6_complex_feladatok

October 26, 2016

1 Haladó bash, gawk

Jegyzetben az 8-9.-10. fejezet (97-től 125.-ig oldalig). http://stegerjozsef.web.elte.hu/teaching/szamalap.pdf

Egyéb források:

Linux parancsok: http://www.letix.hu/

Bash programozás: http://www.tldp.org/pub/Linux/docs/HOWTO/translations/hu/Bash-Prog-Intro-HOWTO-hu.txt

1.1 Segítség a tanuláshoz:

Bíró Gábor készített egy összefoglaló LaTeX fájlt, ami minta lehet akár a szakdolgozatotokhoz is. Minden szükésges parancs benne van, amire a BSc alatt valószínűleg szükségetek lesz.

http://itl7.elte.hu/~iracz/Oktatas/SzA/latex.zip

Illetve mindenki hozzáférhet a "caesar.elte.hu" géphez, ahol kipróbálhatja a linux parancsokat. Infó: https://ugykezelo.elte.hu/

1.2 Ablakos szövegszerkesztők

A linuxban több féle ablakos szövegszerkesztő van alapértelmezésben telepítve. Az egyik legelterjettebb a **`gedit`** nevű program.

Mindenki indítsa el a gedit nevű programot: - Menü -> Accessories -> gedit - Kiadjuk a terminélban: **`gedit`**

Hasonlóan működik, mint a windowsban a jegyzettömb.

1.3 Párhuzamos programkezelés

A linuxban lehetőség van a programok párhuzamos kezelésére, akár 1 terminál ablakon belül is. Az épp futó alkalmazást háttérbe lehet küldeni, és az így visszakapott promt után más parancsok is kiadhatók.

Mindenki indítsa a gnuplotot: **`gnuplot`**

Ezután adjuk ki a **`CRTL+Z`** billenytű kombinációval felfügesztés parancsát, majd **`bg`** beütésével háttérbe küldhető a prgram.

Végül indítsuk el a **`top`**-ot!

A következő dia előtt ismét adjuk ki **`CTRL+Z`** majd **`bg`** utasításokat.

1.4 Futó programok kezelése

A **jobs** paranccsal ellenőrizhető, milyen futó programok vannak.
Minden program melett van egy szám (ID), amivel hivatkozható a program:
Program kilövése: **kill %ID** paranccsal

```
kill %2
```

Program előrehozása: **fg %ID** paranccsal

```
fg %1
```

1.5 Program háttérben története indítása

Egy programot terminálból úgy is el lehet indítani, hogy a promt megmaradjon, azaz a program háttérben indul el (olyan, mintha **CTRL+z** és **gb** utasítást adtunk volna ki. Ezt a program neve után írt **&** karakterrel lehet elérni.

Példa:

```
evince latex.pdf &
```

1.6 Különleges karakterek bash-ban

- **'** : Megakadályozza a héjat (shell), hogy a `$`-t a változók jelzésének tekintse
- **"**"** : Az idézőjelek arra valók, hogy egyes karakterek vagy lefoglalt szavak speciális jellegét feloldják.
- **`** : Operátor, mely a két ` ` között lévő jelsort végrehajtja, és a kimenetét adja vissza úgy, hogy az új sor karakter helyére szóközt rak.

1.7 Különleges karakterek bash-ban 2

- \ : Bourne Again Shell escape-karaktere. Többek között ezzel a jellel lehet elérni, hogy az őt követő speciális karakter elveszítse módosító hatását.
- **$** : Segítségével változó neve és értéke között tudunk különbséget tenni. Ha `$valtozo`-t írunk, az a változónk értékét fogja visszaadni.

1.8 Különleges karakterek bash-ban 3 ("Joker karakterek")

- **?** : 1 darab "akarámit" helyettesít

  ```
  "sinusa?atok.dat" : ez lehet "sinusadatok.dat"
  ```

- ***** : Akármilyen hosszú szövegrészt helyettesít

  ```
  "sin*.dat" : Minden fájl ami ".dat"-ra végződik, és "sin"-nel ketdődik
  ```

1.9 Fájlok keresése:

Fájlokat a **find** paranccsal lehet keresni.

```
find ~ -name "*.jpg" # az összes kép keresése a home-ban

find ~ -iname "*.DAT" # kis és nagybetű különbség nincs!

find / -perm 777
#777-es joggal rendelkező állomány keresése a gyökrében

find ~/latex -size +1024M
#1024 Mb-nál nagyobb fájlok keresése a /latex mappában
```

1.10 Szöveg keresése bash-ban

Szöveget keresni a **grep** parancs segítségével lehet. **Alapértelmezésben nem azonos a kis/Nagy betű.**

```
grep equation ./latex/latex.tex # "equation" szó keresése a latex.tex-ben
grep equation * # "equation" szó keresése minden fájlban a mappán belül
grep -i EquAtion * # Kis/Nagy betű nem számít
grep -n equation * # kiírja a sor számát
grep -l equation * # Fájlnevet ad vissza
```

1.11 Kimenet átirányítása

Lehetőség van a kimenetet a terminál ablak heylett pl. fájlba irányítani:

```
ls -l > file_lista.txt

# Létrejött a file_lista.txt (ha létezett már a fájl, akkor ezzel felülírodott)
cat file_lista.txt
```

Megjegyzés: Fájlból bemenetet csinálni "**<**" tud.

1.12 Hozzáírás fájlhoz:

Hozzáírni a **>>** paranccsal lehet egy meglevő fájlhoz:

```
echo "Kiscica,ezt hozzáírom a fájlhoz"

echo "Kiscica,ezt hozzáírom a fájlhoz" >> file_lista.txt

tail file_lista.txt
```

1.13 Több parancsa kiadása

A "&&" vagy ";"jellel lehet több parancsot kiadni egy sorban. **;** utáni parancs mindenképp lefut, a **&&** pedig csak ha minden rendben volt.

```
ls -al && cp valami.tex ize.tex && cd /latex
```

1.14 Parancs kimentének feldolgozása

A | jellel át lehet adni egy parancs kimentét egy másik parancsnak: #### Példa:

```
ls | sort -r
```

1.15 Hasznos parancsok1 (otthonra):

- **passwd**: Jelszó csere. Ekkor kéri a régi jelszót, majd 2-szer az újat a rendszer! Ezeken a gépeken nem használható!
- **file akarmi.tex** : megvizsgálja az akarmi.tex típusát
- **uniq** : Több sorból álló szövegben az ismétlődő sorokkal kezd valamit
- **tree** : kiírja a könyvtárstruktúrát
- **du** : Az aktuális könyvtár fájljainak méretét adja meg
- **df** : Szabad terület számítása, partíciónként

1.16 Hasznos parancsok2 (otthonra):

- **touch file** : létrehoz egy "file nevű" üres állományt
- **cut** : Bement (stdin), vagy paraméterként megadott fájl minden sorának egy megadott részét vágja ki
- **tr** : karakterek lecserélése, változtatása adott karaktersorban
- **echo vegyes | tr a-z A-Z** : a vegyes szóban a kis betűket nagyra cseréli.
- **echo egyesek | tr -d e** : az egyesek szóból kitörli az e betűket

1.17 Hasznos parancsok3 (otthonra):

- **diff -u file1 file2 > eredmeny** : Összehasonlítja a fájlok tartalmát, a különbséget pedíg az "eredmeny"-be írja
- **tar** : Ki-be tömörítő
- **tar -cvfz file.tar.gz /eleresi/ut/amit/mentunk** : az elérési útvonal alatt levő adat gzip-be tömörítése.
- **tar -xvfz file.tar.gz**: Kitömöríti a gz-t, majd a tar-t, és egy /file könyvtárba teszi

1.18 Hasznos parancsok4 (otthonra):

- **login** : Bejelentkezés
- **logout** : Kijelentkezés
- **who** : Bejelentkezett felhasználók kiíratása
- **ps** : Futó folyamatok kiírása
- **killall command** : az összes fehasználó (amihez van jogunk) által futtatott "command" nevű folyamat leállítása
- **shutdown -P now** : Kikapcsolás (-P=Poweroff), azonnal (percben lehet megadni időt)

1.19 Bash programozás

A parancsértelmező által ismert parancsok fűzhetőek össze egy "programba" vagy más néven shell script-be. (lásd .bat) - A program első sorában célszerű minden esetben meg kell adni a futtató programot (shell-t), jelen esetben a bash-t. Tehát a fájl kezdete legyen bash esetén: **`#!/bin/bash`** - Más futtató program (parancsértelmező) is megadható a program első sorában (például python). - Futtatási jog nélkül nem mindig futtatható, ezért célszerű kiadni: **`chmod +x programnev`**

- Program futtatásához a parancs(ok)

```
bash programnev.sh
```

```
(./programnev.sh)
```

Más interpreterekhez:

```
python akarmi.py #python script
```

```
octave valami.m #octave script
```

1.19.1 Példák:

Hozzunk létre egy **hello.sh** fájlt, és írjuk bele:

```
gedit hello.sh &
```

```
#!/bin/bash
echo Hello World!
```

Mentés után futtassuk le:

```
(chmod u+x hello.sh)
bash ./hello.sh
```

1.20 Hasznos dolgok a szrikptekhez:

- Változók: Lehetséges változókat eltárolni és később hivatkozni rájuk: `$izé` az izé nevű változó.

```
szam=20
echo $szam
```

- Egész számok generálása a **`seq`** paranccsal lehetségel (pl. 10-től 100-ig)

```
seq ELSŐ NÖVEKMÉNY UTOLSÓ
seq 10 2 100 # 10-től kettesével 100-ig megy
seq 100 -5 10 # 100-tól 10-ig 5-ösével
```

1.21 Bash mint számológép

Próbáljuk ki:

```
a=2
b=5
echo $a*$b
```

- A **let** parancs: Ez a parancs képes aritmetikai műveleteket végezni: (“b++” jelentése, a b-t 1-gyel megnöveli)

```
a=1234; let "a=a+1"; echo "$a"
a=2; b=5; let c=$a*$b; echo $c
let 'b = a' "(a += 3) + $((a = 1)), b++"
```

1.21.1 Egyéb példák:

A következő 3 példához használjuk a **test.sh** -t állomány:

```
    gedit test.sh &
```

```
chmod u+x test.sh
```

A következő 3 példát másoljuk bele (külön-külön), mentsük el a fájlt, majd teszteljük le szkripteket a következő parancs kiadásával:

```
bash test.sh
```

- **While** szerkezet: Ez egy olyan ciklus ami először megvizs egy feltételt, majd végrehajta, ha igaz

```
#!/bin/bash
    COUNTER=0
    while [ $COUNTER -lt 10 ]; do
        echo a szamlalo erteke: $COUNTER
        let COUNTER=COUNTER+1
    done
```

COUNTER változót 1-el növeli addíg, amíg el nem éri a 10-et.

- **Until** szerkezet:

```
#!/bin/bash
    COUNTER=20
    until [ $COUNTER -lt 10 ]; do
        echo a szamlalo erteke: $COUNTER
        let COUNTER-=1 # COUNTER-=1 jelentése: COUNTER=COUNTER-1
    done
```

COUNTER változót 20-as kezdőértékről 1-el csökkenti, amíg el nem éri a 10-et.

- **For** ciklus (ez egy olyan ciklus, ami addig fut, amíg igaz az állítás) Ezt használjuk általában mindig.

Szintatktika: **for "Változó megadás" "Állítás" "Utasítás"**

```
#!/bin/bash
    for i in `seq 1 10`;do
    echo $i
    done
```

Az **i** változó felveszi 1-től 10-ig az értékeket, majd ki is írja a képernyőre

1.21.2 A prokramok írhatóak 1 folytonos sorba is, ahol az eddigi új sort ";" vagy "&&" jel váltja fel:

- while szerkezet:

```
SZAM=0; while [ $SZAM -lt 10 ]; do echo érték: $SZAM ;let SZAM=SZAM+1;done
```

- until szekezet:

```
COUNT=20; until [ $COUNT -lt 10 ]; do echo érték:$COUNT; let COUNT-=1;done
```

1.21.3 Példák for ciklusra

- for ciklus:

```
for i in `seq 1 10`;do echo $i; done
for i in `seq 2 5 100`;do echo $i; done
for i in $(ls);do echo fájlok: $i; done
```

2 Gawk

Jegyzetben az 4. fejezet (53-től 67.-ig oldalig). http://stegerjozsef.web.elte.hu/teaching/szamalap.pdf

Az **awk** (vagy **gawk**, azaz GnuAWK)egy olyan programozási nyelv, amit szöveges állományok feldolgozására terveztek. Egy tipikus awk programot (szkript) egy interpreter olvas be és hajt végre. A szkript végrehajtása során a feldolgozott szöveges állomány(ok)at (tartalmának változatlanul hagyásával) másféle kimenetté formálja át. A bemeneti adatok változatlanul hagyása biztosítja, hogy ugyanazon adatokra (például szövegfájlok) többféle awk program is futtatható egymás után, vagyis ugyanazon fájl(ok)ból többféle adat is kinyerhető legyen.

Töltsük le a két fájlt:
wget itl7.elte.hu/~iracz/Oktatas/SzA/millenium.dat
wget itl7.elte.hu/~iracz/Oktatas/SzA/millenium_new.dat
Nézzük is meg őket:

```
more millenium.dat
more millenium_new.dat
```

Mi az eltérés köztük?

2.0.4 Írassuk ki a két fájlt gawk-val!

“‘bash awk millenium.dat

```
>```bash
awk millenium_new.dat
```

Mi a hiba?

2.0.5 A "print" parancs

A **print** paranccsal lehet kiíratni az awk-kal bármit.

Példa: (Csak az első oszlop kiíratása)

```
awk '{print $1}' millenium.dat
```

```
awk '{print $1}' millenium_new.dat
```

Most mi a hiba?

2.0.6 Elő és utófeldolgozók

A **BEGIN** előfeldolgozó a lényegi programot megelőzően fut le, itt lehet megadni például a bemeneti fájlt szerkezetét (pl. mi az oszlopelválasztó karakter). Az **END** utófeldolgozó a léynegi program után fut le. Itt adható meg, hogy pl. mentse el fájlban adott stílusban az eredményt a szkript.

2.0.7 Különleges vezérlő karakterek

- RS: Record separator (Sorok elválasztója)
- FS: Field separator (Oszlopok elválasztója)
- NR: Number of record (Sorok száma)
- NF: Number of field (oszlopok száma)

2.0.8 Speciális helyettesítő karakterek:

- `[a-z]` a,b,c,...z
- `[0-9]` 0,1,2,3,...9
- `[:alnum:]` betű vagy szám
- `[:alpha:]` betű
- `[:digit:]` szám

Példa:

```
awk 'BEGIN{FS=";"}{print $1}' millenium_new.dat
```

```
awk 'BEGIN{FS=";"}{print $1,$2*$3,$8}' millenium_new.dat
```

Példa programok összekapcsolására:

```
awk '{print $8,$3,$4,$5}' millenium.dat | sort > adat.dat

more adat.dat

gnuplot

splot "adat.dat" u 2:3:4 lw 5
plot "adat.dat" u 1:2 w p ps 7
```