

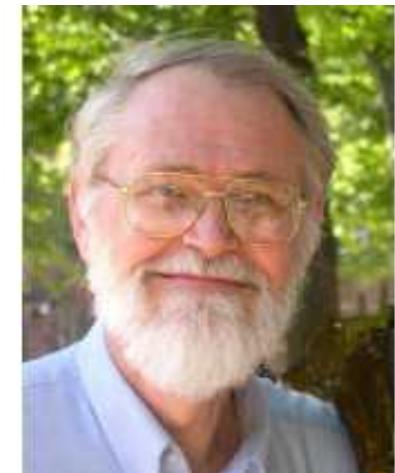
# Rendszerprogramozás

Dr. Iszály György Barna

# Unix



*Thompson és Ritchie*



*Brian Kernighan*

- ▶ **Ken Thompson és Brian Kernighan** 1969 AT&T vállalat Bell laboratóriumában készíti el
- ▶ Az első magas szintű programozási nyelven készített operációs rendszer volt (nem egy konkrét gép assembly nyelvén), így lehetővé vált a rendszer egyszerű portolása más architektúrákra.
- ▶ A felhasznált magas szintű nyelv a C nyelv volt és ehhez a projekthez hozta létre **Dennis Ritchie**
- ▶ A trösztellenes törvény értelmében az AT&T nem forgalmazhatott szoftvereket, ezért a nyelvet és az operációs rendszert **ingyenessé** tették.

# GNU



*Richard Stallman*

- ▶ Az AT&T-t felosztották kisebb vállalatokra, amelyek már értékesíthettek szoftvert, így zárttá tették a UNIX-ot.
- ▶ 1983 Richard Matthew Stallman (rms) szabad szoftver mozgalom – GNU (*GNU is Not Unix*)
- ▶ Ennek keretében megalakult a **Free Software Foundation (FSF)** és elkezdték kifejleszteni egy ingyenes környezet programcsomagjait. (GCC (GNU Compiler Collection)).

# Linux

- ▶ 1991 **Linus Torvalds** a helsinki egyetem diákja – egy UNIX-klónt készített az IBM 386-os gépére és megosztotta az interneten
- ▶ A nevet barátja, **Ari Lemmke** adta, így nevezte el Linus FTP könyvtárát.
- ▶ A Linux tulajdonképpen magát a kernelt és egy sor ahhoz közeli (részben FSF – fejlesztésű) programot jelent.



*Linus Torvalds*

# Linux felépítése

- ▶ A Linux **monolitikus kernellel** rendelkezik, amelynek feladatai:
  - folyamatkezelés (process),
  - fájlrendszer kezelése,
  - perifériák elérése,
  - hálózatkezelés.
- ▶ Az eszközkezelők (device driver) beépülhetnek a kernelbe, de modulárisan is kapcsolódhatnak ahhoz.
- ▶ A kernel számozásának alakja: A.B.C(.D):
  - A: főverzió – kernel version
  - B: fő átdolgozás – major revision
  - C: kisebb átdolgozás – minor revision
  - D: hibajavítás, biztonsági frissítés – bug-fixes, security patches
- ▶ A kernelt Linus felügyeletével folyamatosan fejlesztik – legutolsó stabil verzió 4.4.2, 2016. február 17.  
[\(https://www.kernel.org/\)](https://www.kernel.org/)

# Kernel feletti komponensek

- ▶ A kernel fölötti alacsony szintű komponensek a rendszer legalapvetőbb funkcióit látják el:
  - **bootloader**: az első program, ami a gép bekapcsolásakor betöltődik, ez végzi a kernel betöltését a memóriába (pl. GRUB, LILO),
  - **init**: a legelső folyamat (process), amit a kernel indít el – ez indítja a rendszer szolgáltatásait és minden további folyamat rajta keresztül indul el,
  - **dinamikus könyvtárak**: a Linux lehetőséget ad dinamikusan csatolt könyvtárak használatára (SO – shared object), amelyek formátuma: ELF (Executable and Linkable Format),
  - **különböző felhasználói felületek**: léteznek szöveges és grafikus felületek

# X11

- ▶ A Linuxos grafikus felületek alapja az X Window System, jelenlegi verziószáma 11, ezért X11-nek is hívnak.
- ▶ MIT-n kezdték el fejleszteni 1984-ben
- ▶ Számítógép-hálózatokon működtetett, távoli grafikus felületek létrehozását biztosítja architektúra-független módon.
- ▶ Alacsony szintű hálózati protokollja segítségével egy keretrendszer biztosít grafikus programok létrehozására.
- ▶ Általában az X-re épülő, komplexebb grafikus könyvtárakat (Desktop Environment) használnak (GNOME, KDE, XFCE, stb)

# Disztribúciók

- ▶ Disztribúciók (distribution) általános felépítése a következő:
  - friss Linux kernel,
  - GNU programok,
  - ablakozó környezet(ek) (GNOME, KDE, stb.),
  - egyéb ingyenes programok,
  - csomagkezelő.
- ▶ Ismertebb disztribúció:
  - Ubuntu,
  - Suse,
  - Debian,
  - Red Hat,
  - Fedora,
  - CentOS,
  - Gentoo,
  - UHU-linux (magyar).

# Csomagkezelő

- ▶ A programok kezelésének egy központosított módja a csomagkezelő (package manager) használata.
- ▶ Egy csomagkezelő:
  - képes egy program függőségeit (a futtatáshoz szükséges könyvtárak, egyéb programok) nyilvántartani,
  - elvégzi egy program telepítését az azt tartalmazó csomagból,
  - képes a hiányzó függőségek automatikus telepítésére vagy az azokat tartalmazó csomagok listázására,
  - automatikusan kezeli a csomagok frissítéseit,
  - képes a rendszer karbantartására – pl. nem használt, elavult csomagok eltávolítására.

# Csomagkezelő

- ▶ Internethető szabadon hozzáférhető adatbázisokkal dolgoznak.
- ▶ Leegyszerűsítik a programok telepítését – egy parancs, vagy egy klikkelés
- ▶ Pl.  
`sudo apt-get install build-essential`  
fejlesztő-programokat telepít fel(GNU C, C++ compiler, stb.).
- ▶ Előnyei:
  - nem kell a függőségeket „kézzel” megkeresni és telepíteni,
  - automatikus, egységes és biztonságos frissítési mechanizmus
  - a feltelepített csomagok lista elmentésével egyszerűen és automatikusan reprodukálható egy működő rendszer.

# Linux elérési utak

- ▶ Kis- és nagybetű érzékeny!!!!
- ▶ A teljes fájl-szerkezet egy fa.
- ▶ **Nincsenek meghajtók**, mint Windowsban – a fizikai meghajtók a fába vannak „felcsatolva” (**mount**).
- ▶ A fa gyökere a gyökérkönyvtár, amelynek neve: /
- ▶ A szeparátor jel is a / jel  
pl. /usr/include/c++/4.6.3/
- ▶ **Abszolút elérési útvonal**: a gyökértől elindulva megadja egy könyvtár/fájl teljes elérését.
- ▶ **Relatív elérési útvonal**: az aktuális könyvtártól kiindulva adja meg az elérést (nem kezdődhet / jellet).
- ▶ Speciális könyvtárnevek:
  - . - a mindenkor aktuális könyvtár
  - .. - az aktuális könyvtár szülőkönyvtára

# Hozzáférési jogosultságok

- ▶ Az egyes fájlokhoz való hozzáférés szabályozott módon történik
- ▶ Fájlokkal végezhető műveletek:
  - olvasás (r),
  - írás (w),
  - futtatás (x).
- ▶ Az egyes műveletkhez való jog aszerint definiált, hogy „mennyire állunk közel a fájlhoz” – a Linux három csoportba osztja a felhasználókat az egyes fájlokhoz való hozzáférés szerint:
  - tulajdonos (u),
  - tulajdonoscsoport (g),
  - más (o).
- ▶ A tulajdonoscsoportok létrehozása leegyszerűsíti a felhasználók és a nekik adott jogosultságok adminisztrálását egy nagy rendszerben.

# Hozzáférési jogosultságok

- ▶ Könyvtárak esetén a hozzáférési jogok:
  - olvasás: a benne található állományok listázása,
  - írás: fájlok létrehozása, létezo fájlok törlése,
  - futtatás: a könyvtárba való belépés
- ▶ Ha például egy könyvtárra van írási jogunk, de nincs olvasási, akkor létrehozhatunk ott fájlokat, vagy törölhetünk a már létezők közül anélkül, hogy ki tudnánk listázni a könyvtár tartalmát.

# Hozzáférési jogosultságok

- ▶ Egy fájl jogosultsági táblázata egy kilencbites érték:

u			g			o		
r	w	x	r	w	x	r	w	x
0	1	2	3	4	5	6	7	8
$2^2$	$2^1$	$2^0$	$2^2$	$2^1$	$2^0$	$2^2$	$2^1$	$2^0$

- ▶ Hárrom oktális számjeggyel szokták jellemezni a hozzáférési jogokat.
- ▶ Pl. a 764 azt jelenti, hogy
  - a tulajdonos olvashat, írhat, futtathat,
  - a tulajdonoscsoporth többi tagjai olvashatnak, írhatnak, de nem futtathatnak,
  - a többi felhasználó pedig csak olvashat.

# Hozzáférési jogosultságok módosítása

- ▶ chmod paranccsal  
**chmod 774 valami.txt**
- ▶ Könnyebben kezelhető, amikor jogokat tudunk hozzáadni, illetve elvenni az egyes csoportok és jogok betűjelei segítségével:  
**chmod u+x valami.txt**  
a valami.sh futtatását engedélyezzük a tulajdonosának  
**chmod g=rw valami.txt**  
utasítással a tulajdonoscsoporttól elvettük az olvasási és írási jogokat  
**chmod u+r-w-x,g+r+w-x,o-r+w-x valami.txt**  
 minden adat megadása felhasználócsoporthoz külön-külön  
**chmod +x valami.txt**  
összes felhasználó jogainak megadása egyszerre  
**chmod u=rw valami.sh**  
egy felhasználó jogainak megadása abszolút módon (nem inkrementálisan):

# Felhasználók

- ▶ A felhasználókat egy felhasználónév és egy jelszó azonosítja.
- ▶ A jelszó a **passwd** utasítással változtatható meg.
- ▶ minden felhasználónak van egy saját home könyvtára ami a /home/ könyvtárban található 8/home/hallgato/)
- ▶ Saját könyvtár rövid neve: ~
- ▶ Másik felhasználó home könyvtára:  
~felhasználoinev

# Adminisztrátor

- ▶ A rendszer telepítését és karbantartását végzik.
- ▶ Csak nekik van joguk:
  - a rendszer működését érintő beállítások módosításához,
  - a rendszer belső leírófájljainak írásához,
  - programok telepítéséhez,
  - felhasználók adatainak kezeléséhez.
- ▶ Az adminisztrátor neve **super user** vagy **root**.
- ▶ A normál felhasználóknak is lehetnek adminisztrátor kiváltságai, ilyenkor a parancsok elő oda kell írni, hogy sudo (super user do):
  - sudo apt-get install build-essential
  - majd meg kell adniuk a jelszavukat.
- ▶ Linux alatt egy programnak ugyanazok a jogosultságai, mint annak a felhasználónak, aki elindította.

# Parancsok – állománykezelés

- ▶ Parancsok általános alakja  
parancs kapcsolók paraméterek
- ▶ **pwd** – kiírja az aktuális könyvtár abszolút elérési útját
- ▶ **cd <könyvtár>** – a <könyvtár> lesz az aktuális könyvtár Paraméter nélkül a home könyvtárba ugrik
- ▶ **ls <kapcsolók> <lista>** – kilistázza a listában megadott fájlokat és könyvtárakat (ha nincs lista, akkor olyan mintha . lett volna megadva). Kapcsolók:
  - –l részletes lista
  - –a rejtett fájlokat is kiírja
  - –d a könyvtárakat ugyanúgy mint a fájlokat (nem a tartalmukat listázza ki)
  - –R a könyvtárakat rekurzívan
  - –r csökkenő betűrendben listáz

# Parancsok – állománykezelés

- ▶ **mkdir <kapcsolók> <lista>** – létrehozza a listában található könyvtárakat. Kapcsolók:
  - –p a teljes megadott útvonalat létrehozza
  - –m a könyvtár jogosultságainak megadása (oktális alak) –m <jog>
- ▶ **rmdir <kapcsolók> <lista>** – törli a listában megadott könyvtárakat. Kapcsolók:
  - –p a teljes megadott útvonalat törli
- ▶ **mv <kapcsolók> <eredeti> <új>** – Átnevezi az <eredeti> nevű fájlt vagy könyvtárat <új> névre (ha nem létezik ilyen könyvtár). Kapcsolók:
  - –b ha az új létezik, készít róla egy biztonsági mentést (<új>~ néven)
- ▶ **mv <kapcsolók> <lista> <újhely>** – átmozgatja <lista> elemeit az <újhely> könyvtárba (aminek léteznie kell) Kapcsolók:
  - –f kétes esetben nem kérdez
  - –i bármely kétes esetben megerősítést vár
  - –u csak a régebbi fájlokat írja felül

# Parancsok – állománykezelés

- ▶ **cp <kapcsolók> <eredeti> <új>** – létrehoz egy <eredeti> nevű fájlból vagy könyvtárból egy másolatot <új> néven (ha nem létezik ilyen nevű könyvtár)
  - **-b** ha az új létezik, készít róla egy biztonsági mentést (<új>~ néven)
- ▶ **cp <kapcsolók> <lista> <újhely>** – átmozgatja a <lista> elemeit az <újhely> könyvtárba (aminek léteznie kell)
  - **-f** kétes esetben nem kérdez
  - **-i** bármely kétes esetben megerősítést vár
  - **-r -R** könyvtárakat rekurzívan átmásolja
  - **-l** másolás helyett linket készít
  - **-s** másolás helyett szimbolikus linket
  - **-u** csak akkor másol, ha a cél helyen régebbi fájlok vannak, vagy az adott fájl hiányzik

# Parancsok – állománykezelés

- ▶ **rm <kapcsolók> <lista>** – Törli a listában megadott fájlokat
  - -f kétes esetben sem kérdez vissza
  - -i bármely kétes esetben megerősítést vár
  - -r -R ha könyvtárnevet talál a listában, törli azt
- ▶ **du <kapcsolók> <lista>** – összeszámolja a könyvtárban található fájlok méretét
  - -a a kimeneti listába a fájlok is bekerülnek
  - -s a méreteket összeadja
  - -m kilobyte-ok helyett megabyte-okban számol
  - -h az ember számára olvasható mértékegységek
- ▶ **quota** – A felhasználó által használható lemezterület méretét írja ki.
- ▶ **df** – A fájlrendszer lemezhasználatáról készít jelentést.
  - -h az ember számára olvasható mértékegységek

# Parancsok – szöveges fájlok

- ▶ **cat <fájl>** – a <fájl> teljes tartalmát egyszerre kiírja a képernyőre
- ▶ **more <fájl>** – a <fájl> teljes tartalmát oldalanként kiírja a képernyőre
- ▶ **head -<n> <fájl>** – a <fájl> első <n> sorát kiírja a képernyőre
- ▶ **tail -<n> <fájl>** – a <fájl> utolsó <n> sorát kiírja a képernyőre
- ▶ **grep <minta> <fájl>** – a <fájl> azon sorait írja ki, amelyekben megtalálható a <minta> szövegrészlet
- ▶ **more <fájl>** – a fájl teljes tartalmát oldalanként írja ki a képernyőre
- ▶ **tee** – a bemenetként kapott szöveget a kimenetre írja.
- ▶ **less <fájl>** – a fájl tartalmát görgethetően írja ki
- ▶ **wc <kapcsolók> <fájl>** – a <fájl>-ban található byte-ok/sorok/szavak számát írja ki
  - **-c** a fájl méretét írja ki
  - **-w** a szavak számát írja ki
  - **-l** a sorok számát írja ki
  - **-m** a karakterek számát írja ki
- ▶ **sort <kapcsolók> <fájl>** – a fájl sorait ábécé sorrendben írja ki.
  - **-r** csökken sorrend (z-a)

# Parancsok – jogosultság és egyebek

- ▶ **man <parancs>** – Előhozza a parancs részletes leírását
- ▶ **file <fájl>** – Megadja a fájl típusát.
- ▶ **echo <szöveg>** – Kiírja a szöveget.
- ▶ **passwd** – A jelszó megváltoztatására szolgál.
  
- ▶ **Jogosultságok**
  - Tulajdonos (User), Csoport (Group), Többiek (Others)
  - Olvasás (Read), Írás (Write), Végrehajtás (eXecution)
- ▶ **chmod <kapcsolók> <jogosultság> <lista>**
- ▶ **Jogosultság** – <kinek><hogyan><mit>
  - <kinek> a (all), u (user), g (group), o (others)
  - <hogyan> + (megadás), – (megvonás), = (beállítás)
  - <mit> r (read), w (write), x, (execute)
- ▶ **<user><group><other>**
  - read: 4, write: 2, execute: 1 – ezek összege a kód
- ▶ **Kapcsolók**
  - –R rekurzívan beállítja a jogokat a könyvtárban található fájlokra és könyvtárakra

# Parancsok – linkek, mintaillesztés

- ▶ **ln -s <forrás> <cél>** – létrehoz egy a <forrás> -ra mutató soft linket (szimbólikus link)
- ▶ **ln <forrás> <cél>** – Létrehoz egy a <forrás> -ra mutató hard linket (könyvtárra nem alkalmazható)
  
- ▶ ? egyetlen karaktert helyettesít
- ▶ \* akárhány karaktert helyettesít
- ▶ [...] a zárójelek között felsorolt karaktereket helyettesítheti
- ▶ \c a c speciális karaktert jelent (\, szóköz, ', ", ?, \*, [, ], `)

# Parancsok – felhasználók kezelése

- ▶ **finger** – Megadja, hogy ezen a gépen kik vannak bejelentkezve.
- ▶ **finger <név>** – Felsorolja azokat a felhasználókat akiknek a neve vagy azonosítója megegyezik a <név> paraméterrel.
- ▶ **finger @host** – Megadja, hogy a host gépen kik vannak bejelentkezve.
- ▶ **who** – Megadja, hogy ki van bejelentkezve a gépre. A fingernél kevesebb adatot szolgáltat.
- ▶ **w** – who, de megmondja azt is, hogy ki min dolgozik.
- ▶ **last** – az utolsó néhány bejelentkezett felhasználót jeleníti meg

# Parancsok – multitasking

- ▶ **ps** – Kílistázza a processeket.
  - -e minden process kiírása
  - -f minden információt megjelenít
  - -u <user> csak az adott felhasználó processeit jeleníti meg
- ▶ **jobs** – job-ok lekérdezése.
- ▶ **fg %n** – Ha n egy leállított job száma, az előtérben újraindítja.
- ▶ **bg %n** – Ha n egy leállított job száma, a háttérben újraindítja.
- ▶ **kill n** – Ha n egy job száma, leállítja a jobot.
  - -s signal küldése a leállításról (*/usr/include/linux/signal.h*)
  - -l az összes szignál kiírása (több mint 60)
  - & a program indítása a háttérben
  - ^C megszakítja a program futását (CTRL+C)
  - ^Z megállítja a program futását, de később újraindítható (CTRL+Z)
- ▶ **killall <név>** – Leállítja az összes <név> nevű processzt.

# Parancsok – shell

- ▶ **/dev/null** – egy olyan fájl, ami minden beleírt adatot „elnyel”.
- ▶ **>** – alapértelmezett kimenet átirányítás
- ▶ **<** – alapértelmezett bemenet átirányítás
- ▶ **|** – a program kimenetét a következő program bemenetére állítja
- ▶ **;** – **program1; program2; program3;** a programok egymás után futnak le (a megadott sorrendben, eredménytől függetlenül)
- ▶ **&& program1 && program2** – a program2 csak akkor indul el, ha az első sikerült
- ▶ **|| program1 || program2** – a program2 csak akkor indul el, ha az első nem sikerült
- ▶ **~** – home könyvtár
- ▶ **\$PWD** – az aktuális könyvtár abszolút útvonalát tárolja (pwd)
- ▶ **\$HOME** – a home könyvtár abszolút útvonalát tárolja
- ▶ **\$PS1** – a prompt kinézetét írja le
- ▶ **\$PATH** – keresési útvonal, a shell az itt felsorolt könyvtárakban keresi a futtatandó programokat
- ▶ **export** – Globális környezeti változó megadása.
- ▶ **set** – kiírja a beállított környezeti változókat
- ▶ **unset <változónév>** – kitörli a <változónév> nevű környezeti változót

# Programok futtatása

- ▶ Futtatási jogunknak kell lennie a fájlon
- ▶ Ha beírjuk egy program nevét, akkor a Linux a \$PATH nevű környezeti változóban felsorolt útvonalakon keresi a programot, ahol az egyes elérési utak kettősponttal vannak elválasztva  
pl.: /usr/local/sbin:/usr/local/bin:/usr/bin:/bin
- ▶ PATH beállítása

```
PATH=$PATH:uj_konyvtar
export PATH
```

Vagy rövidebben:  

```
export PATH=$PATH:uj_konyvtar
```
- ▶ Ha az aktuális könyvtár nincs benne ebben a felsorolásban, akkor az fájl teljes elérési útvonalával kell megadni a nevet:  
/home/nagger/bin/valami.sh
- ▶ Relatív elérési úttal is futtathatjuk  
./amoba

# Shell – burok, héj

- ▶ Elrejt valamit (a felhasználó elől)
- ▶ A burok mindig valamely célközönség számára készül
- ▶ A fülhöz emelve a tenger zúgását halljuk (kommunikáció a felhasználóval)
- ▶ Van díszesebb és kevésbé díszes

# Shell – burok, héj

- ▶ A shell vagy héj tulajdonképpen egy parancsértelmező, a felhasználó által parancssori szöveggel leírt parancsokat hajtja végre, így interfészt biztosít a felhasználó és az operációs rendszer között.
- ▶ Ebben az értelemben parancsnak nevezünk bármilyen futtatható programot amelyik része egy Unix alapú operációs rendszernek.
- ▶ A parancsértelmező neve arra utal, hogy burokként veszi körül a UNIX operációs rendszer legfontosabb komponensét, a rendszer magját vagy kernelt, és ennek szolgáltatásaihoz csak a parancsértelmezőn keresztül lehet hozzáférni, amikor egy felhasználó használni akarja a rendszert.
- ▶ A héj legfontosabb feladatai: Unix parancsok indítása, összefűzése, felügyelete.

# Shellek

- ▶ Szabványos: sh, bash, ksh, csh, tcsh
- ▶ easy shell, pdmenu
- ▶ limited shell (dos szerű)
- ▶ router cli (cisco shell)
- ▶ screen
- ▶ stb, bárki írhat shell-t

# Shell tulajdonságai

- ▶ Hátrány: felesleges erőforrás használat, lassú kezelhetőség
- ▶ A shellekkel – bár sok gyakorlati tudással, de – gyorsabban érhetjük el céljainkat, és kényelmi funkciói is megjelentek

# Szabványos unix shell-ek

- ▶ Bourne (/bin/sh)
  - Legősibb (1977 Stephen Bourne, AT&T)
  - alapvető kellék, mentes minden kényelmi funkciótól => nem kellenek csicsás dolgok ha csak programokat akarunk elindítani. (pl. daemonok, scriptek, stb.)
- ▶ Korn (/bin/ksh) • zsh
  - Második legrégebbi (David Korn, AT&T)
  - 2000-ben nyílt forráskódúvá tették és jelenleg Common Public Licence licenc alatt fejlesztik (lásd <http://www.kornshell.com/>)
  - A sh továbbfejlesztése.
  - 1szerűbb változó megadás (1 lépében)
  - be és kikapcsolható opciók
  - helyettesítő nevek (aliasok) megjelenése
  - parancsnaplózó (history)
  - speciális változók
- ▶ C shell (/bin/csh) –
  - Bill Joy a Sun Microsystem alapítója
  - Szerkezetei C alapúak
- ▶ Bourne again shell (/bin/sh) – Bash
  - Nyílt forráskódú
  - GNU operációs rendszerek héja => Linux alapértelmezett héja
  - Leggazdagabb programozási lehetőséggel rendelkezik

# Shell tulajdonságai

- ▶ Parancsértelmező (interpreter) és programozási nyelv. A parancsokat a standard inputról, vagy állományból olvassa.
- ▶ Egyszerűen kialakítható az ún. shell script vagy parancs file, amivel a parancskészlet bővíthető.
- ▶ A shell scriptek paraméterezhetősége teljesen megegyezik a programok paraméterezhetőségével.

# Shell tulajdonságai

- ▶ Programnyelv, amely string változókra és hatékony vezérlési szerkezetekre épül.
- ▶ Egyszerű szintaxis a standard input–output átirányítására. (> < )
- ▶ Csővezeték ( | ) segítségével komplex feladatok megoldása a meglévő segédprogramokkal.

# Shell tulajdonságai

- ▶ Sok DOS-ban megszerzett ismeret jól hasznosítható, de vannak eltérések.
  - a dos nem többfelhasználós. ??? Miért nem másolták jól le a Unix shellt a dos készítői.
- ▶ Beépített parancsok (cd, umask, echo, stb.)
- ▶ Külső parancsok (kezdetben szinte minden külső parancs volt, kivéve ha nem lehetett máshogy megoldani „cd” )

# Shell tulajdonságai

- ▶ Lehetővé teszi a processzek áttekinthető kezelését.
- ▶ A rendszer része, de egy új shell elkészítése nem igényel semmilyen rendszerprivilégiumot, bármikor lecserélhető.
- ▶ Egyszerűen konfigurálható a felhasználó igényei szerint

# Shell beállítása

- ▶ Speciális változók segítségével:
  - \$HOME: login katalógus neve
  - \$PATH: keresési út a programok végrehajtásához
  - \$CDPATH: keresési út a cd parancshoz
  - \$PS1: elsődleges prompt
  - \$PS2: másodlagos prompt
  - \$editor: alapértelmezett szövegszerkesztő
  - Stb..

# Shell beállítása

- ▶ Több jól definiált ponton speciális parancsállományok futtatására van lehetőség:
  - login
  - logout
  - start

# Shell beállítása

<u>Shell</u>	<u>Login</u>	<u>Start</u>	<u>Logout</u>	<u>rsh</u>
sh	/etc/profile ~/.profile			
ksh	/etc/profile ~/.profile			
csh	/etc/csh.cshrc /etc/cshlogin ~/.cshrc -/.login	/etc/csh.cshrc ~/.cshrc	/etc/csh.logout ~/.logout	/etc/csh.cshrc ~/.cshrc
tcsh	/etc/csh.cshrc /etc/cshlogin ~/.tcshrc    ~/.cshrc ~/.login	/etc/csh.cshrc ~/.tcshrc ~/.cshrc	/etc/csh.logout ~/.logout	/etc/csh.cshrc ~/.cshrc
bash	/etc/profile ~/.bash_profile ~/.bash_login ~/.profile	~/.bashrc	~/.bash_logout	

# Kényelmi funkciók

- ▶ aliasok (alias ..=„cd ..”)
- ▶ history
- ▶ állománynév kiegészítés (tab)
- ▶ terminálkezelés (jogosultságok, átirányíthatóság, felfüggeszthetőség, msg...)
- ▶ Egymásba ágyazhatóság
- ▶ Online manual (man, apropos)

# Terminál - parancssor kezelés

- ▶ Intr, Ctrl-C – a megszakító billentyű, a futó programot szakítja meg
- ▶ Quit, Ctrl-\ – szintén megállítja a futó programot, az un. QUIT jelzéssel
- ▶ Eof, Ctrl-D – fájl vége jel
- ▶ Erase, Ctrl-H vagy BS – a törlésre használt karakter
- ▶ Werase, Qrl-W – egy szót töröl (word erase)
- ▶ Kill, Ctrl-U – egy teljes parancssort töröl
- ▶ Suspend, Ctrl-Z – felfüggeszti az éppen futó folyamatot (a folyamat ideiglenesen leáll)
- ▶ Stop, Ctrl-S – megállítja a terminálra való írást
- ▶ Start, Ctrl-Q – újraindítja az írást
- ▶ Clear, Ctrl-L – törli a terminál képernyőjét, vagy más alkalmazások esetében újraraajzolja a képernyőt

# Terminál - parancssor kezelés

- ▶ Ctrl-B – back – balra lép egy karaktert
- ▶ Ctrl-F – forward – jobbra egy karaktert
- ▶ DEL vagy BS – a kurzor bal oldalán levő karaktert törli
- ▶ Ctrl-A – a sor elejére ugrik
- ▶ Ctrl-E – end – a sor végére ugrik
- ▶ Alt-F – egy szót ugrik előre
- ▶ Alt-B – egy szót ugrik vissza
- ▶ Ctrl-U – törli a teljes sor
- ▶ Ctrl-K – a kurzortól a sor végéig töröl előre
- ▶ Ctrl-W – a kurzortól visszafelé töröl az első szóközig (gyakorlatilag egy szót töröl)
- ▶ Ctrl-Y – visszamásol (yank): a törölt szöveget visszailleszti a kurzor pozíciójától (kivéve a teljes sor törlését)
- ▶ Ctrl-T – kicseréli a kurzor alatti és az előtte levő karaktereket (ha rossz sorrendben gépeltük őket)

# Unix/Linux fájlrendszer

- ▶ Hierarchikus
- ▶ Kis és nagybetű érzékeny
- ▶ A könyvtár is egy fájl
- ▶ „..”
- ▶ Home könyvtár – /home/nevemsenki
- ▶ Munkakönyvtár – a futó program könyvtára
- ▶ A fájl elnevezési konvenciók a UNIX alatt nem kötelezőek, bármilyen nevet és kiterjesztést adhatunk a különböző tartalmú fájloknak

# Fontosabb könyvtárak

- ▶ /boot – A rendszer indításához szükséges fájlokat tartalmazza, itt van a kernelt tartalmazó fájl is.
- ▶ /home – A felhasználók saját könyvtárai, az un. "home" könyvtárak vannak itt.  
Például: /home/nevemsenki
- ▶ /dev – Az eszközökre mutató fájlok a /dev könyvtárban találhatóak.
- ▶ /bin – Bináris, futtatható fájlok helye. Itt vannak a parancsok.
- ▶ /sbin – Csak a superuser (root) által végrehajtható fájlok.
- ▶ /usr – A felhasználók és rendszer által használt fontos fájlok. A /usr alatt találhatóak például: /usr/bin futtatható fájlok, /usr/src forrás fájlok, /usr/lib könyvtárak.
- ▶ /tmp – mindenki által írható ideiglenes fájlok helye.
- ▶ /var – Terjeszkedő, rendszer által írt fájlok (pl. naplózó fájlok).
- ▶ /proc – egy kernel által létrehozott virtuális fájlrendszer amely a futó folyamatokat és egyéb rendszerparamétereket írja le.

# Hozzáférési jogok

- ▶ A felhasználókat a rendszer a felhasználó neve és jelszó szerint azonosítja, a rendszerben viszont egész számokkal tartja nyilván (user id vagy uid, group id vagy gid).
- ▶ id parancssal listáztathatóak ki ezek
- ▶ kétféle tulajdonjog létezik a fájlokon:
  - felhasználói (user)
  - csoport (group).
- ▶ A jogosultságok a chmod parancssal változtathatóak, a fájlhoz rendelt (tulajdonos) felhasználó és csoport pedig a chown és chgrp parancsokkal.

# Hozzáférési jogok

- ▶ A chmod parancs négy csoportra osztja a felhasználókat:
  - ▶ a felhasználó (user) jogai, szimbóluma: u
  - ▶ a csoport (group) jogai, szimbóluma: g
  - ▶ a többiek (others) jogai, szimbóluma o
  - ▶ mindenki jogai (all), szimbólum: a
- ▶ Jogok jelölései állományokon:
  - rwxrwxrwx – user group others (4-2-1)
- ▶ Könyvtárak esetében:
  - read – olvasható a könyvtár tartalomjegyzéke
  - write – létrehozható benne új állomány
  - execute – be lehet lépni a könyvtárba és írhatja olvashatja az állományokat jogosultság megléte esetén

# Időbélyegek

- ▶ minden fájl 3 időbélyeget tartalmaz
  - az utolsó hozzáférés ideje: atime (access time),  
ls -ltu
  - az utolsó módosítás ideje: mtime (modification time)  
ls -lt
  - az utolsó állapotváltozás ideje (a tárolt tulajdonságok változása, jogok, tulajdonos, linkek száma): ctime (status change time)  
ls -lct

# Parancs szintaktika

*parancs kapcsolók argumentumok ....*

- ▶ kapcsolók: a parancs működését, eredményét befolyásoló paraméterek. Többnyire „-” kezdődnek (pl.: ls -al, ps -a)
- ▶ Argumentumok: filenév, vagy másik program bemenete
- ▶ Egy parancs több parancsot is tartalmazhat:
  - prog &
  - prog1; prog2
  - prog1&& prog2
  - prog1|| prog2
- ▶ Egy parancs több sorból is állhat. (folytató sor „\” )
- ▶ Lehetővé teszi, hogy az egyik program bemenete a másik kimenete legyen. (csővezeték, átirányítás)

# File kezelési parancsok

- ▶ cd (change dir) katalógus váltás
- ▶ pwd (print w. dir.) munkakatalógus kiírása
- ▶ mkdir (make dir.) katalógus létrehozása
- ▶ rmdir (remove dir.) katalógus lista
- ▶ ls (list) katalógus törlése
- ▶ cp (copy) fájl(ok) másolása
- ▶ rm (remove) fájl(ok) törlése
- ▶ mv (move) fájl(ok) áthelyezése
- ▶ ln (link) fájl hivatkozás készítés
- ▶ ln-s szimbolikus hivatkozás készítése

# A rendszer állapotával kapcsolatos parnacsok

- ▶ who      Aktuális felhasználók
- ▶ ps          Futó processzek
- ▶ kill        Szignál küldése
- ▶ jobs        Háttérben levő feladatok lista
- ▶ date        Dátum
- ▶ top          Leg processzek
- ▶ du           Diszk használat
- ▶ df           Szabad diszkterület
- ▶ quota        Diszk kvóta ellenőrzés

# Shell program

- ▶ Egy egyszerű szöveges fájl
- ▶ A héj soronként értelmezi
- ▶ A sorokat egyenként hajtja végre
- ▶ Megjegyzés: #
  
- ▶ Ha a shell kap egy parancsnevet, az alábbiak között keres:
  - Alias-ok: helyettesítő nevek vagy pszeudo parancsok
  - Függvények (function)
  - Beépített parancsok (builtin commands) – a nagyon gyakran használt parancsokat beépítik a héjba. Így ezeket sokkal gyorsabban lehet végrehajtani. Pl.: echo, pwd, stb.
  - Külső parancsok – azok a parancsok amelyek programok formájában valahol a fájlrendszerben találhatóak.

# Shell futtatása

1. Shell-en keresztül  
/bin/sh filename
2. Vagy futtathatóvá kell tenni. (chmod)  
chmod 700 filename  
sticky bit (más jogosultságával futtatható)  
chmod 4700
3. Kezdő sor a shell elérési útja:  
#!/bin/sh

# Példa a shell futtatására

## Hello World!

- ▶ cat > hello1.sh  
echo "Hello World"  
/bin/sh hello1.sh
  
- ▶ cat > hello2.sh  
#!/bin/sh  
echo "Hello World"  
chmod 700 hello2.sh  
./hello2.sh //relatív elérés, mert nincs az elérési útban
  
- ▶ cat > hello3.sh  
#!/bin/sh  
hello="Hello World"  
echo \$hello  
chmod 700 hello3.sh  
./hello3.sh

# Shell futtatása

- ▶ sh héjprogram neve
- ▶ ./héjprogram neve  
./hello.sh
- ▶ /elérési út/héjprogram neve  
/home/gyiszaly/bin/helle.sh
- ▶ héjprogram neve (a PATH változóban megadott helyeken keresi a programot)  
echo \$PATH  
PATH="/bin:/usr/bin:/sbin:/usr/sbin:/home/gyiszaly/bin:"  
hello.sh

FUTTATHATÓVÁ KELL TENNI!  
chmod

# chmod

*-rwxrw-r-- 1 gyiszaly student 427 2005-12-14 12:55 program.sh*

chmod [augo][+−][rwx] fájlnév ...  
(u – user, g – group, o – others, a – all)

x-futtatható (1), r-olvasható (4), w-írható (2)

chmod a+r filenev – olvasási jog mindenki számára  
chmod +r filenev – ugyanaz, mivel az alapértelmezés az "all"  
chmod go-rwx filenev – a tulajdonoson kívül senkinek semmi  
chmod u+x filenev -- futtathatóvá teszi

chmod 100 filenev – csak futtatható lesz és csak a tulajdonosa számára

*---x----- 1 gyiszaly student 427 2005-12-14 12:55 program.sh*

# Futtatásnál előforduló hibák

- ▶ Nem futtatható

*ubi:~/script>program.sh*

*ubi:~/script>program.sh :Restricted acces*

- ▶ Nincs benne a PATH ban

*ubi:~/script>program.sh*

*ubi:~/script>program.sh :Command not found*

export PATH=\$PATH:. – munkakönyvtár az elérési útba

Ha minden bejelentkezéskor szükség van erre, akkor a parancsot be kell írni a . bash\_profile vagy .bashrc fájlba.

# Környezeti változók

- ▶ \$HOME: login katalógus neve
- ▶ \$PATH: keresési út a programok végrehajtásához
- ▶ \$MAIL:a file neve ahova emailjeink érkeznek
- ▶ \$PS1: elsődleges prompt
- ▶ \$PS2: másodlagos prompt
- ▶ \$EDITOR: alapértelmezett szövegszerkesztő
- ▶ \$PWD: munkakönyvár (ahol éppen állunk)
- ▶ \$OSTYPE:rendszer típusa
- ▶ \$SHELL: jelenleg futó shell típusa
- ▶ \$TERM:terminál típusa (xterm,linux, vt100, stb)
- ▶ \$RANDOM: véletlen szám
- ▶ \$USER: felhasználói név
- ▶ \$MAILCHECK: mailbox ellenőrzés gyakorisága
- ▶ Stb..... "man bash" "man sh"
  
- ▶ env és printenv parancssal jeleníthetőek meg

# Változók

- ▶ A változók neve és értéke is string, létrehozása dinamikus, nem igényel előzetes deklarációt
- ▶ szam=32
- ▶ szoveg1=hello
- ▶ szoveg2=world
- ▶ szoveg3="world"
- ▶ FONTOS! (=) értékadásnál nincs [space] sem előtte sem utána
- ▶ FONTOS! A szöveget " " közzé kell tenni.
- ▶ Változók törlése a héjból: unset

# Változók

- ▶ A változókra a héj parancsoknál a \$ jellel hivatkozunk
- ▶ Pl.: echo \$szoveg3
  
- ▶ szoveg1=hello
- ▶ szoveg2=world
- ▶ szoveg3="hello world"
  
- ▶ echo "szoveg3"
- ▶ echo "\$szoveg1 world"
- ▶ echo "\$szoveg1 \$szoveg2"
  
- ▶ szoveg4=\$szoveg1" "\$szoveg2
- ▶ ~~szoveg5=echo "\$szoveg1 \$szoveg2"~~

# Idézőjelek használata

- ▶ Az idézőjelek használatával megszabhatjuk, hogy az adott karakterSORozatot milyen mértékig akarjuk betű szerint értelmezni
- ▶ ‘...’ betű szerinti kiíratás (nincs értelmezés)  
*echo ‘pontos ido: \$ido’*
- ▶ ...” a shell számára értelmes karaktereket értelmezi és behelyettesíti pl:\$változó  
*echo "pontos ido: \$ido"*
- ▶ `...` parancsvégrehajtás, behelyettesítés, egy string tartalmába be lehet szúrni egy parancs kimenetét  
*ido=`date` (AltGr+7)*

# Idézőjeleken belül

- ▶ ! (felkiáltójel) – az előzőleg végrehajtott parancsokra hivatkozhatunk (history expansion opció) alapszinten be van kapcsolva ha a héj interaktívan fut.

pl.: !! az utolsó parancssort helyettesíti be  
!-5 – az öt parancsal megelőzőt.

echo "Szia!" – interaktív munka folyamán hibát ad, míg a végrehajtott szkriptben simán lefut  
\$ echo "Szia"!"!  
Szia!

- ▶ \ (vissza-per jel) – mint a Java-ban

# A {} zárójel

- ▶ {} – a változónevek megadásának általános szintaxisa a parancssoron, ha hivatkozunk a változó értékére. A {} zárójelek elhagyhatóak, ha a sztringben a változó nevét nem betű, számjegy vagy alulvonás jel követi.  
Pl.: \$ echo "\${szin}alma"
- ▶ \${változó:-piros} – Ha a változó nem létezik, nem ad értéket a változónak, de visszatér a "piros,-al".
- ▶ \${változó:=piros} – Ha a változó nem létezik, értéket ad a változónak és visszatér a "piros"-al.
- ▶ \${változó:?hibaüzenet} – Ha a változó nem létezik a héj program hibaüzenettel leáll, ha a héj nem interaktív (végrehajtott szkriptek esetében ez az implicit helyzet).
- ▶ \${változó:+érték } – A változó létezésének tesztelésére használjuk: a kifejezés visszatér az "érték" sztringgel, ha a változónév létezik és nem üres sztringet tartalmaz, egyébként az üres sztringet kapjuk vissza.
- ▶ \${#valtozo név} – a változóban tárolt sztring hossza (Bash)
- ▶ \${valtozo:n:m} – változóban tárolt szting részsztringje, n.-től m karakter

# A héj névvel nem rendelkező belső változói

- ▶ \$# – A parancssori paraméterek száma
- ▶ \$n – Az n-edik parancssori paraméter értéke (max 9)
- ▶ \$0 – A pillanatnyi héjprogram neve (nulla)
- ▶ \$\$ – A futó program azonosítója
- ▶ \$? – exit státusz
- ▶ \$- – A héjprogramot végrehajtó héjnak átadott kapcsolók
- ▶ \$\* – Valamennyi parancssori paraméter egyben, egyetlen karakterláncként („\$1 \$2 ...\$9)

# A héj névvel nem rendelkező belső változói

► Példa:

```
cat >program.sh
echo $#
echo $3
echo $0
echo $*
```

```
program.sh 1 3 5
```

```
3
```

```
5
```

```
program.sh
1 3 5
```

# Matematikai kifejezések

- ▶ + összeadás
- ▶ - kivonás
- ▶ \* szorzás
- ▶ / egészosztás
- ▶ % maradékképzés
  
- ▶ <, -lt            Kisebb
- ▶ <=, -le        Kisebb egyenlő
- ▶ =, -eq         egyenlő
- ▶ ==, -eq        Egyenlő
- ▶ !=, -ne        Nem egyenlő
- ▶ >=, -ge       Nagyobb egyenlő
- ▶ >, -gt        Nagyobb
- ▶ expr (reláció)      1 – igaz  
                          0 – hamis

# Matematikai kifejezések

- ▶ | „VAGY” operátor.  
Visszatérési értéke az első paraméter, ha az nem nulla, vagy nem üres karakterlánc, ellenkező esetben a második.
- ▶ & „ÉS” operátor.  
Visszatérési értéke az első paraméter, ha egyik argumentuma sem nulla vagy üres karakterlánc. Ellenkező esetben nulla.
- ▶ expr – kis műveletek végrehajtására, csak egészekkel dolgozik
  - szam=`expr 1 + 1`
  - szam=`expr 1 \\* 1` (védő karakter)
  - echo \$((szam1+szam2))

# Aritmetikai kiértékelés - (( ... ))

- ▶ valtozo=\$(( aritmetikai műveletek ))
- ▶ Az operátorok a C nyelvből valóak, minden operátor használható.
- ▶ A Bash csak egész számokkal dolgozik. Kiértékeléskor a Korn héj dupla pontosságú valós számokat használ (a Korn héj tud valós változókkal dolgozni).
- ▶ A \$ jelet nem kell változók előtt használni, de lehet. Kivételt képeznek a héj speciális változói, azok előtt viszont kell. Pl.:
  - b=\$(( a + 2 )), b=\$(( \$a + 2 )), b=\$(( \$1 + 2 ))
- ▶ A C nyelv hozzárendelő operátorai is használhatóak például: (( x+=2 )), de a ( \$x+=2 ) már helytelen
- ▶ Zárójelezés használható.
- ▶ A relációs operátorok esetében az igaz értéke 1 – de csak a () belséjében. A tesztek esetében a (( )) –ból való kilépés után a héj a ? változót fordított értékre állítja.
- ▶ A szóköz elválasztór itt nem lényeges pl.: ( (2+2) ) vagy (( 2 + 2 )) is helyes
- ▶ A hatványozás operátora a \*\*

# Standard kimenet és bemenet

- ▶ A UNIX minden eszközt, hardvert, fájlként kezel a virtuális fájlrendszerre, így az eszközökre való írás mindenkor egyszerű fájlba való írásra vezethető vissza
- ▶ A parancsértelmező a parancsokat és adatokat a standard bementről vagy állományból olvassa.
- ▶ Unix terminológia: (háromágú átfolyó csatorna)
  - szabványos bemenetre érkező adatokat fogadja (*ált. billentyűzet, ..., file, másik program kimenete, hardware eszköz,*) (feldolgozza)
  - szabványos kimentre küldi (*ált. képernyő... file, másik program bemenete, hardware eszköz*)
  - szabványos hibacsatorna (Hiba csatorna + egyéb közlendők) (*ált. képernyő, file*)
- ▶ Azonosítók
  - standard bemenet, azonosító: 0
  - standard kimenet, azonosító: 1
  - standard hibakimenet, azonosító: 2

# Átirányítás

- ▶ Átirányításnak nevezzük az a műveletet, amikor egy adott kimenetre (vagy bemenetről) érkező adatsort egy, az eredetitől különböző kimenetre küldünk (illetve bemenetről fogadunk).
- ▶ A standard bemenet átirányítása ( < )  
*program < file* (program bemenetét file-ból veszi)  
*program << VEGE*  
ezt a prog megkapja a standard bemenetén a VEGE végjelig
- ▶ A standard kimenet fájlba irányítása ( > )  
*program > file* - (program kimenet file-ba, ha már létezik felülírja !!!, ha nem létrehozza)  
*program >> file* - (kimenet file-hez való hozzáírása, ha nem létezik létrehozza)  
*program > /dev/audio*  
*program > /dev/printer*  
*program > /dev/null*

# Csővezetékek

- ▶ A csővezetékek kommunikációs csatornák, amelyeken keresztül egy folyamat kimenete egy másik bemenetére irányítható
- ▶ \$ cat lista.txt | sort

# Szűrők

- ▶ Tipikusan a szabványos inputról olvasnak és a szabványos kimenetre írnak. Egyszerűen csővezetékbe szervezhetők.
- ▶ Gyakran fájl paramétereket is értelmeznek (pl. more, cat, sort)  
parancs1 | parancs2 -paraméter | parancs3 -paraméter ...
  
- ▶ cat                   – fájlok összemásolása
- ▶ more                  – fájl(ok) kiírása ernyőképenként
- ▶ less                  – fájl(ok) kiírása ernyőképenként
- ▶ head                 – fájl első n sora
- ▶ tail                 – fájl utolsó n sora
- ▶ tee                   – adatfolyam elágaztatása
- ▶ sort                 – sorbarendezés
- ▶ tr                    – karakter helyettesítő
- ▶ wc                   – sor és karakterszámláló
- ▶ diff                 – file összehasonlítás
- ▶ sed                   – adatfolyam editor
- ▶ uniq                 – előfordulást vizsgál

# cat

- ▶ cat – szabványos bemenetről olvas szabványos bemenetre ír
- ▶ cat (önmagában)
- ▶ cat adatok.txt
- ▶ cat > adatok.txt
- ▶ cat >> adatok.txt
- ▶ cat < adatok.txt

*cat adatok.txt / sort*

*cat adatok.txt / sort >> rendezett\_adatok.txt*

*more adatok.txt / sort >> rendezett\_adatok.txt*

*less adatok.txt / grep „003670 / sort >> voda-szamok-sorbarendezve.txt*

# I/O átirányítása

- ▶ A > < jelölések bármelyikét megelőzheti egy szám. Ekkor a szabványos bemenet ill. kimenet helyett a számnak megfelelő állományleírót kell érteni.
- ▶ >&n a szabványos kimenet helyett az n. állományleírót használja
- ▶ <&n a szabványos bemenet helyett az n. állományleírót használja

# Hibacsatornák átirányítása

- ▶ Szabványos bement (stdin) 0
- ▶ Szabványos kimenet (stdout) 1
- ▶ Szabványos hiba (stderr) 2

*cat adatok.txt 1> kimenet.txt*

( ☺ nem jelöljük)

*cat adatok.txt 2> hiba.txt*

( 2>állomány a hibakimentet az állományba irányítja)

*cat adatok.txt > kimenet.txt 2> hiba.txt*

*cat adatok.txt > kimenet+hibak.txt 2>&1*

(a hibakimenet és a szabványos kimenet összekapcsolódik)

- ▶ Saját hibaüzenet gyártása: (*programon belül*)

*echo „Hibaüzenet .....”* (nem jó mert csak a *stdout*-ra küldi)

*echo „Hibaüzenet .....” 1>&2* (így már tudjuk *stderr*-ként kezelni)

- ▶ *Meg is szabadulhatunk a hibáktól:*

*program 2> /dev/null*

# Csövek elágaztatása

- ▶ `tee [-a] {fájl}`
- ▶ A parancs a bemenetet a kimenetre és a megadott nevű fájlba írja (a -a kapcsolóval hozzáfűzi, nélküle új fájlt nyit).

- ▶ Példa:

```
$cat lista.txt / sort / tee rendezett.txt / head -1
```

```
$ls -al / tee filelista
```

```
$ps auxf / sort / tee processlista
```

# Vezérlési szerkezetek

- ▶ A C program visszatérési értéke, amely bekerül a ? változóba (lekérdezhető \$?)
  - 0 – sikeres végrehajtás
  - Pozitív egész – valamilyen hibakód
- ▶ Fordított, mint a C vagy a Java!!!
- ▶ && szerkezet – végrehajtja a parancssoron következő parancsot, amennyiben az előző lefutott parancs "jól" futott le, tehát igaz visszatérített értékkel zártul.  
Pl.: ls l1.txt && echo 'van ilyen fájl,
- ▶ || szerkezet – végrehajtja a második parancsot, ha az első nem jól futott le  
pl.: ls l1.txt || echo ,üzenet') látszik.

# test vagy [] parancs

## ▶ Fájlok tesztelése

- - d file – Igaz ha a file létezik és könyvtár.
- - e file – Igaz ha a file létezik.
- - f file – Igaz ha a file létezik és szabályos fájl.
- - L file vagy - h file – Igaz ha a file létezik és szimbolikus hivatkozás (szimbolikus link).
- - r file – Igaz ha a file létezik és olvasható.
- - s file – Igaz ha a file létezik és 0-nál nagyobb méretű.
- - w file – Igaz ha a file létezik és írható.
- -x file – Igaz ha a file létezik és végrehajtható.
- -0 file – Igaz ha a file létezik és az aktuális felhasználó tulajdonában van.
- -G file – Igaz ha a file létezik és az aktuális csoport tulajdonában van.  
file1 -nt file2 – Igaz, ha file1 újabb (a módosítási idő), mint file2.  
file1 -ot file2 – Igaz ha file 1 régebbi, mint file2.
- file1 -ef file2 – Igaz ha file1 és file2 –nek azonos eszköz- és i-node száma van. Tulajdonképpen ez azt jelenti, hogy hard linkek.

# test parancs

## ▶ Sztringek tesztelése:

- `-z string` – Igaz ha a string 0 hosszúságú.
- `-n string` – Igaz ha a sztring nem 0 hosszúságú
- `string1 = string2` – Igaz ha a stringek megegyeznek.
- `string1 != string2` – Igaz ha a sztringek nem egyeznek meg.

## ▶ Logikai tesztek két test kifejezés között:

- `! expr` – Igaz ha expr hamis.
- `expr1 -a expr2` – Igaz ha expr1 és expr2 is igaz
- `expr1 -o expr2` – Igaz ha expr1 vagy expr2 igaz

# test parancs

- ▶ Számokat tartalmazó sztringek összehasonlítása:
- ▶ arg1 OP arg2
  - OP operátor valamelyik a következőkből:
  - –eq, –ne, –lt, –le, –gt, –ge.
  - equal, not equal, less than, less or equal, greater than, greater or equal

# test parancs példák

```
$ # létezik-e a file  
$ test -f 1.txt ; echo $?  
0
```

```
$ # könyvtár-e  
$ test -d 1.txt ; echo $?  
1
```

```
$ # írható-e  
$ test -w 1.txt ; echo $?  
0
```

```
$ # létrehozok 2 sztringet  
$ a=,abc'  
$ b='def'  
$ # a $a hossza 0 ?  
$ test -z $a ; echo $?  
1
```

```
$ # $a egyenlő-e $b-vel  
$ test $a = $b ; echo $?  
1
```

```
$ # létrehozok 2 sztringet  
amelyek számot tartalmaznak  
$ x=2  
$ y=3  
$ # $x értéke kisebb mint $y ?  
$ test "$x" -lt "$y" ; echo $?  
0
```

```
$ # $x értéke kisebb mint 1 ?  
$ test "$x" -lt "1" ; echo $?  
1
```

# if szerkezet

A feltételt jelentő parancssorban bármilyen parancs végrehajtható, akár egyedülálló, akár csővezetékkel összekötött parancs lehet.

```
if parancssor  
then  
    parancssor  
    ...  
else  
    parancssor  
    ...  
fi
```

```
if parancssor  
then  
    parancssor  
    ...  
elif parancssor  
then  
    parancssor  
    ...  
else  
    parancssor  
fi
```

# if példák

A test zárójelezésénél a [ ] zárójelek minden két oldalán egy elválasztó szóközt kell hagyni!!!

```
#!/bin/bash
szoveg="Ezt írjuk a fájl végére."
file="1.txt"
#teszteljük, hogy a fájl nem szabályos fájl
if ! [ -f „$file” ]
then
echo "$file" nem létezik vagy nem szabályos fájl
exit 1
fi

#teszteljük, hogy a fájl írható-e
if [ -w "$file" ]
then
echo "$szoveg" >> "$file"      #szöveg a fájl végére
else
echo "$file" nem írható
fi
```

# Többszörös elágazás – case

## case változó in

```
minta1 ) parancsok ::;  
minta2 ) parancsok ::;  
...  
mintaN ) parancsok ::;
```

esac

A megadott változóra illeszti a következő sorok elején található mintákat. Ahol a minta egyezik, ott végrehajtja a jobb zárójel utáni parancssorozatot amelyet kettős ; karakter zár le.

**Csak az első találatnak megfelelő alternatívát hajtja végre!!!**

A minta típusa shell minta, azaz használható a \*, ?, [], |

- abc – pontosan abc-re illeszkedik
  - \* – bármire illeszkedik
  - ?bc – bármi az első karakter pozíció, és utána pontosan bc
  - ab | cd – ab vagy cd láncokra illeszkedik

# case példa

```
read x
case $x in
    a) echo Az 'a' betűt nyomta le!;;
    b) echo A 'b' betűt nyomta le!;;
    *) echo Egyéb betű!;;
esac
```

# break, continue, exit, shift

- ▶ break – megszakítja a ciklust
- ▶ continue – a következő iterációra lép
- ▶ exit – kilép a héj programból, ha sikeresen futott le a parancs, akkor 0-val kell visszatérjen
- ▶ shift – elmozgatja a parancssori paramétereket eggyel balra, az argumentumok számát tartalmazó # változó értéke eggyel csökken

```
#!/bin/bash
while [ $# -gt 0 ] #amíg van még argumentum
do
    echo A '$#' értéke: $#, a '$1' értéke: $1
    shift
done
```

# for ciklus

```
for i in lista  
do  
    parancssor  
done
```

- ▶ Az i változó sorra felveszi a lista elemeinek értékét. A lista megadása opcionális, amennyiben elhagyjuk, akkor az i változó a parancssor paraméterein iterál, tehát a @ változóban található sztringeket járja végig.

# for példa

```
lista=`ls -al`  
for i in $lista  
do  
    echo $i  
done
```

```
#!/bin/bash  
for i in $(seq 1 5)      - ciklus 1-5-ig  
do  
    echo $i  
done
```

# C stílusú ciklus

```
for (( i=0; i<10; i++ ))  
do  
    héj parancsok  
    ...  
done
```

Példa:

```
#!/bin/bash  
for (( i=1; i<=10; i++ ))  
do  
    echo $i  
done
```

# while és until ciklusok

- ▶ A while addig folytatja a ciklust, míg feltétele igaz

```
while parancssor  
do
```

```
    parancssor
```

```
    ...
```

```
done
```

- ▶ Az until addig folytatja a ciklust, míg feltétele hamis

```
until parancssor  
do
```

```
    parancssor
```

```
    ...
```

```
done
```

# while példa

```
i=1
while [ $i -le 10 ]
do
    echo $i
    i=`expr $i + 1`
done
```

```
#!/bin/bash
end="end"
#addig olvas sorokat a terminálról, amíg begépeljük
#az "end" sztringet
while [ "$line" != "$end" ]
do
    read line      – terminálról olvas, hamissal tér vissza, ha
                    beolvasás nem sikerül
done
```

# until példa

```
i=1
until [ $i -ge 10 ]
do
    echo $i
    i=`expr $i + 1`
done
```

```
#!/bin/bash
#
until [ $line ]
do
    echo "A $line változó értéke: " $line
    echo "írja be pontosan az \"abc\" sztringet"
    read line
    if [ "$line" != "abc" ]; then
        unset line                         #változó törlése
    fi
done
```

# select parancs

select változó in lista

do

    parancsok

done

- ▶ Egy kis menüt ír ki a lista (sztring lista) segítségével. Ezek után készenléti jelet ír ki . A menü sorának sorszámaival választjuk ki a menüpontot. A változó értéke felveszi a listából kiválasztott karakterlánc értékét, és ezt fel lehet használni parancs végrehajtásra. Ugyanakkor a héj REPLY változója felveszi a beütött szám vagy más sztring értékét.
- ▶ A szerkezet addig ismétli a menü kiírását és a végrehajtást amíg bemenetként állomány vége jelet kap (Ctrl-D) vagy a parancsok végrehajtási részében egy break parancs hajtódiik végre. Ha olyan bemeneti számot adunk meg ami nincs a menüben, a változó üres sztringre lesz állítva (a REPLY viszont megkapja a beütött értéket).

# select példa

```
#!/bin/bash
PS3= '»'
select változó in Első Második Harmadik Vege
do
    echo $valtozo
    echo a '$REPLY' változó értéke ilyenkor: "$REPLY"
#itt bármilyen parancsot végre lehet hajtani, felhasználva
#a $valtozo es $REPLY értékeit
    if [ "$valtozo" = 'Vege' ];then
        echo "...és jön a kilépés a select-ből"
        break
    fi
done
```

# read parancs

- ▶ Egy sort olvas be a parancssoról egy változóba, vagy változó listába
- ▶ Ha elhagyjuk a változónevet, akkor a héj REPLY nevű beépített változójába kerül a bemeneti sor
- ▶ A reád igaz értékkel tér vissza a ? változóban ha sikerült beolvasnia. Fájl vége (Ctrl-D) esetén hamissal tér vissza
- ▶ Opciói:
  - -p prompt – kiír egy készenléti jel karakterláncot olvasás előtt
  - -s – silent, nem használ visszhangot, tehát nem látjuk a leütött betűket: jelszó beolvasásra használjuk
  - -t timeout – vár timeout másodpercig, ha az alatt nem írunk be semmit, visszatér
  - -n nkar – csak nkar darab karaktert olvas, utána visszatér
  - -d kar – delimiter, más elválasztót keres a sorok végén mint az újsor karaktert

# read példa

*read valtozo* - az egész sor a valtozo-ba

*read v1 v2 v3* - a héj implicit elválasztói szerint feldarabolja a beolvasott karaktersorozatot, és a változókban teszi őket

`$ read -p "Írd be a neved:" line`

`$ read -p "Igen vagy nem [I/N]?:"` -n 1 -t 3 igen\_nem  
3 másodpercig vár, és csak egy karaktert vár vissza

*while read line*

*do*

*# a beolvasott sor feldolgozása*

*done*

Kilép a ciklusból, ha nem tud tovább olvasni

# Fájl olvasása read-del

Fájlból is olvashatunk vele, de először ehhez az állományt az exec parancssal meg kell nyitni, és utána lezárni.

*exec 6< "1.txt"#bemeneti fájl megnyitása 6-os azonosítóval*

*read -u 6 line #egy sor beolvasása a line változóba*

*echo \$line*

*exec 6<&- #fájl lezárása*

# Függvények

- ▶ A függvények úgy viselkednek mint egy-egy külön kis shell program, nem kell argumentumlistát definiálni, és argumentumaik az \$1, \$2, stb. változókkal érhetőek el
- ▶ Szintaxisa:

```
function nev ()  
{  
    #a függvény kódja  
}
```

- ▶ A function kulcsszó akár el is hagyható
- ▶ A függvény megkapja a hívó héj munka könyvtárát, környezeti változóit. A függvényben létrehozott változók láthatóak lesznek a hívó héjban, és a függvény is látja az ott előzőleg létrehozott változókat.
- ▶ Létezik a lokális változó fogalma, amit a local kulcsszóval adhatunk meg
- ▶ Rekurzív hívás lehetséges

# Függvény példa

```
function equal()
{
if [ "$1"= "$2" ]
then
    return 0
else
    return 1
fi
}

if equal "$a" "$b"
then
echo egyenloek
fi
```

Ha külön állományba írjuk a függvényt, akkor azt az aktív héjban a „..” parancsal használhatjuk. Törölni az unset parancsal lehet ( – v ha csak a megadott változóneveket, – f ha csak megadott függvényneveket akarunk törölni)

```
./equal.sh
$equal "a" "b
$echo $?
1
```

# Függvény visszatérési értéke

- ▶ A return parancssal adhatunk meg visszatérési értéket, ami a hívó héj ? változójába kerül
- ▶ Az eredmények visszaadásának módjai:
  1. globális változó beállítása
  2. nyomtatás a függvényben és parancssor helyettesítés a hívásnál

```
function getline
{
    line=""
    if ! read -p "Kérek egy sztringet:" line
    then
        echo "Olvasás hiba";
        return 1
    fi
    return 0
}
```

Használata:

```
./getline.sh
$ if getline ; then echo $line ; fi
```

# Függvény példák

## ▶ Lokális változók

```
function teszt ()  
{  
    local valtozo=3  
    local eredmeny=  
    $(( valtozo * $1 ))  
    echo $eredmeny  
}
```

```
$ . teszt.sh  
$ eredmeny=33  
$ teszt 8  
24  
$ echo $eredmeny  
33  
$ echo $valtozo  
  
$
```

## ▶ Rekurzió

```
function fact_fgv()  
{  
    if (( $1 == 0 >)) ; then  
        fact=1  
        return  
    else  
        fact_fgv $(( $1 - 1 ))  
        fact=$((fact * $1 ))  
    fi  
    return  
}
```

# Tömb

- ▶ Csak az egydimenziós tömböket ismeri
- ▶ Létrejönnek, ha első elemüknek értéket adunk
- ▶ Méretük korlátlan
- ▶ Indexük csak egész szám lehet és 0-tól indul

```
tombnev[indexertek]=kifejezes  
 ${tombnev[indexertek]}
```

```
valtozo=5  
for i in 1 2 3  
do  
    tomb[$i]="$valtozo+$i  
    echo ${tomb[$i]}  
done
```

# Regular expressions

- ▶ Szabványos kifejezések
- ▶ Karakterek vagy szövegrészek keresésére, szűrésére, cseréjére alkalmas
- ▶ Vezérelhetőek vele a programok
- ▶ Megvalósítása a grep, sed, awk parancsokkal történhet

# Regular expressions

- ▶ A **grep** a bemenetre érkező sorok közül csak azokat küldi ki a kimenetre, amelyek megfelelnek a megadott kifejezésnek:  
`tail -f acces.log | grep "Login"  
grep "Login" access.log  
grep "Login" *`
- ▶ **sed (stream editor)** a megadott parancsok szerint módosítani is tudja a kimenetet (csere). Sor alapú de a teljes bemenetet egyben is kezelheti.  
`sed 's/Login:root/Login:nobody/' -f access.log`
- ▶ **awk** – programozási nyelv szövegfeldolgozásra. Soronként kezeli a bemenetet, és a kimenetet módosítani tudja!!! A sorokon belül mezőket is megkülönböztet (lehetőséget ad a pozicionálásra).

Jan 2 09:00:01 zeus.nyf.hu Login: root

```
awk '{print $4,$6}'  
zeus.nyf.hu root
```

# Regular expressions

- ▶ Kivételes karakterek \*, ., [ ], \, ^, \$, +
- ▶ Ezeknek valamilyen jelentésük van
- ▶ ^ – sor eleje: **grep „^1” filename**
- ▶ \$ – sor vége: **grep „1\$” filename**
- ▶ . – Tetszőleges karakter (kivéve újsor)  
**grep “...” filename**
- ▶ \ – speciális jelentés ki/(be) kapcsolás  
**grep “\.\.\.” filename**
- ▶ []
  - – bármely karakter illeszkedése grep “[123]” filename
  - – tartomány illeszkedése grep “[1–3]” filename
  - – nem illeszkedés grep “[^1–3]” filename

# Regular expressions

- ▶ \* - ismétlő karakter A \* előtt álló karakter vagy kifejezés tetszőleges számú (akár 0) előfordulása.  
`grep "A.*" filename`  
`grep "A*" filename`
- ▶ + - ismétlő karakter, de megköveteli, hogy az előtte lévő karakter/kifejezés legalább egyszer előforduljon.  
`grep "A\+" filename`
- ▶ () - csoportba rendezi a szabványos kifejezéseket, egységként kezelik
- ▶ | - logikai VAGY  
`grep „\(\w{A}|\w{B}\)" filename`
- ▶ {} - illeszkedések száma ({x} pontosan , {x,} legalább, {x,y} intervallum)  
`grep "A\{3\}" filename,`  
`grep "[1-9]\{10\}" fn,`  
`grep "[1-9]\{10,\}" fn,`  
`grep "[1-9]\{10,20\}" fn`

# Regular expressions

► Példák:

”^[0-9]” – számmal kezdődik

”^[0-9]\{1\}[^ 0-9]” – 1 db számmal kezdődik és követi valami

”^[0-9]\{1\}[^ 0-9]\*” – 1 db számmal kezdődik, de lehet csak 1 db szám is

„Dr.\|dr.”

„Dr.[A-Za-z]\|dr.[A-Za-z]”

„Dr\.[A-Za-z]\|dr\.[A-Za-z]”

„Dr\.[A-Z][a-z]\|dr\.[A-Z][a-z]”

„Dr\.\[\][A-Z][a-z]\|dr\.\[\][A-Z][a-z]”

„[dD]r\.\[\][A-Z][a-z]”

„ ^\([dD]r\.\[\])\{1,2\}[\][A-Z][a-z]\+”

# sed editor

- ▶ **sed – stream editor**
- ▶ folyamat editor vagy programozható szövegszerkesztő
- ▶ A szabványos bemenetére érkező szöveget képes (röptében) feldolgozni és átalakítani (lehet file is)
- ▶ Hogyan?
  - a feldolgozandó szöveget soronként egy átmeneti tárba, az úgynévezett mintatérbe olvassa be,
  - szabályos kifejezések alapján megkeres benne bizonyos részeket,
  - majd elvégzi rajtuk az egybetűs parancsok formájában megadott műveleteket.
- ▶ ... | sed 'program'  
sed 'program' filenév
- ▶ program = „a sed saját nyelvén írt szövegfeldolgozási utasítássorozatot jelenti (szabályos kifejezések + egybetűs kapcsolók)”

# sed editor

- ▶ Általános program: <cim1>,<cim2> parancs
  - <cím>
  - szám (a bemenet adott sorszámú sora)
- ▶ **sed 10 parancs** – csak a 10. soron hajtódik végre
- ▶ **sed 1,10 parancs** – tartomány az 1–10 sorig hajtódik végre
- ▶ **sed parancs 10** – a keresett mintának csak a 10. előfordulásán hajtódik végre
- ▶ csak soron belül szabványos kifejezés /...../
- ▶ **sed /[0-9]/parancs** – csak a számokat tartalmazó soron érvényesül
- ▶ **sed /[a-z]/parancs** – csak a kisbetűket tartalmazó soron érvényesül

# sed editor

## ► Alapvető parancsai

- **p** kiíratás
- **d** törlés
- **s** helyettesítés
- **a** hozzáfűzés
- **i** beszúrás
- **c** a mintatér cseréje
- **y** a karakterek cseréje
- **n** next, még 1 sort olvas a bemenetről és hozzáfűzi a mintatérhez

# **sed editor**

- ▶ **p – kiíratás**
  - **cat szöveg | sed p** – duplán jeleníti meg a sorokat
  - **cat szöveg | sed '10 p'** – duplán jeleníti meg a 10. sort
  - **cat szöveg | sed '1,10 p'** – duplán jeleníti meg az 1–10-ig a sorokat
  - **-n** letilthajuk a default megjelenítést
  - **cat szöveg | sed -n '10 p'** – csak a 10. sort jeleníti meg
- ▶ **d – törlés**
  - **cat szöveg | sed '1,10 d'** – törli a 1–10-ig a sorokat

# sed editor

## ▶ s (g) – helyettesítés

- **s/mit\_cserélünk/mire cseréljük/** – csak az első előfordulását helyettesíti
- **s/mit\_cserélünk/mire cseréljük/g** – minden előfordulását helyettesíti
- **s/mit\_cserélünk/mire cseréljük/n** – az n-ik előfordulását helyettesíti

```
echo 1234abba | sed 's/a/A/' 1234Abba
```

```
echo 1234abba | sed 's/a/A/g' 1234AbbA
```

```
echo 1234abda | sed 's/[a-c]/A/a' 1234AAbA
```

## ▶ y – karaktereket helyettesít

- **y/helyettesítendő/helyettesítő/** – a kettőnek karakterszáma egyenlőnek kell lennie

```
echo "abcdabcdABCDABCD" | sed 'y/abcd/qxyz/'
```

# sed editor

- ▶ (a) hozzáfűzés (i) beszúrás (c) a mintatér cseréje
- ▶ Nem változtatják meg a mintateret a program későbbi parancsai nincsenek rá hatással
- ▶ A hozzáadott szövegnek a programon belül mindenkorábban új sorban kell kezdődni

```
#!/bin/sh
```

```
echo „1234abc”| 'i\  
Csak számok:  
s/[a-z]//g'
```

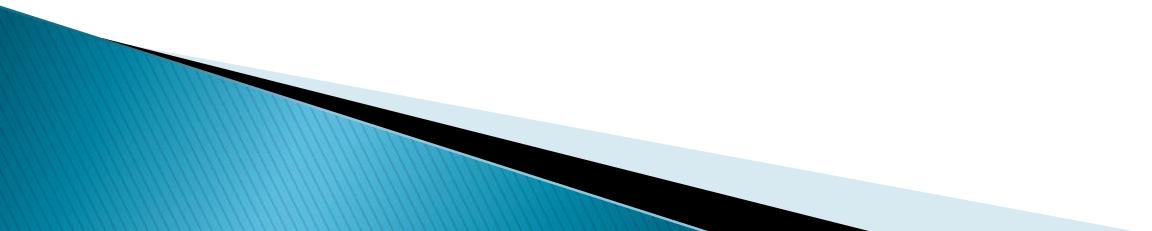
```
#!/bin/sh
```

```
echo „1234abc„ | 'i\  
Csak számok:
```

```
s/[a-z]\+//'  
echo 1234abcde | sed '/\([0-9]\+\)\([a-z]\+\)/\2\1/'
```

# sed editor

- ▶ `sed 'program' szoveg.txt > szoveg.txt`  
NEM JOO
  - `sed 'program' -f szoveg.txt`
- ▶ utolsó sor \$ szinbolum
  - `sed -n '$p' filename`



# Az AWK

- ▶ Awk, a pattern scanning and processing language
- ▶ Szövegfeldolgozásra szakosodott programnyelv, több változat: awk, gawk (GNU project), tawk (MS-Windows DLL), awka, mksawk, awkcc (c interpreter), mawk, original-awk
- ▶ A C szövegszerkesztésre kihegyezett változata, képességei azonosak egy grafikus felületen futó táblázatkezelővel
- ▶ Az awk alapvető feladata, hogy olyan szövegegységeket keressen file-okban, amelyek tartalmaznak egy bizonyos mintát.
- ▶ A minta lehet szabványos kifejezés vagy logikai feltétel, ha nincs minta minden sorra végrehajtja
- ▶ Az awk programok adatvezéreltek
  - először az adatot adjuk meg amivel dolgozni szeretnénk
  - aztán azt, hogy mit szeretnénk csinálni

# AWK opciók

- ▶ -F c – Field separator, azaz mezőelválasztó karaktert megadása. Implicit elválasztó a szóköz és a TAB, illetve ezek ismétlődése
- ▶ -f fájl – a szkript fájl megadása.
- ▶ -v var=val – a var változónak val értéket ad indulás előtt
- ▶ awk program – szöveg [ fájl(ok) ...]  
\$ awk '{print}' teszt.txt
- ▶ awk -f program-fájl [fájl (ok)...]  
\$ awk -f doit.awk teszt.txt
- ▶ Az Awk programok .awk kiterjesztésűek, és a következő elemmel kezdődik:  
#!/bin/awk -f

# Az AWK futtatása

- ▶ Szabványos bemenetről:

```
... | awk '{parancsok}'
```

```
$ cat teszt.txt | awk '{print}'
```

- ▶ Feldolgozandó file megadásával:

```
awk '{parancsok}' filename1 <filename2>
```

```
$ awk '{print}' teszt.txt
```

- ▶ Ha a program hosszabb vagy külön file-ban van:

```
... | awk -f programfile
```

```
awk -f programfile filename1 <filename2>
```

# Feldolgozás

minta1 {tevékenység1}

minta2 {tevékenység2}

- ▶ Soronként történik (rekord), a sorokat mezőkre osztja (legkisebb feldolgozási egység)
- ▶ A mezőket a mezőelválasztó karakterrel tagolhatjuk, értéke definiálható [-F] (alapértelmezetten: szóköz, tabulátor)  
!!! (LEHET SZABVÁNYOS KIFEJEZÉS IS)  
A mezőkre való hivatkozás: \$1 , \$2.....\$i, a \$0 az egész sort jelöli
- ▶ A minták megadása reguláris kifejezéssel /.../jelek között, logikai feltételek a C-ben megszokott operátorokkal „a>2”
- ▶ Kiíratás: print, printf\*\*\* (újsor karakter nélkül)  
awk '{print \$2,\$1}' ; awk '{print \$2" "\$1}'; awk '{print Mezők:\$2" "\$1}' ;

# Programozás

- ▶ A programsorok szabályokból állnak: egy szabály egy mintából (pattern) és a hozzá tartozó tevékenységből (action) vagy utasításokból (statement) áll, amely {} közé zárt blokkban van:  
minta {utasítások}
- ▶ A {} blokkban utasítások követik egymást. A nyelvben függvények vagy nyelvi szerkezetek hívhatóak különböző műveletek elvégzésére.

```
$ echo '112 LinkinPark Numb numb.txt 35, | awk '{print $1, $4}'  
112 numb.txt
```

```
$ echo piros | awk '{print $0}'  
piros
```

```
$ echo piros | awk '/^p/'  
piros
```

p-vel kezdődő sorokat

# printf

- ▶ A kiíratás vezérlése a %-jellel történik:  
( '%' jel áll, ahoz tartozik egy sorszámozott paraméter, az érték megjelenítését egy betű jelöli)
  - szám esetén a "%d" decimális, a "%x" hexadecimális
  - sztringekkel esetén %s-t használunk
- ▶ A kiírás szélességét a '%' jel után lehet megadni, ha az adott szöveg ennél rövidebb, akkor előről szóközökkel lesz feltöltve (jobbra zárás), azonban ha a '%' jel után egy '-' áll, akkor balra zárt.

```
awk '{printf ("%10s",$1)}'
```

# Változók

## ▶ Felhasználó által megadott változók:

- nem kell külön deklarálni, első értékkedáskor létrejönnek
- Megadhatjuk program futása előtt is, vagy a héjprogram változóját is átvehetjük.
- Két típus van:
  - Dupla pontosságú valós – alapértelmezetten 0
  - Sztring – alapértelmezetten üres sztring

```
$ awk 'BEGIN { print s+2; }'  
2
```

```
less /etc/passwd | awk -F: '{nev="Felhasznaloi nev:"; konyvt="Konyvtar:"; print nev  
$1" " konyvt $6}'
```

```
less/etc/passwd | awk -F: -v nev="Felhasznaloi nev:" -v konyvt="Konyvtar:" '{print  
nev $1" " konyvt $6}'
```

- BEGIN blokkban használva:

```
awk-F: 'BEGIN {nev="Felhasznaloi nev:"; konyvt="Konyvtar:"} {print nev,$1" ", konyvt  
$6}'
```

# Változók

## ▶ Belső változók:

- FILENAME – aktuális bemeneti fájl neve
- FS – bemeneti mezőelválasztó karakter
- NF – az aktuális sor mezőinek száma
- NR – az aktuális sor száma
- OFS – kimeneti mezőelválasztó karakter

```
awk -v FS=: -v nev="Felhasznaloj nev:" -v konyvt="Konyvtar:"  
'print nev $1 ", " konyvt $6'
```

Megszámozott sorok:

```
awk -v FS=: -v nev="Felhasznaloj nev:" -v konyvt="Konyvtar:"  
'print NR " " nev $1 ", " konyvt $6'
```

```
$echo 'a b' | awk '{$1="b"; print $0}'  
b b
```

# BEGIN – END blokk

- ▶ A főciklustól független műveletek végrehajtását teszi lehetővé
- ▶ A BEGIN még a sorok feldolgozása előtt, de csak egyszer fog végrehajtódni
- ▶ Az END a sorok feldolgozása után végez el valamilyen műveleteket

awk 'BEGIN {bevezető műveletek} {förogram} END {záró műveletek}'

```
awk -F: 'BEGIN {nev="Felhasznaloi nev:"; konyvt="Konyvtar:";  
print "ELEJE"} {print nev $1 ", " konyvt $6} END {print "VEGE"}'
```

```
$ echo '1 2' | awk 'BEGIN{a=1} {a=a+$1+$2} END {print a}'  
4
```

# Operátorok

- ▶ = értékkomparáció
- ▶ || vagy
- ▶ && és
- ▶ !tagadás
- ▶  $>= < <= == !=$  relációs operátorok
- ▶ + - \* / % ++ -- aritmetikai operátorok
- ▶  $\sim \text{!} \sim$  reguláris kifejezésre való illeszkedés, nem illeszkedés

# Példák

```
$ echo ' ' | awk '{i=0; i++; print i}'  
1
```

```
$echo '1.1 2.2' | awk '{ s=$1+$2; print s }' //vagy tizedesvesszővel  
3.3
```

```
$ echo '112' | awk '{if ($1 ~ /^[0-9]+$/)\\  
{print "az első mező számokból áll"}'  
az első mező számokból áll
```

```
$ echo 'abc' | awk '{if ($1 !~ /^[0-9]+$/) \\  
(print "az első mező nem áll számokból")}'  
az első mező nem áll számokból
```

```
$ echo '1 2 3 abc def' | awk '{print $4 $5}'  
abcdef
```

# Vezérlési szerkezetek

## A C szintaktikájához hasonlóan

if(feltétel)  
    utasítás

```
{if ($2!=root) print $2}
```

else  
    utasítás

```
echo '2.2'| awk '{ if ($1==1)  
{print "igaz"} else {print  
"hami"}}'  
hamis
```

for(kifejezés1;feltétel;kifejezés2)  
    utasítás

for(változó in tömb) utasítás

```
{for(i=1 ;i<11 ;i++)}
```

while(feltétel)  
    utasítás  
    break  
    continue  
    next  
    exit

```
{i=0  
while (i<10)  
print i  
i++}
```

# Beépített függvények

- ▶ `cos(kif)` – a kif koszinusza
- ▶ `exp(kif)` – a kif exponenciális függvénye
- ▶ `getline()` – a következő sor beolvasása. Visszatérés: 0, ha fájl vége, 1 ha nem
- ▶ `index(s1,s2)` – az s2 kezdőpozíciója s1-ben
- ▶ `int(kif)` – a kif egészrészre
- ▶ `length(s)` – az s string hossza
- ▶ `log(kif)` – a kif logaritmusa
- ▶ `rand()` – 0 és 1 közötti véletlen számot ad vissza
- ▶ `srand([szám])` – véletlen generálás kezdőpontja
- ▶ `sin(kif)` – a kif szinusza
- ▶ `split(s,a,d)` – az s-t d elválasztójel szerint a[1]...a[n] tömbelemekre osztja, visszatérési értéke n
- ▶ `sprintf(fmt,...)` – a ...-ot fmt formátum string szerint formázva adja vissza
- ▶ `substr(s,m,n)` – az s string m-edik karaktertől kezdődő n karakteres része

# Saját függvény

- ▶ Szintakszisa:

```
function név (arg1, arg2, . . .)
```

```
{
```

```
    utasítások;
```

```
}
```

- ▶ Az Awk szabályokon kívül kell történjen a programban
- ▶ Nem szükséges őket használatuk előtt definiálni, ajánlott a program végére tenni ezeket
- ▶ Függvény hívásakor nem lehet szóköz a függvény neve és a zárójel között
- ▶ Argumentumok lokális változók
- ▶ A függvény belsejében létrehozott változók globálisak!!!
- ▶ Rekurzió lehetséges

# Függvény példa

```
#!/usr/bin/awk -f
BEGIN {
    for (i=0; i < 10; i++) {
        printf "%4d\t%4d\n", i,négyzet(i)
    }
    print "\nna főprogramból látható negy:" negy
}
function négyzet(i) {
negy=i*i
return negy
}
```

```
$ awk -f négyzet.awk
```

```
0      0
1      1
2      4
3      9
4      16
...
9      81
```

főprogramból látható negy: 81

# Folyamatok

- ▶ Process – az operációs rendszerben a futó program egy példánya
- ▶ Az rendszer tárolja többek között
  - Memóriaterület
  - Megnyitott állományok
  - Környezeti változók
- ▶ Párhuzamosan futnak – scheduler (ütemező) felváltva juttatja processzoridőhöz (time slice) a folyamatokat
- ▶ Folyamat tulajdonságai:
  - Saját védett memóriaazona
  - Rendelkezik az induláskor megnyitott három állománnyal
    - Input
    - Output
    - Error
  - A környezetét leíró változók kulcs érték párokban
  - Megkapja az indításkor beírt parancssorát
  - Prioritás van hozzá rendelve

# Folyamat állapotok

- ▶ Fut (Running) R – A folyamat fut, vagy nem fut, de futásra kész. Ilyenkor a folyamat arra vár, hogy processzoridőhöz jusson.
- ▶ Várakozik (Interruptible Sleep) S – A folyamat várakozik valamilyen eseményre vagy erőforrásra és megszakítható egy jelzés által.
- ▶ Eseményre vár (Uninterruptible Sleep) D – A folyamat várakozik valamilyen eseményre vagy erőforrásra és nem szakítható meg jelzés által.
- ▶ Fel van függesztve (Stopped,Traced) T – Fel van függesztve és áll. Terminál Ctrl-Z .
- ▶ Apátlan (Zombie) Z – Ha egy folyamat kilép, az apa folyamatnak át kell vennie a fiú kilépési állapot adatait. Amíg ezt nem teszi meg, a fiú un. apátlan állapotba kerül, bár már nem fut. Ha az apa ezt nem teszi meg, az init fogja átvenni helyette.

# Folyamat azonosító

- ▶ Process id – egész szám
- ▶ Kernel – 0 számú folyamat, ps nem listázza
- ▶ Az init folyamat indul el elsőként a rendszer indulásakor –1 számú
- ▶ A folyamok az init-től származnak
- ▶ pstree parancs – a folyamatok fa stukturálása
- ▶ A héj saját azonosítóját a \$\$-val érhetjük el.

# Job

- ▶ Egy parancssor által kiadott feladat a job
- ▶ jobs parancs – a végrehajtás alatt álló jobok
  - -p csak a folyamatazonosítót listázza
  - -1 minden listáz
  - -r csak a futó folyamatokat listázza
  - -s csak a leállított feladatokat (stopped) listázza
- ▶ előtérben és háttérben futtatás
- ▶ fg és bg parancsok
- ▶ & jel
- ▶ Az utoljára háttérben indított folyamat azonosítóját a \$!-el kaphatjuk vissza

# wait

- ▶ A futó folyamtoknak egy futó folyamat fiaként kell létezniük
- ▶ Az indító folyamatnak meg kell várnia a fiát
- ▶ Erre szolgál a wait parancs
  - wait – minden fiú folyamatra vár
  - wait n – az n azonosítójú folyamatra vár
  - wait %n – az n.-dik job folyamataira vár
- ▶ Végrehajtása alatt a shell nem indít új parancsot és nem olvassa a terminált sem

# top

- ▶ A kiugróan sok erőforrást igénylő folyamatok listázására alkalmas
- ▶ ? – segédletet listáz
- ▶ k – kill parancsot közvetít
- ▶ r – renice: prioritást változtat
- ▶ u – csak egy bizonyos felhasználó folyamatait mutatja
- ▶ q – kilép
- ▶ o – order: meg lehet adni interaktívan melyik oszlop szerint rendezzen

# Jelzések

- ▶ A folyamatok kommunikálnak egymással
  - Adatokat küldenek át egymáshoz (csővezeték, állomány)
  - Szinkronizációs adatokat küldenek összehangolási céllal
- ▶ Aszinkron kommunikáció – bármikor küldhető
- ▶ Signal – aszinkron értesítés, amelyet egy folyamat küld valamelyen esemény bekövetkezésekor
- ▶ Signal handler – üzenet kezelő. Ha üzenet érkezik, akkor a normál futás megszakad, és ez fut le
- ▶ Jelzéseknek általában létezik implicit kezelési sémája

# Jelzések

- ▶ Terminálról küldött jelzések
  - TSTP 20 – Felfüggeszti a folyamatot a terminálról. A Ctrl-Z karakter leütése után ez hajtódik végre.
  - INT 2 – Megszakítja és azonnal leállítja a folyamatot a terminálról, a Ctrl-C karakter leütése váltja ki.
  - QUIT 3 – Leállítás: a folyamat lezárhatja állományait, takaríthat de kilép (Ctrl-\ váltja ki a terminálról).

# Jelzések

- ▶ Folyamatokból küldött jelzések (parancssorról a kill parancsal):
  - KILL 9 – Feltétel nélkül azonnal leállítja a futó folyamatot, minden egy folyamat küldi folyamatnak. A KILL jelzést nem lehet kezelni.
  - ABRT 6 – Abort – leállítás, de előtte un. core dump (program memóriaképe) nyomtatása.
  - HUP 1 – Hang up: ha a folyamat interaktívan fut, és megszakad a kapcsolat a felügyelő terminállal, akkor ezt a jelzést kapja. De gyakori egy folyamat újraindítására való használata is (konfigurációs állományok újraolvasása, inicializálás).
  - TERM 15 – Szintén leállítást kér (terminate), egy másik folyamat küldi. Akárcsak a QUIT esetén, ajánlott, hogy leállás előtt a folyamat elvégezze takarítási feladatait. A legtöbb jelzés küldő parancs ezt küldi implicit jelzésként.
  - JSR1, USR2 10,12 – User: tetszés szerinti művelet programozására használt jelzések.
  - FPE 8 – Lebegőpontos művelet hiba, implicit kezelése a leállás.
  - STOP 19 – Program által küldött jelzés felfüggesztésre. Program által küldött jelzés felfüggesztésre.
  - CONT 18 – Újraindítja a STOP-al leállított programot.
  - CHLD 17 – Ezt a jelzést akkor kapja a folyamat ha egyik fiú folyamata kilép.