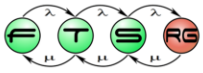
Intelligens rendszerfelügyelet

Scriptelés alapok

Tóth Dániel, Szatmári Zoltán



Budapesti Műszaki és Gazdaságtudományi Egyetem
Méréstechnika és Információs Rendszerek Tanszék

Utolsó módosítás: 2012. február 13.

Tartalom

- Műveletek automatizálása, scriptelés
  - Eltérések az általános programozási nyelvekhez képest
- **Linux alatt Bash**
- Windows PowerShell (következő óra)

Miért éppen Bash?

- Számos elterjedt script nyelv létezik
  - Bash
  - Perl
  - Python
  - Ruby
- Bash
  - Minden gépen elérhető
  - Talán a legegyszerűbb nyelv
  - Legtöbb helyen ezt használják open-source projektekben (gondoljunk a telepítő vagy akár „init” scriptekre)

Motiváció

- Fájlok csoportos átnevezése
- MP3 csoportos átkódolás
- Több fejlesztési projekt együttes fordítása
- Felhasználók csoportos felvétele
- Laborgépek menedzsmentje

Motiváció 2.

- Nem szükséges speciális fejlesztői környezet
- A legtöbb számítógépen van futtatókörnyezet hozzá
- Gyors és hatékony eszköz
- Sok online segédanyag, példa elérhető

Shell scriptelés

- Általában a script nyelvek jellegzetességei
  - Típustalan változókezelés
  - Interpreter futtatja
  - Akár soronként is értelmezhető
  - Minden futási időben értékelődik ki
  - String paramétert is képes értelmezni parancsként
  - Erősen eltér az eddig látott programnyelvektől (C, C++, Java, C#)

Bash

- Bash (Bourne Again Shell)
  - 1987-óta fejlesztik
  - Elődje az alap UNIX-os Bourne shell (sh) 1978-ból
  - Nem mai programozási szemléletmódot követ

- Működési elv (nagyvonalakban):
  1. Soronként elemzi a bemenetet
  2. Azonosítja a saját fenntartott szintaxis elemeit
  3. Stringként mindent behelyettesít, amíg csak lehet
  4. Ami utána marad, azt megpróbálja végrehajtani

Bash

- Alapvető Bash funkciók
  - Automatikus kiegészítés
    - TAB billentyű
  - Parancs előzmények tárolása
    - Fel + Le gombokkal navigálás
    - CTRL+R billentyű kombinációval keresés
  - Terminál gyors bezárása
    - CTRL+D billentyűkombináció

Hello world példa

- Kedvenc editorba írjuk be (joe, mcedit, vi, emacs, kwrite…)

```
#!/bin/bash
#ez egy komment
echo "Hello world"
```

- A fájl végén egy újsort szokás tenni.
- Az első sor kommentje a „shebang”. Egy hint, a `file` nevű programnak jelzi, hogy ez milyen fájl is valójában.
- Adjunk neki futtatási jogot:

```
chmod +x helloworld.sh
```

- Futtassuk (a ./ azért kell, mert az aktuális könyvtár nincs a path-ban)

```
./helloworld.sh
```

DEMO Bash

- Bash alapfunkciók áttekintése
- Fájlok másolása Windows és Linux között
- Hello World példa


```
#!/bin/cat

echo "Hello World";
```

Átirányítások

- Standard I/O, minden programnak
  - 0 – stdin
  - 1 – stdout
  - 2 – stderr
- Átirányítás
  - `cat fájlnév` #fájl→stdout
  - `cat fájlnév 2>&1` #stderr→stdout
  - `cat fájlnév > másikfájl` #fájl→stdout→másikfájl
  - `cat fájlnév 2> másikfájl` #fájl→stdout, stderr→másikfájl
  - `cat fájlnév &> másikfájl` #minden a fájlba ömlesztve

Csővezetékek

- `cat fájl | grep 'x'`
  `#cat stdout-ját a grep stdin-jába`
- Láncolhatóak az alkalmazások… DE…
  - Formázatlan bináris adatátadás történik
  - Gyors, de strukturált adatot nem kezel
  - Strukturált adatot szöveges formába kell alakítani (valamilyen módon), majd a fogadó oldalon sorokra, majd azon belül mezőkre bontva feldolgozni
  - Erre használható programok: cut, awk, sed (tokenizálás, reguláris kifejezések stb.)
  - Erre jó a bash is, pl a `pipecmd | while read VAR` vagy `for VAR in $(pipecmd)` konstrukciókkal
  - Egyszerű adatszerkezeteknél még elmegy…
    - Az emberek itt szokták értékelni a Powershellt ☺

Idézőjelek

- A „dupla” idézőjelek
  - a shell nem értelmezi speciálisan a *, ?, [,], {, }, ;, <, >, stb. karaktereket
  - viszont működik a változókra való hivatkozás és a külső parancs kimenetének behelyettesítése
- Az 'aposztrófok'
  - semmilyen speciális karaktert nem értelmez, még a dollárjelet sem
  - Magát az aposztrófot sehogyan sem tudjuk aposztrófok közé írni, mivel a backslash is elveszíti a szokásos escape-funkcióját

Idézőjelek

- A `backtickek`
  - közé írt parancssor standard outputját a shell behelyettesíti oda, ahol a backtickes kifejezés szerepelt
  - jobb ehelyett a $(zárójeles) alakot használni, mivel az szabadon egymásba ágyazható

Változókezelés

- Környezeti változók
  - Lehetnek beállított/beállítatlan állapotban
  - Értékadással beállítható, akár üres stringre is
  - `set`-tel megnézhetők az éppen beállított változók
  - `unset`-tel visszatér beállítatlan állapotba

Változókezelés

- Típusok
  - Minden változó stringként tárolódik
  - A behelyettesítés helye alapján derül ki, hogy hogyan értelmeződik
  - Magában illetve " " és ` ` környezetben stringként értelmeződik
  - $(( )) környezetben egész számként értelmeződik

Változókezelés

```
#!/bin/bash
VAR1=Ertek #Értékadások
VAR2=4     #Nincs space az értékadás két oldalán!
var3=8
VAR4=3
echo "$VAR1 $VAR2 $var3"; # behelyettesít
echo '$VAR1 $VAR2 $var3'; # nem helyettesít be
echo "${VAR4}. Évfolyam"; # így tudunk közvetlenül
   karakterláncot illeszteni változó értékéhez
echo ${VAR1} ${VAR2}${var3}; #behelyettesít, ez 2 paraméter
   az echo-nak
```

Változókezelés

```
#!/bin/bash
echo ${HELLO}; # deklarálatlan változó, üres stringgel helyettesíti, hacsak a
               # környezetből nem kapott értéket

HELLO=`./helloworld.sh`;    #parancsot futtat, az eredményt behelyettesíti
HELLO=$(./helloworld.sh);   #ugyanaz, mint fent

echo ${HELLO}; #már van értéke.
   #vigyázat a Hello World! string 2 paraméterré válik, ha nem "${HELLO} "-t írunk!
unset HELLO; #már nincs értéke
```

DEMO Változókezelés

- Különböző idézőjelek használata
- Parancs kimenet behelyettesítése


```
#!/bin/bash
VAR1=Ertek
VAR2=4
var3=8
VAR4=3

echo "$VAR1 $VAR2 $var3";
echo '$VAR1 $VAR2 $var3';
echo "${VAR4}. Évfolyam";
echo ${VAR1} ${VAR2}${var3};
```

```
#!/bin/bash

echo ${HELLO}
HELLO=`./HelloWorld.sh`;
echo ${HELLO};
unset HELLO;
echo ${HELLO};
```

Környezeti változók láthatósága

- A környezeti változók alapból nem adódnak tovább a gyerek folyamatnak.
  - Sima script futtatás új „gyerek” bash folyamatot indít!
  - **`export`** – kilistázza az öröklődő környezeti változókat
  - **`export`** `VALTOZONEV` – öröklődővé teszi a változót
- Gyerek folyamatban beállított változók nem adódnak „felfele” a hívónak
  - **source** `script_file` – úgy futtatja a megadott fájl tartalmát, hogy nem indít gyerek folyamatot. Gyorsabban fut, a változók beállítva maradnak.

Tömbök

- Használhatunk tömböket

```
ARRAYVAR=(ERTEK1 Ertek2 "Ertek3");  #értékadás #felsorolással
OTHERARRAY[5]= "Ertek4"; #értékadás konkrét indexre, ritka tömb

echo "elemszám: ${#ARRAYVAR[@]} ";
echo "első elem $ARRAYVAR";

   #for ciklus az összes elemre, figyeljünk a ;-k helyére!
   #Ritka tömbnél így nem működik!
for ((i=0;i<${#ARRAYVAR[@]};i++));
do
   echo "${ARRAYVAR[$i]}";
done;
```

„Mágikus” változók

- Néhány változónak speciális jelentése van
- Bemenő paraméterek
  - `$1, $2` … - script indításakor bemenő paraméterek
  - `$#` - paraméterek száma
  - `$@` - az összes paraméter egy tömbben
- Előző parancs
  - `$?` – legutóbbi futtatott parancs visszatérési értéke
  - `$!` – legutóbbi háttérben indított folyamat PID-je
  - `$$` - éppen futó shell PID-je
- Mezőelválasztó karakter
  - `$IFS` – ennek az értéke alapján automatikusan darabolja a stringeket külön paraméterekké
  - Alapból az IFS értéke minden whitespace `” ” ”\n” ”\t”`

Nevesített paraméterek

- Sok esetben nem célszerű a paraméterek sorrendjét megkötni
- Elegendő névvel hivatkozni rájuk
- Pl.:
  - ./script.sh -a 3 -b 6
  - ./script.sh -b 6 -a 3
  - ./script.sh -h

Nevesített paraméterek

```
while getopts “ha:b:v” OPTION
do
    case $OPTION in
        h)
            printHelp
            ;;
        a)
            A=$OPTARG
            ;;
        b)
            B=$OPTARG
            ;;
        v)
            VERBOSE=1
            ;;
        ?)
            printHelp
            ;;
    esac
done
```

DEMO Nevesített paraméterek

- example.sh -a 4 -b 6
- example.sh -b 4 -a 7 -v

Visszatérési érték

- Minden parancsnak van visszatérési értéke
  - Következtethetünk belőle a lefutás eredményére

```
TEMPERATURE=`./getTemperature`;
if [ $TEMPERATURE -gt 30 ]; then
    echo "Hot";
    exit 2;
elif [ $TEMPERATURE -gt 25 ]; then
    echo "Warm";
    exit 1;
else
    echo "Ok";
    exit 0;
fi
```

DEMO Visszatérési érték

- Weboldal másoló script
  - wget parancs használata
  - man használata


```
#!/bin/bash

wget -O - -q -t 1 http://inf.mit.bme.hu
RETURNCODE=$?;
if [ $RETURNCODE -eq 5 ]; then
                echo "Baj van. Hibakód: $RETURNCODE";
else
                echo "Ok";
fi
```

Fájlnév kiegészítés

- Szokásos fájlnév-minta kiegészítő karakterek
  - * - tetszőleges string
  - ? – egy karakter
- Egyik megoldás:

```
SHFILES=*.sh; # behelyettesítődik fájlnevekre
echo $SHFILES; # egy stringben összes fájl neve
for a in $SHFILES; do
 # vigyázat, ha fájlnévben $IFS karakter van,
 # akkor ott elválasztja!
   echo "$a";
done;
```

- Másik megoldás (újsor karakternél vág, nem pedig $IFS-nél):

```
ls *.sh | while read line; do
   echo "$line";
done;
```

String darabolás

```
VAR='foo:bar:cuc:mak'

# darabolás IFS-sel
OLDIFS=$IFS;  #sosem árt elmenteni
IFS=':';
for a in $VAR; do echo "$a"; done
IFS=$OLDIFS    # ne felejtsük el visszaállítani

# darabolás mintaillesztéssel
# levágja amire illeszkedik a minta
echo ${VAR#*:};    # legnagyobb postfixet tartja meg :bar:cuc:mak
echo ${VAR##*:};   # legkisebb postfixet tartja meg :mak
echo ${VAR%:*};    # legnagyobb prefixet tartja meg foo:bar:cuc:
echo ${VAR%%:*};   # legkisebb prefixet tartja meg foo:
```

- Bonyolultabb esetekre: `cut`, `awk`, `sed`

DEMO Fájlnév kiegészítés

- Header fájlok vizsgálata
  - `#define` sorok kigyűjtése
- $IFS elválasztó karakter használata
  - /etc/passwd fájl feldolgozása
  - Azon felhasználók adatainak kigyűjtése, melyek `/bin/sh` login shell-t használnak.


```
#!/bin/bash

HEADERS=Files/*.h
echo $HEADERS;
for a in $HEADERS; do
                    echo $a;
                    cat $a | grep "#define";
done;
```

```
#!/bin/bash

OLDIFS=$IFS
IFS=":";

cat /etc/passwd | grep "/bin/sh" | while read line; do
                    for a in $line; do
                                        echo $a;
                    done;
done;
IFS=$OLDIFS;
```

Elágazás

- Példa:

```
NUM1=$1;
NUM2=$2;
if [ $NUM1 -eq $NUM2 ]; then
    echo "$NUM1 egyenlő $NUM2";
elif [ $NUM1 -gt $NUM2 ]; then
    echo "$NUM1 nagyobb, mint $NUM2";
else
    echo "$NUM1 kisebb, mint $NUM2";
fi
```

- A [a test program neve (igen ez egy program neve ☺)
  - Lásd: `man test` a lehetséges paraméterezésre
  - Figyeljünk a space-ek megfelelő helyére!
  - Visszatérési érték: **0 - igaz**, **egyébként hamis**, ez a shell scripteknél pont fordítva van, mint a „rendes” programnyelveknél

Üres string vizsgálat

- Hogyan vizsgáljuk meg, hogy a változó üres stringet tartalmaz-e?
  - [-z "$1"]
    - helyes
  - ["$1" = ""]
    - helyes
  - [x$1 = x]
    - Hibás megoldás, mert a változó tartalmazhat szóköz karaktereket és behelyettesítés után összezavarja a shell-t

C-jellegű aritmetika

- Vannak C-jellegű szintaktikus segítségek. Pl.:
  - Inkrementálás

```
VAR=1
((VAR++))
echo $VAR  # 2-t ír ki
```

  - Ciklus

```
for ((i=0;i<10;i++)) {
        echo $i
}
```

Alapértelmezések használata

- Hogyan adjunk alapértelmezett értéket változónak?
  - Rossz példa:

```
VAR=defaulteretek
VAR=$1
```

  - Jó példa:

```
if [ -z "$1" ]; then
        FOO=defaultérték
else
        FOO="$1"
fi
```

  - Egyszerűbben:

```
[ -z "$1" ] && FOO=defaultérték || FOO="$1”
```

Konfigurálható script

- Ha a bemenet eleve így van megadva:

```
BEALLITAS=ertek1
OPCIO=ertek2
LISTA=(elem1 elem2 elem3)
```

- A bash maga parse-olja nekünk, és utána mint változókat használhatjuk:

```
source ./inputfile
echo $BEALLITAS; #ertek1 lesz a kimenet
```

- De vigyázat, mindent végrehajt, ami a bemenő fájlban van! Nagy biztonsági kockázat.

Aritmetikai műveletek

- A script nyelv nem támogatja az elemi aritmetikát
- Létezik az `expr` Linux utasítás, melynek segítségével ilyen műveleteket végezhetünk
  - Bemeneti paramétereket értelmezi: az operandusok és az operátor is egy-egy külön paraméter
  - A kimenetre írja ki a végeredményt
  - Praktikusan backtickek között alkalmazzuk
- Példa:

```
RESULT=`expr 3 + 5`;
echo $RESULT          # Kiirja, hogy 8
```

Reguláris kifejezések

- Sok helyen használhatjuk őket
  - Pl. sed, awk, grep
  - (Perl, Java, C#...)
  - Egyszerű string manipulációt nagyon megkönnyíti
- Példa kinek a nevét írtuk rosszul

	A	B	C
1	**Személy**	**Kedvenc étel**	**mennyiség**
2	Don Mascarpone	Tiramisu torta	3 szelet
3	Vito Mascarpone	Bolognai spagetti	2 tányér
4	Kicsi Angelo	Gelato fagylalt	5 gombóc
5	Nagy Luzio	Gelato fagylalt	2 gombóc
6	Federico mortellini	mogyoró	nagy zsák

Reguláris kifejezések

- Megoldás:
  - Exportáljuk CSV-be a táblázatot, így fog kinézni:

```
"Személy","Kedvenc étel","mennyiség"
"Don Mascarpone","Tiramisu torta","3 szelet"
"Vito Mascarpone","Bolognai spagetti","2 tányér"
"Kicsi Angelo","Gelato fagylalt","5 gombóc"
"Nagy Luzio","Gelato fagylalt","2 gombóc"
"Federico mortellini","mogyoró","nagy zsák"
```

Reguláris kifejezések

- Egy lehetséges megoldás:

```
cat temp/csvdemo.csv |
cut –delimiter=',' -f 1 |
grep -v "[A-Z][a-z]* [A-Z][a-z]*"
```

- Eredmény:

```
"Személy"
"Federico mortellini"
```

Reguláris kifejezések

- SED == Stream EDitor
  - Alapvetően az stdinről olvasott szöveg-streamen végez programozható átalakításokat, és az eredményt az stdoutra írja.
  - Egyszerre valósítja meg többek között a *cut*, a *grep*, a *tr*, a *head* és a *tail* parancsot.
  - Write-only programozás
    - Példa: Hanoi tornyai

```
s~^xx*$~:n:3:2:1:&:~;tB;d;:B;/^:$/d;h
s~^:.:\(.\):.:\(.\):*:.*~\2 --> \1~;x
/^:y:.:.:.:*:.*/b0;/^:n:.:.:.:x:.*/b1
s~:n:\(.\):\(.\):\(.:x*\)x:\(.*\)~:n:\2:\1:\3:y:\1:\2
:\3x:\4~
bB;:1;x;p;x;s~^:n:.:.:.:x:\(.*\)~:\1~;bB;:0;x;p;x
s~^:y:\(.\):\(.\):\(.\):x\(x*:*\)~:n:\1:\3:\2:\4~ bB
```

DEMO SED

- Hanoi tornyai megoldása SED segítségével
- Kutya – macska karakterlánc csere

cat kutya.txt | sed "s/kutya/macska/g"

Tanácsok, hibakeresés

- Legyen komment a script elején
  - Ki írta, mire való, hogy kell paraméterezni
- A bemenő paramétereket ellenőrizzük
  - Dobjunk hibaüzenetet, ha helytelen a paraméterezés
- A script NE töröljön vagy írjon felül olyan fájlokat, amire nem kértük
  - ☠
  - Ideiglenes fájlokhoz használjuk az `mktemp`-et

Shell opciók

- Hibakereséshez hasznos: `bash -x`
  - A behelyettesítések utáni parancsot kiírja
  - Script futása közben követhető minden művelet, értékadás stb.
- Változónév elírások ellen: set -u
  - Ne helyettesítsen be üres stringgel olyan változókat, amik nincsenek definiálva, helyette dobjon hibát
- Parancs meghiúsulás ellen: set -e
  - Ha egy parancs sikertelenül tér vissza, akkor a script lépjen ki és nem folytatódjon.
  - Pl:

```
cd "$1";
rm -rf *.csv
```

Feladatmegoldás

Készítsen egy Bash scriptet, ami fogad egy felhasználó és hozzárendelt könyvtár listát CSV formátumban, létrehozza a felhasználókat és a könyvtárakat és beállítja a jogosultságokat úgy, hogy minden felhasználó be tudjon lépni, olvasni és írni is tudjon a hozzárendelt összes könyvtárban, de ne tudjon belépni egyéb könyvtárakba, amikhez nem volt hozzárendelve. Egy felhasználó több könyvtárhoz és is lehet rendelve és egy könyvtárhoz is több felhasználó lehet rendelve. Posix ACL-eket nem használhat, viszont szükség esetén létrehozhat új csoportokat. Ha a rendszeren meglévő felhasználót talál, azt ne módosítsa, hagyja ki teljesen! Feltételezhet angol locale beállítást. A bemenetet a következő formátumban kapja meg:

```
konyvtar1:usernev1
konyvtar1:usernev2
konyvtar2:usernev2
```

Megoldás felépítés

- **Fejkomment**
- **Paraméterek ellenőrzése**
- Bemenetből a felhasználók és könyvtárak kigyűjtése
- Még nem létező felhasználók létrehozása
- Még nem létező könyvtárak létrehozása
- Csoportok létrehozása az egyes könyvtárakhoz
- Jogok beállítása

Feladatmegoldás

```
#!/bin/bash
# Name:           irfhf_1C_1.sh
# Author:         Mekk Elek
# Date:           2011.02.22.
# Desc:           Felhasznalok felvetelet vegzo script
# Param:
#   $1 - konyvtar-felhasznalonev parokat tartalmazo csv fajl
#   $2 - egy konyvtareleresi ut, mely megadja, hogy hol kivanjuk elkesziteni
#        a csv fajlban levo konytarakat

actuser=`id -nu`;
if [ $# -ne 2 ]; then
   echo 'A bemeno parameter egy fajlnev es egy konyvtar eleresi ut kell legyen!';
elif [ ! -f $1 ]; then
   echo 'Nem letezik a paramatereben megadott fajl!';
elif [ ! -d $2 ]; then
   echo 'Nem letezik a paramatereben megadott konyvtar!';
elif [ "$actuser" != "root" ]; then
   echo 'A scriptet csak root felhasznalokent lehet futattni!';
else …….
```

Megoldás felépítés

- Fejkomment
- Paraméterek ellenőrzése
- **Bemenetből a felhasználók és könyvtárak kigyűjtése**
- Még nem létező felhasználók létrehozása
- Még nem létező könyvtárak létrehozása
- Csoportok létrehozása az egyes könyvtárakhoz
- Jogok beállítása

Feladatmegoldás

```
# vegigmegyunk a fajl minden soran, ha olyan felhasznalot talaltunk,
# ami meg nincs a rendszerben, eltaroljuk egy tombben
ROWS=`cat $1`;

i=0;    # ciklusvaltozo

for a in $ROWS; do
        USERS[$i]=${a#*:};      # Felhasznalo neve
        DIRS[$i]=${a%:*};       # Letrehozando konyvtar
        i=$(($i+1));
done;
```

Megoldás felépítés

- Fejkomment
- Paraméterek ellenőrzése
- Bemenetből a felhasználók és könyvtárak kigyűjtése
- **Még nem létező felhasználók létrehozása**
- **Még nem létező könyvtárak létrehozása**
- Csoportok létrehozása az egyes könyvtárakhoz
- Jogok beállítása

Feladatmegoldás

```
for ((i=0;i<${#USERS[@]};i++));
do
   # ellenorizzuk, hogy mar van-e ilyen felhasznalo a rendszerben
   username=${USERS[$i]};     # letrehozando felhasznalo neve
   t=`grep $username /etc/passwd | grep -c ':'`;
   if [ $t -eq 0 ]; then      # ha meg nincs, akkor letre kell hozni
        useradd $username;    # felhasznalo letrehozasa
   fi

   # letrehozando konyvtar eleresi utjanak elkeszitese
   lentmp=`expr length $2`    # celkonyvtar eleresi utjanak hossza
   pertmp=`expr substr $2 $lentmp 1` # eleresi ut utolso karaktere
   if [ "$pertmp" !=  "/" ]; then    # ha NEM volt
        dirpath=$2/${DIRS[$i]};
   else
        dirpath=$2${DIRS[$i]};       # ha VOLT perjel a parameterben
   fi
```

Megoldás felépítés

- Fejkomment
- Paraméterek ellenőrzése
- Bemenetből a felhasználók és könyvtárak kigyűjtése
- Még nem létező felhasználók létrehozása
- Még nem létező könyvtárak létrehozása
- **Csoportok létrehozása az egyes könyvtárakhoz**
- **Jogok beállítása**

Feladatmegoldás

```
    grpname="irfhf2_1grp_${DIRS[$i]}"; # konyvtarnevbol
    generalt csoport
    # megnezzuk, hogy letezik-e mar a konyvtar
    if [ ! -d $dirpath ]; then     # ha meg nincs ilyen konyvtar
        mkdir $dirpath;            # konyvtar letrehozasa
    fi
    groupadd $grpname;
    chgrp -R $grpname $dirpath;
    chmod a-rwx $dirpath;   # minden jog megvonasa
    chmod g+rwx $dirpath;   # csoportbelieknek minden jog

    # felhasznalo felvetele a konyvtarhoz tartozo csoportba
    usermod -a -G $grpname $username;

done;
```

További info

- LinuxConfig: „Bash scripting Tutorial”, http://www.linuxconfig.org/Bash_scripting_Tutorial
- „Advanced Bash-Scripting Guide”, http://tldp.org/LDP/abs/html
- A Unix operációs rendszer: http://www.hit.bme.hu/~szandi/unix/index.html
- man bash, man sed, man cut, man sort, man grep… ☺