

# Linux, Python és PowerShell alapok

*Gyakorlati útmutató*

*Készítette: Micskei Zoltán, Szatmári Zoltán, Horányi Gergő, Nádudvari Tamás*

*Utolsó módosítás: 2015.02.27.*

A gyakorlat célja, hogy bemutassa azokat az alapvető technológiákat, amik szükségesek a szkripteléses házi feladatok megoldásához. Mivel a gyakorlat ideje véges, ezért itt nyilván csak a legfontosabbakra tudunk kitérni, az előadásokat és a dokumentum végén megadott további anyagokat is érdemes még megnézni a két környezet megfelelő szintű elsajátításához.

**FIGYELEM:** az utasításokat, szkripteket ne másoljuk, hanem tényleg gépeljük is be. Különböznél nem sok mindent tanulunk belőle, nem rögzül a szintaktika.

## 1 Linux és Python

A feladatokat egy VMware virtuális gépbe telepített Fedora rendszeren fogjuk végrehajtani. Ez a virtuális gép előre telepítve tartalmazza a Python parancsértelmezőt és néhány egyszerűbb szövegszerkesztő alkalmazást a szkriptek létrehozásához.

### 1.1 Linux alapok

Az első feladatban áttekintjük a Linux rendszerek használatának alapjait.

#### 1. Indítsuk el a virtuális gépet!

Lépjünk be a virtuális gépre:

- Ha gyakorlaton vagyunk, akkor az ott megadott felhasználóval.
- Ha VCL-ben foglaltunk virtuális gépet, akkor az ott kapott felhasználóval.
- Ha letöltöttük a gépet, akkor a README fájlban megadott felhasználóval.

A VCL-ben csak parancssoros környezet érhető el, a többi esetben elindíthatjuk az Xfce grafikus felületet:

```
startxfce4
```

Az elindulás után indítsunk el egy Terminal ablakot.

#### 2. Prompt

A terminálban látható prompt leggyakrabban az aktuális felhasználó és a számítógép nevét jeleníti meg @ karakterrel elválasztva.

```
meres@linux-vm>
```

A fenti prompt tehát azt jelöli, hogy a linux-vm nevű gépre a meres nevű felhasználóval vagyunk bejelentkezve. (A prompt pontos kinézete ettől eltérhet.)

### 3. A kapott terminálnál próbáljuk ki az alapvető funkciókat:

**FIGYELEM:** törekedjünk arra, hogy használjuk a TAB billentyűvel az automatikus parancs- és fájlnev kiegészítést. Vegyük észre, hogy amennyiben több lehetőség adódik a kiegészítésre, akkor a TAB kétszeri lenyomása után ezekről listát kapunk.

**TIPP:** A paramétereknél is érdemes a TAB billentyűt használni, mert különböző típusú elemek kiegészítését is tudja a Bash shell.

Alap fájlkezelő parancsok: cd, ls, cp, mkdir, pwd

```
pwd                # print working directory: aktualis könyvtar
cd ~               # ~: a felhasználónk home könyvtára
mkdir test         # make directory: test könyvtar létrehozása
cd test           # change directory: könyvtar váltása
touch 01.txt       # fájl hozzáférési idők módosítása
echo 02 > 02.txt  # kiírás és fájlba átíratás
ls                # könyvtar tartalmának listázása
ls -l             # részletes lista
cp 02.txt 03.txt  # copy: másolás
cat 02.txt        # fájl tartalmának összefűzése és kiírása
clear             # képernyő törlése
chmod a+w 02.txt  # jogosultságok allítása
```

### 4. Súgó oldalak

A súgó oldalak előhozására való a man (manual) parancs. Nézzük meg a fenti parancsok közül néhányat a manual oldalát, pl.:

```
man touch
```

A manualból a q billentyűvel tudunk kilépni.

További információt kaphatunk az info parancs segítségével:

```
info touch
```

### 5. Felhasználóváltás, privilegizált műveletek

A whoami parancs segítségével tudjuk ellenőrizni, hogy éppen milyen felhasználóval vagyunk bejelentkezve.

```
whoami
```

Sok művelet végrehajtása a sima felhasználóknak nem engedélyezett, ehhez root (superuser) jogok kellenek.

```
cat /etc/shadow          # permission denied az eredmény
```

Az egyik módszer ekkor, hogy ha van megfelelő jogunk (a /etc/sudoers fájlban lévő beállítások határozzák ezt meg), akkor ezt az egy műveletet megpróbáljuk a root felhasználóként végrehajtani, erre való a sudo parancs. A sudo paraméterként egy utasítást vár, az elindítása után pedig a SAJÁT felhasználónk jelszavát kéri be

```
sudo cat /etc/shadow          # saját jelszót kell utána megadni
```

Ha sikeres volt, akkor ezután még 5 percig ezt megjegyzi a rendszer, és nem kell a későbbi sudo utasítások végrehajtásához jelszót megadni. (A sudo ennél sokkal többet is tud, pl. -u segítségével tetszőleges felhasználó nevében parancsot végrehajtani, de ennyi most elég lesz nekünk egyelőre.)

A másik lehetőség, ha teljesen átváltunk más felhasználóra, erre való a su parancs. Ilyenkor annak a felhasználónak a jelszavát kell beírni, akire átváltunk. Ha nem adunk meg paraméterként felhasználónevet, akkor alapértelmezetten a root felhasználóra akar átváltani.

```
su -
```

A - hatására a kapott shell login shell lesz, azaz például megkapjuk a cél felhasználó PATH beállításait.

Ezután lépünk is ki a root shellből, mert most nincs szükség rá (biztonsági okokból érdemes csak akkor átváltani, ha erre tényleg szükség van). Erre az exit parancs szolgál:

```
exit
```

## 6. Billentyűzetkiosztás változtatása

Ez eléggé disztribúciófüggő, a gyakorlaton használt Fedora esetén a következő parancs segítségével lehet például angol (amerikai) kiosztást kérni:

```
localectl set-keymap us
```

A továbbiakban mindenki olyan kiosztást használjon, ami neki jobban kézre áll (us – amerikai, hu – magyar, stb.).

## 7. Hálózati beállítások

A legfontosabb hálózati beállításokat az ip paranccsal kérdezhetjük le.

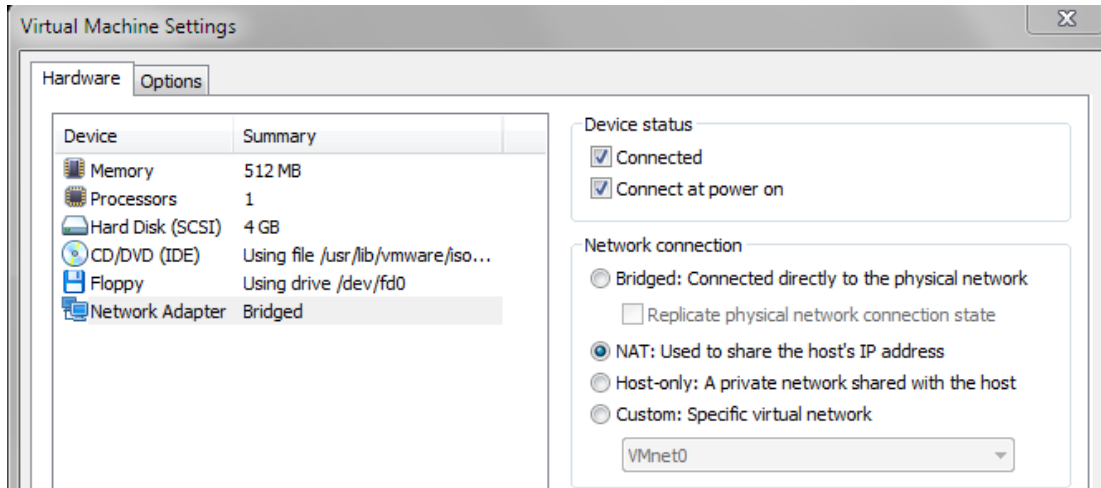
```
ip addr show
```

Keressük meg a kimenetben az IP-címünket. Ha a virtuális gép hálózati kártyája *Bridged* üzemmódban van, akkor ugye úgy viselkedik, mintha a fizikai hálózatunkra lenne „rákötve”<sup>1</sup>. Ilyenkor, ha a virtuális gép DHCP-vel kér dinamikus IP-címet, akkor a fizikai hálózatunkon lévő DHCP szerverhez fordul. Ha ott nincs DHCP-kiszolgáló, vagy az nem oszt ki neki IP-címet (mert pl. MAC cím alapú szűrést használ), akkor érdemes átállítani NAT üzemmódba a virtuális gépet.

Állítsuk át a virtuális gép hálózati kártyáját NAT módba!

---

<sup>1</sup> A VMware Player kezeléséről és részletesen a hálózatkezeléséről a *Mérés labor 4.* kapcsolódó segédletében lehet olvasni, <http://www.mit.bme.hu/oktatas/targyak/vimia315/feladat>



Ha ezt bekapcsolt állapotban tesszük meg, akkor nyilván a virtuális gépen még a régi IP-címünk él, ezért ilyenkor újat kell kérni.

A legegyszerűbb módja ennek a teljes hálózati stack újraindítása, bár ennél nyilván vannak finomabb megoldások is.

```
sudo systemctl restart network # ez disztribucionkent eltero lehet
```

A további hálózati beállítások megnézéséhez hasznosak lehetnek még a következő parancsok:

```
ip route # routing tabla listazasa
cat /etc/resolv.conf # DNS szerverek listaja
```

## 8. Parancstörténet

Most, hogy már elég sok parancsot kiadtunk, tudunk közöttük navigálni.

A FEL és LE billentyűk segítségével lépkedjünk a korábbi parancsok között.

A konzolkimenetben való navigációra a Shift+PgUp és Shift+PgDown billentyűkombinációk használhatók.

A korábbi parancsok között a Ctrl+R megnyomása után lehet keresni („reverse incremental history search mode”). Ilyenkor, ha valamit beírtunk már, akkor a Ctrl+R többszöri megnyomásával lehet visszafelé lépkedni a korábbi parancsok között. Ha megtaláltuk, amit keresünk, akkor az Enter segítségével végrehajthatjuk, vagy az Escape segítségével szerkeszthetjük is.

A Ctrl+U segítségével lehet a parancs kurzor előtti részét kitörölni.

## 9. Szövegszerkesztők használata

Alapértelmezetten általában rengeteg parancssori szövegszerkesztő áll a rendelkezésünkre. Ezek közül mindenki ízlés szerint válogathat.

Kezdeként talán az mcedit és nano programokat érdemes kipróbálni.

```
mcedit hello.txt
```

Itt az alsó sorban ki van írva, hogy az adott F\* billentyűk mit csinálnak, pl. az F10 a kilépés. További hasznos billentyűkombináció a CTRL+O, mely kilépés nélkül a háttérben futó shellre vált, így gyorsan tudjuk tesztelni az elkészített szkripteket.

```
nano hello.txt
```

Itt az alul szereplő betűk elé a Ctrl billentyűt kell hozzárakni a funkció eléréséhez, pl. Ctrl+O a mentés kódja.

## 10. Távoli elérés

A virtuális gépünket könnyen elérhetjük távolról is. Windowsos kliens esetén erre való a Putty<sup>2</sup>, Linux esetén pedig az ssh (Secure SHell) parancs. Utóbbi esetén távoli gép adatait felhasználó@gépnev formában kell megadni. Próbáljunk távolról bejelentkezni a virtuális gépre a GAZDA gépről.

Annyi előnye már rögtön van a megoldásnak, hogy ha a virtuális gép ablakában nem működött a másolás a gazda- és a vendéggép között, akkor a gazdagépen lévő terminálból (vagy Putty-ból) már könnyedén tudunk például a dokumentációba parancsokat átmásolni<sup>3</sup>.

Arra figyeljünk, hogy a néhány Linux disztribúció esetén tradicionálisan úgy működik a másolás, hogy ha az egerrel kijelölünk valamit, akkor az egyből bekerül a vágólapra és a jobb gomb utána már rögtön be is illeszti. Ez a viselkedés a különböző terminálalkalmazások esetén eltérhet.

Most egyelőre lépünk is ki a távoli konzolból, erre szolgál az exit:

```
exit
```

Ha fájlokat szeretnénk másolni két linuxos gép vagy egy linuxos és egy windowsos között, akkor arra az SCP (Secure CoPy) protokoll a legegyszerűbb megoldás. Már nem lepődünk meg, hogy ehhez Linux alatt az scp program használható, Windows alatt pedig például a WinSCP<sup>4</sup>.

Keressük ki az scp leírásából, hogy hogyan tudjuk átmásolni a gazdagépre az imént létrehozott hello.txt fájlt.

```
# ezt a gazdagepen adjuk ki, a parameterezesben az scp sugoja segit  
scp ... ..
```

(Segítség: a cp parancshoz hasonlóan a forrás- és célfájlt kell megadni, a különbség itt annyi, hogy távoli gép esetén felhasználó@gépnev: prefixet kell alkalmazni.)

---

<sup>2</sup> Putty: <http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html>

<sup>3</sup> Okulásként: korábbi félévekben volt olyan hallgató, aki az elkészült házi feladat szkriptet forrásfájlként nem adta le, pusztán a dokumentációba rakott egy be néhány képernyőképet róla, mondván, hogy nem sikerült kimásolni a virtuális gépből. Az ilyet nem szeretjük, nem igazán informatikushoz méltó megoldás.

<sup>4</sup> WinSCP: <http://winscp.net>

## 11. Telepítés

Ha valami hiányzik a virtuális gépről, akkor azt a csomagkezelő segítségével könnyen tudjuk telepíteni, ha van internetelérése a virtuális gépnek. A csomagkezelő neve és használata disztribúciófüggő, Fedora alatt így kell használni:

```
sudo yum install <package_name>
```

Próbáljuk meg például telepíteni a nano csomagot (a csomagkezelő megkeresi, majd egy idő után szól, hogy már fel van telepítve).

## 12. Leállítás

Munkánk végeztével az operációs rendszert le kell állítani, vagy szükség esetén újraindítani. Linux környezetben ezt a `/sbin/halt` és `/sbin/reboot` parancsokkal tehetjük meg.

## 1.2 Python alapok

A következő feladatban a Python nyelv alapvető funkcióit nézzük meg.

### 1. A Python telepítése

(A gyakorlaton ezt nem kell elvégezni, ott a kiadott Linux virtuális gépet használjuk.)

Windows operációs rendszerekhez telepítőt a python.org oldalról tölthetünk le, Linux alatt legtöbbször fel is van telepítve. Arra figyeljünk, hogy jelenleg két párhuzamos verziót fejlesztenek még, a 2.x régi és a 3.x új vonalat. A tárgy keretében a 3.x verziót nézzük már, azonban ezzel még néhány korábbi modul és leírás nem kompatibilis.

### 2. A Python elindítása

Interaktív módban történő indításhoz egyszerűen írjuk be, hogy:

```
python3
```

**FIGYELEM:** Linux esetén a python a 2.x verziót indítja el, nekünk a 3.x kell!

### 3. Változók definiálása

Változók beállítása az egyenlőségjellel történik. A változók neveiben a C és ahhoz hasonló nyelvekben megszokott karaktereket használhatjuk. Az aritmetikai operátorok a megszokott módon működnek. Az utasításokat nem szükséges speciális karakterrel lezárni, elég egy ENTER-t ütni.

```
flying_circus = 1969
now = 2015
years = now - flying_circus
IRF = "VIMIA370"      # this is a comment
quote = 'We are the Knights who say... "NI!'"
quote2 = "Bring us a " + "shrubbery!"
```

A sztringeket megadhatjuk `'''` és `''` között is, elvileg nincs különbség (csak annyi, hogy így nem kell a másik fajta karaktert abban az esetben helyettesíteni).

Változók értékét az alábbi módokon írhatjuk ki:

```
print(quote)
years
print("1st {0} episode was aired {1} years ago.".format('Flying Circus', years))
```

TIPP: Bonyolultabb összefűzésekhez használható a string típus `format()` művelete.

TIPP: Figyeljünk arra, hogy a Python érzékeny a kis- és nagybetűk közti különbségekre.

```
print(irf)                # hiba: name 'irf' is not defined
```

TIPP: A szöveges változók tömbként indexelhetőek, akár egy részük is lekérhető.

```
print(quote2[0])         # B, tehát 0-tól indexelünk
print(IRF[-2])           # 7, negatív index a végétől számolódik
```

```
quote[3:6]          # are
quote2[-10:]       # shrubbery!
```

A `s[i:j]` jelölés az úgynevezett szeletelés (slicing), ami visszaadja azokat a  $k$  indexű elemeket, ahol  $i \leq k < j$ .

Fontos, hogy a Python sztringek utólag nem módosíthatóak.

```
IRF[0] = "B"        # hiba: str does not support assignment
IRF = "BMEVIMIA370" # Ez nem okoz hibát!
```

#### 4. Listák kezelése

```
floats = ["bread", 'apples', 'duck']
floats.append("witch")
print(floats[1:2])    # ['apples']
print(len(floats))    # 4
```

A listákon is hasonló alapl műveletek végezhetőek el, mint a sztringeken, ugyanis mindketten úgynevezett sorozatok (sequence). A listák a sztringekkel ellentétben azonban módosíthatóak.

TIPP: Listák egymásba ágyazása is lehetséges.

#### 5. Szótár (Dictionary)

Pythonban az asszociatív tömböt (vagy mapet) dictionarynek nevezik, de az elnevezéstől függetlenül ez ugyanúgy kulcs-érték párok összerendelése. A kulcsok tipikusan sztringek, értéknek pedig bármilyen Pythonban ismert típus adható.

```
nums = {"egy" : 1, "ketto" : 2, "harom" : 3}
nums["negy"] = 4
nums["egy"]    # 1
nums.keys()    # ["egy", "ketto", "harom", "negy"] (sorrend változhat)
nums["ot"]     # HIBA! KeyError nincs ilyen kulcs
```

#### 6. Külső parancsok hívása

Python alatt is lehetőségünk van külső alkalmazások meghívására, azok kimenetének feldolgozására – ilyenkor azonban figyeljünk arra, hogy a platformfüggetlenség elveszhet! Különösen legyünk tekintettel arra, ha felhasználói bemenet alapján generálunk külső hívásokat, hiszen ilyenkor előfordulhat az ún. *shell injection támadás*<sup>5</sup>.

```
import subprocess
subprocess.call(["ls", "-l"])
```

Ha fel szeretnénk dolgozni az eredményt, akkor célszerű a `check_output` parancsot használni, amivel a parancs kimenetét eltárolhatjuk.

<sup>5</sup> Lásd még röviden: <http://blog.littleimpact.de/index.php/2008/08/11/avoiding-shell-injection-in-ruby-python-and-php/>

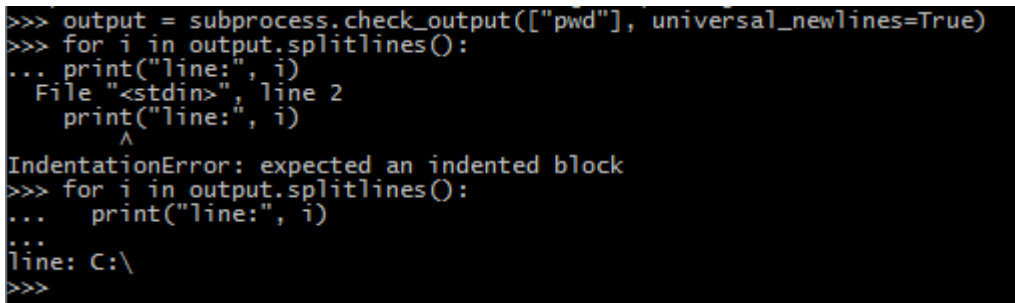


```
output = subprocess.check_output(["ls", "-l"], universal_newlines=True)
for i in output.splitlines():
    print("line:", i)
```

A Python a ciklusmagot leíró blokkot behúzással különíti el a zárójelezés helyett (a behúzáshoz lehet szóközt vagy tabulátort használni, de egyszerre csak az egyiket).

Az `universal_newlines` paraméter azért szükséges, mert így a parancs kimenetét nem bájtfolymként, hanem sztringként kapjuk vissza.

TIPP: Ha az interpreterbe írjuk be a fenti sorokat, akkor a `for` utasítást tartalmazó sor beírása után a következő sorban három pontot ír ki, de ez még az esetleges belső blokkot csak jelzi. Ha ténylegesen azt szeretnénk, hogy az utasítás a `for` cikluson belül hajtódjon végre, akkor bentebb kell kezdeni. Az alábbi ábra ezt szemlélteti.



```
>>> output = subprocess.check_output(["pwd"], universal_newlines=True)
>>> for i in output.splitlines():
... print("line:", i)
File "<stdin>", line 2
    print("line:", i)
    ^
IndentationError: expected an indented block
>>> for i in output.splitlines():
...     print("line:", i)
...
line: C:\
>>>
```

## 7. Fájl soronkénti olvasása

Gyakori feladat, hogy egy fájlt soronként kell olvasni és valamilyen műveletet kell elvégezni minden egyes sorára. Ehhez először meg kell nyitni a fájlt.

```
f = open("/etc/passwd")
```

A soronkénti feldolgozást legegyszerűbben a `readlines()` utasítással és egy `foreach` ciklussal valósíthatjuk meg:

```
f = open("/etc/passwd")
for line in f.readlines():
    if line.find(":") >= 0:
        print(line.partition(":")[0])
```

```
f.close()
```

TIPP: Fájlokat a Python `with` kontextus-kezelőjével is megnyithatunk. Ez leegyszerűsíti a hibakezelést, és a blokkból kilépve automatikusan lezárja a fájlt.

```
with open("/etc/passwd") as f:
    for line in f.readlines():
        if line.find(":") >= 0:
            print(line.partition(":")[0])
```

Python 3.1-től kezdve lehetőség van több fájl megnyitására egy blokkon belül:

```
with open("/input.txt") as input, open("/output.txt", "w") as output:
    pass
```

## 8. CSV fájlkezelés

Ha egyszerűbb szöveges formátumú fájlokkal szeretnénk dolgozni, jó eséllyel találunk megfelelő Python modult, aminek a segítségével gyorsabban és könnyebb értelmezhetjük fájlunkat. Úgynevezett *Comma-separated values* (CSV) fájlok kezelésére a Python `csv` modulja ad támogatást.

Az alábbi példában az `input.csv` fájlt beolvassuk, majd az első oszlop minden sorát – kivéve a fejléct – „IRF”-re lecserélve kiírjuk az `output.csv` fájlba. A `csv.reader` első paramétere egy fájlobjektum, a `delimiter=` paraméter pedig beállítja, hogy a CSV fájlban vesszőt várunk elválasztó karakterként. Egy `for` ciklus segítségével soronként visszkapjuk az `input.csv` fájl tartalmát egy listaként, ahol a lista elemei a CSV fájl beolvasott sorának mezői.

A példa lefuttatásához előbb hozzunk létre egy CSV fájlt az aktuális könyvtárban, például a következő tartalommal:

```
kod,nev,letszam
VIMIA370,"Intelligens rendszerfelügyelet",214
VIMIA219,"Operációs rendszerek",401
```

Az első sor a CSV fájl fejléce, ez határozza meg az oszlopok neveit. A többi sorban vannak az egyes rekordok, jelen esetben vesszővel elválasztva (de ez lehetne más karakter is, pl. magyar területi beállítások esetén gyakori a pontosvessző is). Az egyes mezők tartalmát lehet idézőjelek közé rakni, így pl. akkor tartalmazhatnák a mezőelválasztó karaktert is.

Ezután ezt így tudjuk feldolgozni.

```
import csv

input = open("./input.csv")
output = open("./output.csv", "w")
reader = csv.reader(input, delimiter=",")
writer = csv.writer(output)
rownum = 0
for line in reader:
    # A fejléct nem akarjuk módosítani.
    if rownum != 0:
        print(line[0])
        writer.writerow(line) # Kiírjuk a sort.
        rownum += 1

input.close()
output.close()
```

TIPP: figyeljünk arra, hogy a fenti kódban nincs még hibakezelés, tehát például ha hagyunk egy üres sort a CSV fájl végén, akkor hibaüzenetet kapunk.

TIPP: használhatjuk a `csv.DictReader` osztályt is. Ilyenkor lista helyett egy Python dictionaryt kapunk vissza. A kulcsok az oszlopok nevei lesznek, az értékek pedig a mezőértékek.

```
import csv

with open("./input.csv") as input:
    reader = csv.DictReader(input, delimiter=",")
    for line in reader:
        print(line["kod"]) # Feltételezzük, hogy van kod nevű oszlop a fájlban
```

TIPP: természetesen van `csv.DictWriter` osztály is, ami dictionaryt vár a `writerow` metódusa paramétereként. Figyeljünk, hogy a `csv.DictWriter`-nek van egy `fieldnames` kötelező paramétere, ami egy listát vár az oszlopok neveiről.

A `csv` modulban sok egyebet be lehet még állítani (pl. milyen a fájlok kódolása). használat előtt érdemes átfutni majd a dokumentációját.

### 1.3 Python szkriptek készítése

Az előző feladatban megnéztük, hogy hogyan tudjuk a Pythont interaktív módon használni. Készítsünk most egy-két alapvető szkriptet, amiket később többször, akár más paraméterekkel is le tudunk futtatni.

#### 1. Hello World Python szkript megírása és futtatása

Ehhez érdemes két darab `ssh` kapcsolatot használni, az egyikben a szövegszerkesztőben legyen nyitva a szkript fájl, a másikban pedig futtathatjuk, és ellenőrizhetjük az eredményt.

TIPP: Komolyabb munkák esetén érdemes egy Python IDE-t telepíteni (ilyen pl. a **PyDev**, amely az Eclipse rendszerbe telepíthető)

Írjuk be a következő minimális szkriptet, majd mentsük el `hello.py` néven:

```
#!/usr/bin/env python3

# My first Python script
print("Hello world!")
```

TIPP: az előadáson szerepelt, hogy a `#!` kezdetű első sor szerepe speciális, ez mondja meg, hogy milyen programmal kell futtatni az adott fájlt.

Ahhoz, hogy futtatni tudjuk, először még futtatás jogot kell rá adni. A másik terminál ablakban adjuk ki a következő parancsot:

```
chmod u+x hello.py
```

Ezután futtassuk is le a szkriptünket:

```
./hello.py
```

A ./ azért kell elé, mert Linuxban általában nincs bent a . (az aktuális könyvtár) a PATH-ban, és a shell csak a PATH-ban szereplő könyvtárakban keresi a futtatandó állományt.

Ha minden jól megy, akkor büszkén hátra is dőlhetünk, elkészült az első Python szkriptünk.

## 2. Paraméterkezelés

Nézzünk most egy kicsivel bonyolultabb szkriptet, ami már valami paramétert is vár. Írjuk meg a hello2.py szkriptet, ami első paraméterként egy szót vár, majd ezt kiírja a hello után. A második paramétere opcionális, ez egy fájlnev lehet. Ha ez meg van adva, akkor ebbe a fájlba is kiírja a kimenetét. A szkript ellenőrizze, hogy létezik-e már ez a fájl, ha igen, akkor ne fusson le.

- Gondoljuk előtte végig, hogy milyen értékekkel tesztelnénk az elkészült megoldást!
- Mindenképpen meg kéne nézni a következőket:
  - ha nem adunk meg paramétert, hibát kell jeleznie,
  - ha kettőnél több paramétert adunk meg, hibát kell jeleznie,
  - ha egy paramétert adunk meg, csak a képernyőre kell kiírnia,
  - ha két paramétert adunk meg és a másodikként megadott fájl létezik, hibát kell dobnia,
  - ha két paraméter adunk meg és a másodikként megadott fájl nem létezik, a képernyőre és a fájlra is kell írnia.

A feladatra egy lehetséges megoldás (ez a szkript megtalálható a tantárgy honlapjáról letölthető példa szkripteket tartalmazó zip fájlban):

```
#!/usr/bin/env python3
...
This script greets a user on screen and optionally in a file.
This script is implemented without argparse (not recommended!).
Usage: ParameterHandlingBasic.py name [outfile]

Created on 2013.02.19.
@author: Gergo Horanyi
...

import sys
import os.path

# Checking command line arguments
if len(sys.argv) == 1 or len(sys.argv) > 3:
    print("Usage:", sys.argv[0], "name [outfile]")
    sys.exit(1)
elif len(sys.argv) == 3 and os.path.exists(sys.argv[2]):
    print("File", sys.argv[2], "should not exists!")
    sys.exit(2)
greeting = "Hello " + sys.argv[1] + "!"
print(greeting)
```

```
if len(sys.argv) == 3:
    f = open(sys.argv[2], 'w')
    f.write(greeting)
    f.close()
```

- Nézzük végig a megoldást!
- Próbáljuk tesztelni különböző bemenetekkel!
- Írjuk át argparse modul használatával!

TIPP: Jöjjünk rá, hogy a sys.argv tömb első eleme micsoda!

### 3. Önálló szkript készítése

Készítsünk egy saját szkriptet, mely a következő paramétereket várja:

```
createUserDirs.py shell path
```

A szkript dolgozza fel a /etc/passwd fájlt, és hozzon létre a path paraméterként megadott könyvtárban minden egyes felhasználónak egy, a felhasználó login nevével megegyező könyvtárat, akinek a shell-je a shell paraméterben megadottal egyezik. A path paraméter opcionális, ha nincs megadva, akkor az aktuális könyvtárban hozza létre a könyvtárat. Ha meg van adva, akkor a szkript ellenőrizze, hogy létezik-e az adott könyvtár. A szkript egy lehetséges helyes meghívása:

```
createUserDirs.py --shell /bin/bash --path /tmp
```

### 4. Hogyan tovább?

Ebbe a kis időbe a Python ilyen mélységben fért bele. A Python honlapján azonban egy hosszabb és jól követhető leírás található [2]. Szintén érdemes még megnézni a Google *Python class* című írásait, ez egy nagyon tömör és hasznos bevezető [3].

## 2 Windows és PowerShell

A feladatokat egy Windows 8.1 virtuális gépen érdemes végrehajtani. A virtuális gépre az alap Windows 8.1 telepítésen kívül nem kell semmi, abban elérhető a PowerShell 4.0

(A PowerShell legújabb, 5.0-ás változata jelenleg preview változatban van, ez a WMF 5.0 Preview<sup>6</sup> telepítésével próbálható ki.)

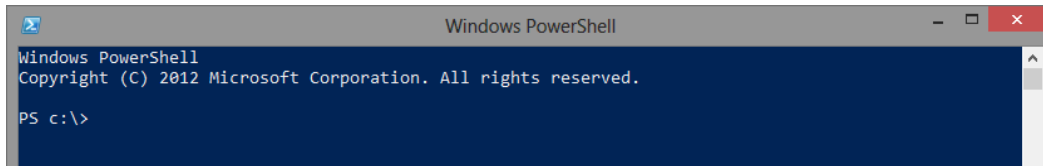
### 2.1 PowerShell konzol használata

A következő feladatban a PowerShell konzol alapfunkcióit tekintjük át.

1. Nyissuk meg a PowerShell konzolt!



2. Ennek hatására megnyílik a PowerShell konzol.



3. Hajtsunk végre egy parancsot!

**FIGYELEM:** szokjuk meg az automatikus kiegészítés használatát (TAB). Ha több lehetséges kiegészítés van, akkor a TAB ezek között vált (Shift+TAB visszafelé lépked). Ha sok lehetséges kiegészítés van, akkor érdemes egy-két betűvel többet begépelni, és utána használni csak a TAB-ot.

Kérdezzük le az elérhető parancsokat!

```
Get-Command
```

4. Adjunk át valami paramétert a fenti cmdletnek!

Legyen ez a -Verb, amivel a cmdletben lévő igére lehet szűrni.

```
Get-Command -Verb Invoke
```

**TIPP:** A paraméter nevénél is érdemes használni az automatikus kiegészítést.

**TIPP:** ha Wordből/PDF-ből másolunk PowerShell parancsokat, akkor arra érdemes figyelni, hogy a Word néha lecseréli a beírt karaktereket tipográfiaiailag megfelelőbbre (pl. " helyett " vagy - helyett -). Ezeket a PowerShell nem szereti, és nehéz is észrevenni a különbséget elsőre.

<sup>6</sup> Windows Management Framework 5.0 Preview, csak en-us területi beállítással telepíthető Pro vagy Enterprise változatra. URL: <http://www.microsoft.com/en-us/download/details.aspx?id=45883>

5. Nézzük meg, hogy milyen további paraméterei vannak a `Get-Command` cmdletnek a súgó segítségével!

```
Get-Help Get-Command
```

Nézzük meg, hogy milyen főbb részekből áll egy súgó (Name, Syntax, Description...).

Próbáljuk értelmezni a szintaxis leírását, azonosítsuk a kötelező és opcionális paramétereket. Hogy jelöli a PowerShell, ha egy paraméter pozícióval is megadható, és hogyan, ha egy paraméter switch típusú?

Nézzük meg a teljes súgót is, figyeljük meg a példák (examples) részt is!

```
Get-Help Get-Command -Full | more
```

A `more` esetén a Space billentyűvel tudunk lapozni, a `q` segítségével pedig bármikor kiléphetünk.

6. Másolás és beillesztés

A PowerShell konzol esetén az egérrel egyszerűen tudunk kijelölni szöveget. Ezután a jobb gomb megnyomásával kerül az át a Vágólapra. Ugyanígy a jobb gomb megnyomásával tudunk beilleszteni szöveget is.

Próbáljunk meg kimásolni valami szöveget, majd azt beilleszteni és végrehajtani!

7. Parancsok kiválasztása grafikusán

Használjuk a `Show-Command` cmdletet:

```
Show-Command
```

Nézzük meg milyen modulok vannak! Válasszuk ki a `NetAdapter` modult, majd onnan a `Get-NetAdapter` cmdletet! Nézzük meg, hogy milyen különböző paraméterezései lehetnek (az egyes fülek, amik a különböző `ParameterSet`eknek felelnek meg). Paraméterezzük úgy fel, hogy a rejtett interfészeket is megjelenítse!

8. Parancstörténet használata

Most már jó pár parancsot beírtunk, így lehet közöttük lépkedni.

A `FEL` billentyű segítségével lépkedjünk vissza arra a parancsra, amikor az *Invoke* ígéhez tartozó parancsokat kérdeztük le, majd hajtsuk is végre!

Az `F7` lenyomásával hozzuk elő a parancstörténet ablakot, majd hajtsunk végre egy korábbi utasítást!

## 2.2 PowerShell alapok

Most nézzük meg a PowerShell alapjait: változókezelés, objektumok használata, főbb cmdletek.

1. Változókezelés és behelyettesítések

Adjunk értéket egy változónak, majd nézzük meg a típusát és azt, hogy milyen műveleteket lehet végrehajtani vele.

```
$subject = "IRF"
$subject.GetType()
Get-Member -InputObject $subject
```

TIPP: ugye a változónévnél is használtuk az automatikus kiegészítést?

Hívjuk meg a változónkon értelmezett egyik metódust:

```
$subject.Substring(1)
```

Próbáljuk ki a különböző behelyettesítési módszereket:

```
echo "Hello $subject"
echo 'Hello $subject'
```

Az echo csak a Write-Output aliasa, ezt könnyen ellenőrizhetjük a Get-Alias echo paranccsal.

Ha a változó egy tulajdonságára akarunk hivatkozni egy kiíratás során, akkor a következő formát kell használni:

```
# a megjegyzes jele #
# hibas
echo "Hossz: $subject.Length karakter"
# helyes
echo "Hossz: $($subject.Length) karakter"
```

Ha bonyolultabb adatstruktúrát akarunk használni, akkor lehet tetszőleges .NET objektumot példányosítani.

```
$stack = New-Object System.Collections.Stack
$stack.Push(6)
$stack.Pop()
```

Tömböt és hash táblát egyszerűen lehet létrehozni:

```
# tömb
$array = "Hello", 5, "IRF"
$array[2]
$array[1]
$array.Length

# hash tabla
$values = @{ "low" = 1; "high" = 2 }
$values["low"]
```

## 2. Alap parancsok és csővezeték (pipeline) kezelése

Hozzunk létre egy könyvtárat és pár fájlt, hogy legyen min dolgozni a következő feladatokban.



```

mkdir test
cd test
echo "aaaa" > a.txt
"bbbb" | out-file b.txt           # lehet így is
$c = "cccc"
Out-File -FilePath c.txt -InputObject $c # vagy így
Get-ChildItem                     # ez visszaad egy gyűjteményt

```

Válasszuk ki csak néhány oszlopot:

```
Get-ChildItem | Select-Object basename, extension
```

Jelenítsük meg listaként az eredményt:

```
Get-ChildItem | Select-Object basename, extension | Format-List
```

Használjuk a rendezést:

```
Get-ChildItem | Sort-Object -Descending
```

Szűrjünk ki néhány elemet:

```

Get-ChildItem | Where-Object {$_.Length -gt 15}
# ugyanez rövidebben
ls | ? {$_.Length -gt 15}

```

Végezzünk el valamilyen műveletet a csővezeték minden elemére! Írjuk ki a fájlok nevét és hosszát egy karaktersorozatként egy-egy sorba:

```

Get-ChildItem | ForEach-Object {Write-Output "$($_.Name): "$($_.Length)}
# ugyanez rövidebben
dir | % {echo "$($_.Name): "$($_.Length)}

```

Keressük ki, hogy mekkora a legnagyobb fájl mérete:

```

(Get-ChildItem | Measure-Object -Property Length -Maximum).Maximum
# rövidebben, kihasználva, hogy a Property az első pozícionálás paraméter
(ls | measure Length -Maximum).Maximum

```

Nézzünk egy picit bonyolultabb műveletet. Kérdezzük le, hogy a Windows könyvtárban lévő log kiterjesztésű fájlok tartalmában hány sorban szerepel a starting szó!

```

(Get-ChildItem -Path C:\Windows *.log | Get-Content | select-string
"starting").Length

```

Ebből a pár példából már remélhetőleg érződik, hogy itt egy picit más stílusú kódot kell írni, mint pl. Java vagy C# esetén. A csővezeték használatával tömören lehet egy bonyolult feldolgozási sorozatot végrehajtani, sokkal kevesebbszer van szükség segédváltozók bevezetésére.

TIPP: amikor csak a konzolba írunk be rövidebb parancsokat, akkor érdemes a cmdletek rövid neveit használni. Később, amikor majd újrahasznosítható szkripteket készítünk, akkor ott inkább a hosszabb, beszédesebb neveket használjuk majd, hogy könnyebben olvasható legyen a kód.

### 2.3 PowerShell szkriptek készítése

Miután megnéztük, hogy hogyan lehet parancsokat beírni a PowerShell konzolba, készítsünk egy-két egyszerűbb szkriptet, amiket később többször meg lehet hívni.

#### 1. ExecutionPolicy beállítása

Ahhoz, hogy szkripteket tudjunk futtatni, lehet, hogy be kell még állítani a következőket (egyébként a futtatáskor hibát kapunk). A PowerShell alapértelmezetten nem hagy szkripteket futtatni, csak ha azok digitálisan alá vannak írva. Ezt a következőképp tudjuk megváltoztatni:

Indítsunk el *rendszergazdaként* egy PowerShell konzolt, majd hajtsuk végre a következő parancsot:

```
Set-ExecutionPolicy RemoteSigned
```

Ennek hatására csak a távoli helyről futó vagy letöltött szkriptek esetén követeli meg a digitális aláírás meglétét.

Zárjuk be a rendszergazda konzolt, és indítsunk egy normál felhasználóit helyette.

#### 2. Hello World szkript

Nyissunk egy jegyzetfüzetet, és másoljuk bele a következő kódot. Az eredményt mentjük el `Out-Hello.ps1` néven. Figyeljünk a `.ps1` kiterjesztésre!

```
param( [Parameter(mandatory=$true)][string] $Name )
```

```
Write-Output "Hello $Name!"
```

Hajtsuk végre a létrehozott szkriptet először paraméter nélkül:

```
.\Out-Hello.ps1
```

Mit tapasztaltunk?

Hajtsuk most végre a létrehozott szkriptet megadva a paramétert:

```
.\Out-Hello.ps1 -Name "IRF"
```

#### 3. Súly hozzáadása

Hasznos, ha a szkriptünknek van fejkomentje, ami a legfontosabb információkat tartalmazza. PowerShellben ezt meg lehet úgy írni, hogy utána a `Get-Help` segítségével a beépített cmdletekéhez hasonló súlyt tudjunk generálni.

Egészítsük ki a fenti szkriptünket a következő fejléccel:

```
<#
.SYNOPSIS
Writes out a greeting message.

.PARAMETER Name
The name to greet.

.NOTES
Sample script for IRF.
#>

param( [Parameter(mandatory=$true)][string] $Name )
```

```
Write-Output "Hello $Name!"
```

Próbáljuk is ki, hogy megjelenik-e a súgó:

```
Get-Help .\Out-Hello.ps1
```

TIPP: figyeljünk arra, hogy ha akár csak egy karaktert is elírunk valamelyik kulcsszóban, akkor semmit nem ír ki a súgóból.

#### 4. Integrated Scripting Environment (ISE) használata

Nyissuk meg az előbb elkészített szkriptünk az *ISE* programban (jobb gomb a tálcán lévő PowerShell ikonon, majd onnan Windows PowerShell ISE)! Ismerkedjünk meg a felülettel!

Helyezzünk el egy töréspontot a kiírató sorra!

Indítsuk el debug módban (F5)! Az ISE jelzi, hogy jelenleg hol tart a végrehajtás.

A `$Name` változó fölé mozdítva a kurzort megnézhetjük annak értékét.

Ha meg szeretnénk adni, hogy mit kapjon paraméterként a szkript induláskor, akkor az alsó konzol részen navigáljunk el a szkript könyvtárába, és innen hívjuk meg a szkriptet a megfelelő paraméterekkel<sup>7</sup>.

### 2.4 Példa szkriptek és önálló feladatok

#### 1. Szkript készítése önállóan

Készítsük el a következő paraméterekkel rendelkező szkriptet:

```
Get-BigProcess [-Name] <String> [[-Size] <Int32>]
```

A szkript lekérdezi a futó folyamatokat, és visszaadja a `Name` paraméterrel megegyező nevéük közül azokat, amiknek a fizikai memórafoglalása (working set) nagyobb, mint a `Size` paraméter megadójában értve. A szkript ellenőrizzé a bemeneti paramétereket, a `Name` paraméter kötelező, a `Size` opcionális. A szkript egy lehetséges meghívása:

```
Get-BigProcess -Name "notepad" -Size 15
```

<sup>7</sup> Ha tud valaki más, jobb megoldást, kérem, hogy írja meg!

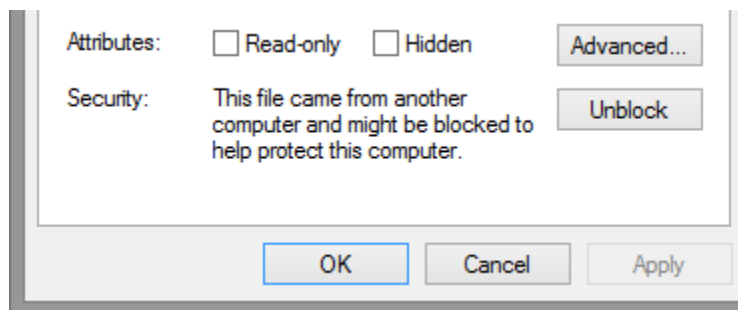
## 2. Komplexebb szkriptek

A gyakorlati anyagok között szerepel két példa szkript is:

- `Get-PrincipalInAcl.ps1`: lekérdezi egy ACL-ben szereplő neveket és SID-eket. Ehhez egy külső programot, a Sysinternals csomag PsGetSid programját használja, a szkript ezt hívja a SID meghatározásához.
- `Get-FaceBookFriendsPicture.ps1`: Letölti egy adott személy Facebookon lévő barátainak a profilképeit. Ez se túl bonyolult, csak van benne hibakezelés és részletes haladás kiírása, azért hosszabb a kód (a lényegi rész 3 sor lenne). A PowerShell 3-ban megjelent REST és webes hívásokat megvalósító cmdleteket használja.

Nézzük meg a kódjukat, próbáljuk megérteni a működésüket.

Internetről letöltött szkriptek futtatása esetén hibát dobhat a PowerShell, hogy nincs digitálisan aláírva a szkript, holott, korábban megváltoztattuk az *ExecutionPolicy* beállítást *RemoteSigned* értékre. Ez azért van, mert a Windows az Internetről letöltött fájlokat megjelöli, és azokat távoli (=veszélyes) fájlkként kezeli. Ezt a fájl tulajdonságlapján tudjuk kikapcsolni az *Unblock* gombbal. (Ezt viszont tényleg csak megbízható fájlokra tegyük meg!)



### 3 Összefoglalás

A gyakorlat során megnéztük azokat az alapvető ismereteket, amik a házi feladatnál az elinduláshoz kellenek. Megismerkedtünk többek között a virtuális gépek kezelésével, alapvető Linux műveletekkel, a Python alapjaival, valamint a PowerShell szkriptek készítésének bevezető lépéseivel.

### 4 További információ

#### Linux

- [1] Szandi Lajos, Leírás a Unix használatáról, BME HIT, <http://www.hit.bme.hu/~szandi/unix/>
- [2] Python Software Foundation. "The Python Tutorial". <http://docs.python.org/3.4/tutorial/>
- [3] Google, Google's Python class, <https://developers.google.com/edu/python/> (még Python 2.x!)

#### Windows

- [4] TechNetKlub, Short Online Trainings (SHOT), PowerShell, <http://technetklub.hu/shot/#5>
- [5] Soós Tibor, Microsoft PowerShell 2.0 rendszergazdáknak – elmélet és gyakorlat, 2010, [https://technetklub.hu/Downloads/Download.ashx?shareid=1&path=PDF%5cE-Book+-+PowerShell+2.0+tank%c3%b6nyv%5cMicrosoft Powershell 2 konyv.pdf](https://technetklub.hu/Downloads/Download.ashx?shareid=1&path=PDF%5cE-Book+-+PowerShell+2.0+tank%c3%b6nyv%5cMicrosoft+Powershell+2+konyv.pdf)