1,2]  ; d=tuple(c)
print('Vaihe III ................................')
print('c[0]=',c[0],',c[1]=',c[1],\
',d[0]=',d[0],',d[1]=',d[1])
c[1]=0 # Muutetaan c:n toisen alkion c[1] arvo
print('Vaihe IV: d[1] ei muuttunut!')
print('c[0]=',c[0],',c[1]=',c[1],\
',d[0]=',d[0],',d[1]=',d[1])
# Ratkaisu 2: Kopioidaan e:n kaikki alkiot muuttujaan f
e = [1,2]  ; f=e[:]
print('Vaihe V ..................................')
print('e[0]=',e[0],',e[1]=',e[1],\
',f[0]=',f[0],',f[1]=',f[1])
e[1]=0 # Muutetaan e:n toisen alkion e[1] arvo
print('Vaihe VI: f[1] ei muuttunut!')
print('e[0]=',e[0],',e[1]=',e[1],\
',f[0]=',f[0],',f[1]=',f[1])
```

python ja octave

Muuttujan tunnuksista

- `python Pmalli2.py`

```
Vaihe I ..............................
a[0]= 1 ,a[1]= 2 ,b[0]= 1 ,b[1]= 2
Vaihe II: b[1] muuttunut!
a[0]= 1 ,a[1]= 0 ,b[0]= 1 ,b[1]= 0
Vaihe III ............................
c[0]= 1 ,c[1]= 2 ,d[0]= 1 ,d[1]= 2
Vaihe IV: d[1] ei muuttunut!
c[0]= 1 ,c[1]= 0 ,d[0]= 1 ,d[1]= 2
Vaihe V ..............................
e[0]= 1 ,e[1]= 2 ,f[0]= 1 ,f[1]= 2
Vaihe VI: f[1] ei muuttunut!
e[0]= 1 ,e[1]= 0 ,f[0]= 1 ,f[1]= 2
```

- Tunnukset `a` ja `b` viittasivat samaan muuttujaan. `a`:n alkion `a[1]=1.0` arvon ja tyypin muutos muutti `b[1]`:n arvon ja tyypin
- Muuttujaan `d` kopioitiin `c`:n alkioiden arvot. `d=tuple(c)` teki listan, jonka alkioita ei voi muuttaa. Siksi `c[1]`:n muutos ei muuta alkiota `d[1]='cd'`
- Muuttujaan `f` kopioitiin `e`:n kaikki alkiot `[:]` $\Rightarrow$ `e[1]`:n muutos ei muuta alkiota `f[1]='ef'`

Muuttujan tunnuksista

Vastaava **octave** ohjelma `Omalli2.m` kotisivulta

```
# Kommenttirivi: Tama on octave ohjelmani Omalli2.m
clear ; clc             # Poistetaan ... Tyhjennetaan ...
a = [1,2] ;  b = a ;    # Kaksi octave kaskya samalla rivilla
# Nayttotulostus=printf, Siirry seuraavalle riville=\n
printf("Vaihe I ..................................\n");
# Merkeilla "{" ja "}" laaditaan kahden rivin pituinen kasky
{printf("a(1)=%4.1f,a(2)=%4.1f,b(1)=%4.1f,b(2)=%4.1f\n",
a(1),a(2),b(1),b(2))};
a(1) = 0; # Muutetaan a:n ensimmaisen alkion a(1) arvo
printf("Vaihe II: b(1) ei muuttunut!\n");
{printf("a(1)=%4.1f,a(2)=%4.1f,b(1)=%4.1f,b(2)=%4.1f\n",
a(1),a(2),b(1),b(2))};
```

`octave Omalli2.m` tulostaa

```
Vaihe I ..................................
a(1)= 1.0,a(2)= 2.0,b(1)= 1.0,b(2)= 2.0
Vaihe II: b(1) ei muuttunut!
a(1)= 0.0,a(2)= 2.0,b(1)= 1.0,b(2)= 2.0
```

- **octave**:ssa tätä **python**:n “ongelmaa” ei ole
- Käytännössä tämä tarkoittaa, että kaikki **python** ohjelmat pitää laatia ja tarkastaa huolellisesti

python ja octave

Kommentit (kuva:@www.brainstuck.com)

- Kaikki merkin # jälkeen tuleva lasketaan kommentiksi
- Kommentit ovat tekstiä jonka **python** tai **octave** jättää huomiotta
- Kommentit auttavat muita ymmärtämään koodin toimintaa
- Käytännössä vähemmän käytetyt komennot unohtuvat
- Komennot voi tarkistaa koodin kommenteista
- Kommentit vähentävät tulevaa työtä. Ei tarvitse “keksiä pyörää uudestaan”
- Malliohjelmissa on pyritty selkeään kommentoitiin
- Mikäli edes ohjelma/komento löytyy? `grep disp /home/username/*/*.m` etsii ohjelmat `*.m` joissa `disp` komento