Linux, Bash és PowerShell alapok

Gyakorlati útmutató
Készítette: Micskei Zoltán, Szatmári Zoltán
Utolsó módosítás: 2012.02.22.

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. Különben nem sok mindent tanulunk belőle, nem rögzül a szintaktika.

1 Linux és Bash

A feladatokat egy VMware virtuális gépbe telepített CentOS rendszeren fogjuk végrehajtani. Ez a virtuális gép előre telepítve tartalmazza a Bash 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!

   A gépbe történő bejelentkezéshez szükséges adatok:

   - Felhasználói név: **meres**
   - Jelszó: **LaborImage**

2. A kapott terminálnál próbáljuk ki az alapvető funkciókat:

   FIGYELEM: törekedjünk arra, hogy használjuk a `TAB` billentyűvel az automatikus parancs- és fájlnév 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 konyvtar
mkdir test                  # make directory: test konyvtar letrehozasa
cd test                     # change directory: konyvtar valtasa
touch 01.txt                # fajl hozzaferesi idok modositasa
echo 02 > 02.txt            # kiiratas es fajlba atiranyitas
ls                          # konyvtar tartalmanak listazasa
ls –l                       # reszletes lista
cp 02.txt 03.txt            # copy: masolas
cat 02.txt                  # fajl tartalmanak osszefuzese es kiirasa
```

```
clear                       # kepernyo torlese
chmod a+w 02.txt            # jogosultsagok allitasa
```

3. Súgó oldalak

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

   ```
   man touch
   ```

   A manualból a `q` billentyűvel tudunk kilépni.

   További információt kaphatunk az `info` parancs segítéségével:

   ```
   info touch
   ```

4. Felhasználóváltás, privilegizált műveletek

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

   ```
   cat /var/log/messages            # permission denied az eredmeny
   ```

   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 /var/log/messages       # sajat jelszot kell utana 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 lépjünk 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
   ```

Lehetőségünk van több helyi konzol használatára, mindegyikre tetszőleges felhasználóval jelentkezhetünk be. A konzolok közötti váltást a `Ctrl+Alt+FX` (X=1..6, a konzol számát jelzi) billentyűkombináció teszi lehetővé. Arra figyeljünk, hogy VMware használata esetén a `Ctrl+Alt` a vendég gépből való kiváltást okozza, így ilyenkor a `Ctrl+Alt+Space+F1` kombinációt kell pl. alkalmazni.

A grafikus felületre (ha az el volt indítva) a `Ctrl+Alt+F7` kombinációval tudunk visszaváltani.

5. Billentyűzetkiosztás változtatása

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

   ```
   sudo loadkeys us
   ```

   A továbbiakban mindenki olyan kiosztást használjon, ami neki jobban kézre áll.

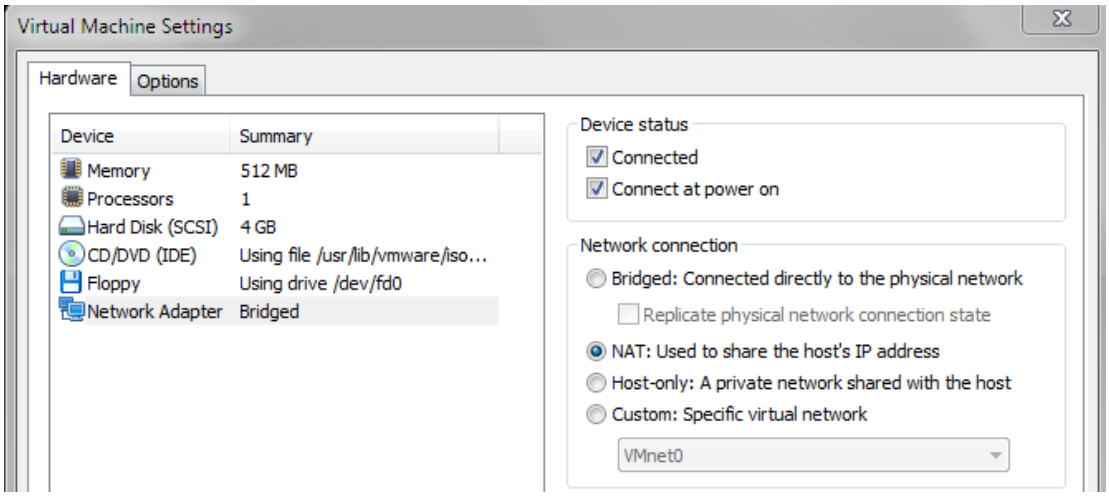
6. Hálózati beállítások

   A legfontosabb hálózati beállításokat az `ifconfig` paranccsal kérdezhetjük le.

   ```
   ifconfig
   ```

   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”[1]. Ilyenkor, ha a virtuális gép DHCP-vel kér dinamikus IP-címet, akkor a fizikai hálózatunkon lévő DHCP szerverhez fordul. Ha ott nincs DHCP szerver, 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!



[1] 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 /etc/init.d/network restart  # ez disztribucionkent eltero lehet, például
                                  # Ubuntu alatt /etc/init.d/networking
```

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

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

7. 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 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` segítésével lehet a parancs kurzor előtti részét kitörölni.

8. Szövegszerkesztők használata

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

   Kezdésként 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.

9. Távoli elérés

   A virtuális gépünket könnyen elérhetjük távolról is. Windowsos kliens esetén erre való a Putty[2], Linux esetén pedig az `ssh` (Secure SHell) parancs. Utóbbi esetén távoli gép adatait `felhasznalo@gepnev` formában kell megadni. Próbáljunk távolról bejelentkezni a virtuális gépre a GAZDA gépről.

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

   Arra figyeljünk, hogy a gyakorlaton használt CentOS Linux esetén tradicionálisan úgy működik a másolás, hogy ha az egérrel 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 lépjünk 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[4].

   Másoljuk át a gazdagépre az imént létrehozott hello.txt fájlt:

   ```
   # ezt a gazdagepen adjuk ki
   scp meres@<VM_IP-címe>:~/test/hello.txt /home/user/feladat
   ```

   Tehát a `cp` parancshoz hasonlóan a forrás- és célfájlt kell megadni, a különbség itt annyi, hogy távoli gép esetén `felhasznalo@gepnev:` prefixet kell alkalmazni.

10. Leállítás

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

1.2 Bash alapok

A következő feladatban a Bash parancsértelmező alapvető funkcióit nézzük meg.

1. Változók definiálása

   Nézzük meg először, hogy milyen beállított változóink vannak:

[2] http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html
[3] 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.
[4] http://winscp.net

```
set
```

Van egy jó pár! Ezeknek az értékét el is érjük akár most is:

```
echo $PATH
```

Állítsunk be egy új változónak értéket:

```
name="meres"
```

TIPP: Figyeljünk arra, hogy az egyenlőségjel egyik oldalán se lehet szóköz.

```
name= "meres"              # hibat dob
name ="meres"              # hibat dob
```

Ennek az értékét már ki is tudjuk akkor íratni:

```
echo $name
```

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

```
echo $Name                 # ures stringet ad vissza
```

Próbáljuk ki a különböző változó behelyettesítési módokat:

```
echo "hello $name"         # idezojel, behelyettesit

echo 'hello $name'         # aposztrof, nem helyettesit be

                           # igy tudunk kozvetlenul az ertek melle irni
echo "hello _${name}_"     # itt a _ sima karakter, csak az a szerepe, hogy
                           # lassuk hol van a szohatar
```

Egy változónak értékül adhatjuk akár egy komplex parancs kimenetét is, erre szolgál az úgynevezett backtick karakter (`AltGr+7` a magyar kiosztáson):

```
IP=`ifconfig eth0 | grep "inet addr"`
```

2. Parancsok elválasztása

   Parancsok elválasztására a pontosvessző szolgál. (Ha egy sorban csak egy parancs van, akkor ezt nem feltétlen kell kitenni a sor végére.)

   ```
   ifconfig; echo "hello"; ls;
   ```

3. Pipeline használata

   Mivel a legtöbb beépített UNIX/Linux valami egyszerű funkcionalitást valósít meg, ezért az igazi erejük abban rejlik, hogy össze lehet fűzni az eredményüket. Erre szolgál a pipe (jele: |). Ilyenkor az egyik parancs kimenetét átadjuk a másiknak. Ám figyeljünk arra, hogy ilyenkor az adatok átadása formázatlan bináris formában történik, így

nekünk kell gondoskodni arról, hogy megfelelő formába hozzuk az adatokat a következő parancs elvárásainak megfelelően.

Bemeneti adatként használjuk most a `/etc/passwd` fájlt, mivel ez elég sok sort tartalmaz és jól strukturált. (Ez a fájl egyébként a helyi felhasználókat tartalmazza.)

Keressük ki azokat, akiknek a bash az alapértelmezett shelljük:

```
cat /etc/passwd | grep "bash"
```

A `grep` mintákat keres a bemenetében. Most egy egyszerű mintát használtunk.

A parancsot használhatjuk úgy is, hogy azokat a sorokat tartsa meg, amik nem illeszkednek a mintára:

```
cat /etc/passwd | grep -v "bash"        # -v, --invert-match
```

Lehet bonyolultabb mintákat is keresni reguláris kifejezések segítségével. Keressük meg például az a-val kezdődő felhasználókat (a felhasználónév a passwd fájlban a sor elején van):

```
cat /etc/passwd | grep "^a"
```

A `grep` ezen kívül rendkívül sok mindent tud még (pl. a `-n` kiírja az illeszkedő sor helyét a sor elejére), érdemes megnézni a manual oldalát.

Számoljuk meg, hogy hány sort talált az előbb a `grep`. Ezt meg lehetne tenni a `grep -c` paraméterével, de most használjuk inkább a `wc` (word count) parancsot, ami a `-l` paraméter hatására a bemenetén kapott állomány sorait számolja meg.

```
cat /etc/passwd | grep "bash" | wc -l
```

Ha fel akarjuk bontani a bemenetként kapott adatot, akkor arra több lehetőségünk is van. Legegyszerűbb esetben elég a `cut` parancs is, ennek meg lehet adni a határoló karaktert (`-d`, `--delimiter`), valamint, hogy melyik mezőket szeretnénk megtartani (`-f`, `--fields`).

Például ha az előző listából csak a felhasználónevet és a shellt akarjuk kiíratni, akkor azt megteszi a következő parancs:

```
cat /etc/passwd | grep "^a" | cut -d ":" -f 1,7
```

Összetettebb szövegfeldolgozó utasítások a `sed` és az `awk`, melyek segítségével egészen egzotikus eredményeket is könnyedén elérhetünk.

4. Fájl soronkénti olvasása

   Gyakori feladat, hogy egy fájlt soronként kell olvasni és valamilyen műveletet kell elvégezni minden egyes sorára. Erre több megoldás is létezik.

   Használhatunk egy `while` ciklust (az alábbi parancsban az egyszerűség kedvéért csak kiírjuk a beolvasott sort, de itt tetszőlegesen bonyolult műveletet el lehetne végezni):

```
cat /etc/passwd | while read line; do echo $line; done
```

Itt most kihasználtuk, hogy egy sorba lehet több utasítást is írni, ilyenkor pontosvesszővel kell el az egyes parancsokat elválasztani.

A másik lehetőség, hogy egy `for` ciklust használunk:

```
for line in `cat /etc/passwd`; do echo $line; done
```

Figyeljük meg, hogy ez ebben a formában még nem jó, ugyanis ez az alapértelmezett `$IFS` (Internal Field Separator) mentén darabol, aminek például része a szóköz is. Így például az olyan felhasználókat, akiknek a nevében szóköz van, több sorban írja ki.

Egészítsük ki a fenti utasítást, hogy helyesen működjön, azaz egy az egyben a passwd fájlt adja vissza.[5]

1.3 Bash szkriptek készítése

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

1. Hello World Bash 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.

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

   ```
   #!/bin/bash

   # My first bash script
   echo "Hello world!"
   ```

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

   ```
   chmod u+x hello.sh
   ```

   Ezután futtassuk is le a szkriptünket:

   ```
   ./hello.sh
   ```

   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 Bash szkriptünk.

[5] Az IFS-nek a következő értéket kell beállítani, ha csak a tabulátor mentén akarunk darabolni: `IFS=$'\t'`

2. Paraméterkezelés

Nézzünk most egy kicsivel bonyolultabb szkriptet, ami már valami paramétert is vár. Írjuk meg a hello2.sh szkriptet, ami első paraméterként egy szót vár, majd ezt kiírja a hello után. A második paramétere opcionális, ez egy fájlnév 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 (ezt a szkriptet már lehet másolni:):

```
#!/bin/bash
# Name:         hello2.sh
# Author:       Micskei Zoltan
# Date:         2011.03.01.
# Desc:         Customizable hello world
# Param:
#               $1 - name to greet
#               $2 - optional filepath to write the greeting, should not exist

# check number of parameters
if [ $# -eq 0 ] || [ $# -gt 2 ]; then
        echo "Usage: $0 name [outfile] "
        exit 1
fi

# check whether output file exist
if [ $# -eq 2 ] && [ -e $2 ]; then
        echo "File $2 should not exist!"
        exit 2
fi

# write greeting
msg="Hello $1!"
echo $msg

# if second parameter was supplied, write the greeting also to the file
if [ ! -z $2 ]; then
        echo $msg > $2
fi
```

- Nézzük végig a megoldást!
- A feltételekben lévő kapcsolók (`-e`, `-z`) jelentését a `test` parancs manualjából lehet kinézni (emlékszünk ugye előadásról, hogy a `[` jel csak alias a `test` parancsra).

```
man test
```

- Próbáljuk tesztelni különböző bemenetekkel!

TIPP: figyeljünk arra, hogy a `then` elé ki kell tenni a pontosvesszőt, mert az már egy külön parancs!

TIPP: figyeljünk arra, hogy a `[` után kell szóközt rakni!

3. Önálló szkript készítése

   Készítsünk egy saját szkriptet, mely a következő paramétereket várja:

   ```
   createUserDirs.sh 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 könyvtárakat. Ha meg van adva, akkor a szkript ellenőrizze, hogy létezik-e az adott könyvtár. A szkript egy lehetséges helyes meghívása:

   ```
   createUserDirs.sh /bin/bash /tmp
   ```

2 Windows és PowerShell

A feladatokat egy Windows 7 virtuális gépen fogjuk végrehajtani. A virtuális gépre az alap Windows 7 telepítésen kívül, ami tartalmazza már a PowerShell 2.0-t, csak a PowerGui[6] programot telepítettük fel.

2.1 PowerShell konzol használata

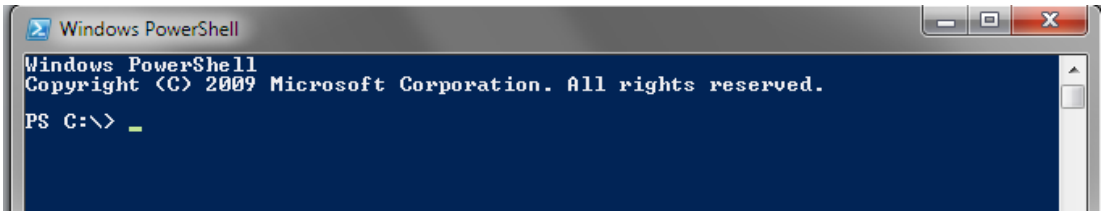
A következő feladatban a PowerShell konzol alapfunkcióit tekintjük át.

1. Indítsuk el a Windowsos virtuális gépet!

   A belépéshez szükséges adatok: meres / LaborImage

2. Nyissuk meg a tálcán lévő PowerShell konzolt!

3. Ennek hatására megnyílik a PowerShell konzol.



4. Kérdezzük le az elérhető parancsokat!

   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.

   ```
   Get-Command
   ```

5. Adjunk át valami paramétert a fenti cmdletnek!

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

   ```
   Get-Command -Verb get
   ```

   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 tipográfiailag megfelelőbbre (pl. " helyett ” vagy - helyett –). Ezeket a PowerShell nem szereti, és nehéz is észrevenni a különbséget elsőre.

[6] PowerGui, http://www.powergui.org/

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

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

   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.

7. 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!

8. Parancstörténet használata

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

   A `FEL` billentyű segítségével lépkedjünk vissza arra a parancsra, amikor a *Get* igéhez 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:

```
# tomb
$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
"bbbbb" | out-file b.txt                    # lehet így is
$c = "cccccc"
Out-File -FilePath c.txt -InputObject $c    # vagy így
Get-ChildItem                               # ez visszaad egy gyűjteményt
```

Válasszunk 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)"}
# ugyenez 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
```

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 jegyzettömböt, és másoljuk bele a következő kódot. Az eredményt mentsük el `Out-Hello.ps1` néven. Figyeljünk a .ps1 kiterjesztésre!

```
# Name:   Out-Hello.ps1
# Author: Micskei Zoltán
# Date:   2011.03.01.
# Desc:   Example powershell script
# Param:      $hello - string to write out

param(
 [string] $Hello = $(throw "Supply the string!")
)

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

Hajtsuk végre a létrehozott szkriptet:

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

3. PowerGui használata
   - Nyissuk meg az előbb elkészített szkriptünk a *PowerGui Script Editor* programban! A program megfelelően színezi a forráskódot.
   - Helyezzünk el egy töréspontot a kiíratás sorra!
   - Az „Input script parameters here” ablakba írjuk be az előbb is átadott paramétereket!
   - Indítsuk el debug módban (`F5`)! A PowerGui jelzi, hogy jelenleg hol tart a végrehajtás.
   - A *Variables* ablakban nézzük meg a *$Hello* változó értékét.
4. Szkript készítése önállóan

   Készítsük el a következő paraméterekkel rendelkező szkriptet:

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

   A szkript lekérdezi a futó folyamatokat, és visszaadja a Name paraméterrel megegyező nevűek közül azokat, amiknek a fizikai memóriafoglalása (working set) nagyobb, mint a Size paraméter megabájtban értve. A szkript ellenőrizze a bemeneti paramétereket, a Name paraméter kötelező, a Size opcionális.

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 Bash 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/index.html

[2] Szandi Lajos, Shell programozás, BME HIT, http://www.hit.bme.hu/~szandi/unix/

[3] Linuxconfig, Bash scripting Tutorial, http://www.linuxconfig.org/Bash_scripting_Tutorial

[4] Mendel Cooper, Advanced Bash-Scripting Guide, http://www.tldp.org/LDP/abs/html/

Windows

[5] TechNetKlub, Short Online Trainings (SHOT), PowerShell, http://technetklub.hu/shot/#5

[6] Soós Tibor, Microsoft PowerShell 2.0 rendszergazdáknak – elmélet és gyakorlat, 2010, http://download.microsoft.com/download/5/8/8/5886EEB0-BFB5-4854-9D17-AAEDE66825D3/konyvek/PowerShell_v2/Microsoft_Powershell_2_konyv.pdf