# 8. Tétel

Linux parancsértelmezők (shell) milyen szolgáltatásokat nyújtanak?

## Shell – parancsértelmező

* ugyanolyan app, mint a többi
* lecserélhető (chsh)
* default: bash (Bourne again shell / Born again shell)

## Parancssor értelmezés

* formátum: *parancs arg1 arg2 arg3*
* parancs értelmezése: PATH-on levő könyvtárakban keresi a shell parancsokat 🡪 végrehajtja
* argumentumok:
  + egyszerű karakterlánc
  + <állománynév>
  + karakterlánc helyettesítő karakterekkel
  + ’parancshelettesítés’
* a shell a command line-ról string tömböt kap

## A shell beépített parancsai:

### Mini-shellek:

* Minimális felület, ami tartalmazza az alap shell parancsokat
* Azok vannak implementálva, amik probléma esetén is biztosan működnek
* pl.: shash (Stand-Alone Shell)

### Állománynév helyettesítés

* \* - nulla vagy tetszőleges számú tetszőleges karakter
* ? – pontosan egy tetszőleges karakter
* [abc] – a, b vagy c karakterek egyike
* [m-n] – a megadott intervallumból egy karakter

### Stdin, stdout átirányítás

Egy shell parancsnak 3 kimenete van:

* standard input
* standard output
* error output

### Átirányítás:

* parancs > kimenet – létrehozza a kimenet állomány, és beleírja a parancs kimenetét (vagy ha már létezik, felülcsapja)
* parancs >> kimenet – append
* parancs 2> kimenet – stderr kimenet belepakolása a kimenetbe
* parancs < input – az inputot a parancs nem a command line-ról várja, hanem az input fájlból

### Csővezeték (Pipe)

* ls | grep „minta” | sort | more
  + ls-ből kiszűri a grep a mintát, sort sorbarendezi, more oldalakra darabolja
* nem kell ideiglenes fájl, ahova az egyik ír, másik olvas
* PIPE gyors, mivel csak a memóriát használja
* az eredmény a cső végén már azelőtt megjelenhez, mielőtt az első program végzett volna

### Parancshelyettesítés

* paraméter, vagy parancs fájlban is lehet
* du $(cat parameters.txt) vagy du ’cat paramterts.txt’
  + megmondja a fájlok méretét, amit előre eltároltunk a parameters.txt-ben

### Parancssorozatok

* date; ls – egymás tán hajtódik végre
* parancs1 || parancs2 – felt. parancs: a parancs2 akkor hajtódik végre, ha parancs1 hibával állt meg (vagy az első, vagy a második)
* parancs1 && parancs2 – felt. parancs: parancs4 akkor hajtódik végre, ha parancs3 igaz megállási státusszal állt meg (első és második)

### Szinkron és aszinkron folyamatok

* parancs &
* a shell azonnal visszaadja a promtot, mielőtt befejezte volna a parancs a végrehajtást
* ***note***: a stdoutot át érdemes irányítani fájlba, különben a consolra pakolja ki az eredményt, ami zavaró lehet

### Csoportosítás

zárójellel

## Változók kezelése

kutya = ugat

echo $kutya

ugat

vagy:

kati = zsuzsi

echo ${kati}ka

zsuszika

* olyan mint a #define: pontosan azt tárolja, amit beleírtunk (szám, karaktersor, rövid fv.)
* unset parancs: változó törlése
* set parancs: kiírja az összes definiált változót
* környezeti változók nem elérhetőek kapásból a programok számára 🡪 ki kell exportálni a változók közé a kívánt környezeti változót, és azt már látni fogja 🡪 export parancs

## Aritmetikai kiértékelés

a=$((5\*2))

a=$((a\*3))

## Parancsállományok – shell scriptek – .sh

* interaktív parancsértelmezés
* futtatás:
  + futási jog nélkül: bash shell\_script.sh argumentumok
  + futási joggal: bash shell\_script.sh argumentumok

### Feltételek kiértékelése

test parancs

* stringek vizsgálata
  + test s : true, if s string is not null
  + test –z s: true, is s string’s length is not zero
  + = ; !=
* numerikus vizsgálat
  + -eq, -ne: equal, not equal
  + –lt, -le, -gt, -ge: lesss than, less or equal, …
* állományok tulajdonságainak ellenőrzése
  + –f f: f egy létező állomány/könyvtár?
  + –r f, -w f, -d f: f is readable, writable, directory?
  + –s f: f létezik, és nem nulla hosszúságú
  + –t fd: ha fd egy megnyitott állomány leírója és az egyben egy terminál
* test elhagyható, ha a feltételt [] közé írjuk
* ha a vizsgált paraméter space-t tartalmaz: {}
* shell tördelésének és állománynév értelmezésének letiltása: [[adf]]

### Vezérlési szerkezetek:

|  |  |  |  |
| --- | --- | --- | --- |
| if [$”1” eq 1]  then echo ’egy’  else echo ’nem egy’  fi | for i in w1 w2  do parancsok  done  for ((i=1; i<=5; i++))  do echo $”i”  done | | while parancs  do parancsok  done  while ((i<=5))  do i=$((i+1))  done |
| until test $”i" –le 5  do I = ‘expr $”i" + 1  done  until ((i>5))  do i=$((i+1))  done | | case $1 in  echo “egy”;;  echo “ketto”;;  \*) echo “sok”  esac | |

Shell scipt:

#!/bin/bash

# file számláló

n=0

for i in \*

do

if [-f $i]

then n=’expr $n + 1

fir

done

## Folyamatok monitorozása: ps

* Attribútumok:
  + -A: összes folyamat
  + -N: negálja a folyamatválasztást
  + a: terminálhoz kapcsolódó összes folyamat (shellek kivételével)
  + r: csak a futó folyamatok
  + -u user: userhez tartozó folyamatok
* formzás (kötőjellel és anélkül is ok)
  + –f: teljes lista
  + –j: job control info
  + –l: hosszú fájlformátum
  + –o formátum: saját definiált formátum

### Hosszú folyamatok: nohup parancs &

* megmondja, hogy ne vegye figyelembe, ha a felhasználó kijelentkezik
* semmiképpen sem az stdout-ot használja (ha nem irányítjuk kimenetbe, akkor nohup.out)

### Kommunikáció a folyamatokkal, megszüntetés

* kill [-szignál] folyamatszál
* szignál:
  + SIGHUP(1) – akkor generálódik, ha kilépünk, miközben a folyamat még fut
    - def: akkor kapja a folyamat, ha a szülője leállt
  + SIGINT(2) – interrupt karakter (Ctrl + C)
    - def: ctrl+c
  + SIGQUIT(3) – legerősebb szignál – ezt nem hagyhatja figyelmen kívül a program (nem javasolt)
    - def: ctrl + \
  + SIGTERM(15) – Folyamatok leállítása. Alapértelmezett érték

### Folyamat vezérlése

* Ctrl+z: folyamat felfüggesztése (Pause) 🡪 dob egy job számot, amivel hivatkozhatunk rá később
* bg <job>: folytatódjon háttér folyamatként
* fg <job>: foreground run
* jobs: felfüggesztett folyamatok listázása
* kill [-szignál] %job

### Prioritás állítás

* renice <prioritás> <processid>
* renice <prioritás> -p pid –g pgrp –u user
  + –p pid: processidvel azonosítás
  + –g pgrp: egy folyamat csoportra hivatkozás
  + –u user: felhasználó összes folyamata manipulálása
* prioritás szintek: 0-20
  + 0: default prioritás 🡪 a pozitív egyre előzékenyebb 🡪 a 20-as csak akkor fut, ha más folyamatnak már nincs szüksége a processzorra
* root állíthat -20-ig – fontos processzektől szipkázhatja el az erőforrást