Operációs Rendszerek

Gyakorlati jegyzet

Összeállította: Rodek Lajos

Szegedi Tudományegyetem

Képfeldolgozás és Számítógépes Grafika Tanszék

A UNIX operációs rendszer

- A sok operációs rendszer közül csak a **UNIX** (ejtsd: juniksz) rendszerrel foglalkozunk.
- Két fő irányzat:
  - Első verzió: Kenneth Thompson és Dennis Ritchie, 1969., AT&T Bell Labs (AT&T UNIX). Ebből származik a manapság is használt **System V**.
  - Egy eltérő kezdeményezés: 1977., University of California, Berkeley (Berkeley Software Distribution – **BSD**). Kevésbé elterjedt, mint a System V.
- Rengeteg változata létezik (pl. AIX, HP-UX, SunOS, Solaris, IRIX, Xenix, Mac OS X, Minix, **GNU/Linux**)
- Nagy részét C nyelven, kisebb részét Assemblyben írták

A UNIX tulajdonságai I.

- Többfelhasználós (multiuser):
  - különböző felhasználók adatainak, beállításainak nyilvántartása, privát munkaterület biztosítása
  - több felhasználó is dolgozhat egy számítógépen ugyanabban az időben
- Többfeladatos (multitask): több feladat (program) futhat egy időben
- Számítógép-hálózatok kiterjedt támogatása: pl.
  - kommunikáció más számítógépekkel
  - állományrendszer elérése hálózaton keresztül
  - a grafikus felületet is lehet hálózaton keresztül használni

A UNIX tulajdonságai II.

- Biztonságos:
  - felhasználók jelszavas beléptetése
  - kritikus műveletek végrehajtása korlátozható
  - privát hozzáférési jogok
  - felhasználói tevékenység naplózása
- Stabil, rugalmas állományrendszer
- Rengeteg apró segédprogram
- Hatékonyan programozható
- Több architektúra támogatása (multiplatform): egymástól eltérő architektúrákra különféle változatai jelentek meg

A UNIX felépítése

- A rendszer elemei:
  1. hardver: maga a számítógép
  2. mag (**kernel**): Az operációs rendszer lényegi része. Feladata az erőforrások (memória, processzor, háttértár, perifériák) kezelése, felügyelete és kiosztása, a programok futtatása, az állományrendszer karbantartása, stb.
  3. segédprogramok, **shell**: Alapvető szolgáltatások biztosítása. Kiemelten fontosak a shell programok (parancsértelmezők).
  4. alkalmazások: mindenféle egyéb program
- A UNIX-ot alapvetően a 2. és 3. pontban említettek alkotják.
- A **felhasználók** (user) **csoportokba** (group) vannak besorolva.
- Egy kiemelt felhasználó van: `root`, ő a rendszergazda (system administrator, supervisor, superuser)
- A `root` felhasználó korlátozás nélkül bármit megtehet, ennek használatával tehát vigyázni kell.

A GNU/Linux operációs rendszer

- A Minix-et túlszárnyaló, UNIX-szerű (System V alapú) operációs rendszer
- A Linux csak a kernel neve. Az op. rendszert GNU/Linux-nak hívják.
- GNU (GNU's Not UNIX): a Free Software Foundation által indított projekt
- Első változat (PC-re): Linus Torvalds, 1991., University of Helsinki
- Több változatban (disztribúcióban) is megjelent, pl. RedHat, Debian, SuSE, Mandrake, Slackware, UHU, Caldera OpenLinux
- Nyílt forráskódú, így sok változata ingyenes
- Több architektúrán is fut:
  - Intel x86, AMD x86 és x86-64 (IBM PC)
  - Motorola m68k és PowerPC (Apple Macintosh, Amiga)
  - Compaq/Digital Alpha
  - Sun Sparc
  - beágyazott rendszerek (pl. mobiltelefonok)
- Bővebben: http://www.linux.org/, http://www.fsf.org/

Parancsok használata, segítségkérés

- Minden segédprogram (parancs) használata azonos módon történik:

  PARANCS `-`*OPCIÓK* `--`*OPCIÓ PARAMÉTEREK*

- Mind a parancsok nevénél, mind az opcióknál különbözőnek számítanak a kisbetűk és a nagybetűk!
- A `-` után egybetűs opciók állhatnak (több is), míg a `--` egyetlen többetűs (beszédes) opció kezdetét jelzi. Mindkétfajta opció megismételhető.
- Néhány opció külön paraméter(ek) megadását is igényelheti.
- Segítségkérés a legtöbb programnál:
  - `-?`, `-h`
  - `--help`
- Beépített dokumentáció, segítség (manual, help):
  - `man` *PARANCS*
  - `info` *PARANCS*
  - önmagukról is adnak leírást: `man man`, `man info`, `info info`

A `man` parancs

- A szöveg megjelenítését igazából egy másik program (`more`, `less`) végzi.
- Hasznos billentyűk:
  - h: segítség a használható billentyűkről
  - szóköz (SPACE), PAGE DOWN: előre egy képernyőnyit
  - b, PAGE UP: vissza egy képernyőnyit
  - FEL, LE: mozgás vissza-előre egy sorral
  - g: ugrás az első sorra
  - G (SHIFT+g): ugrás az utolsó sorra
  - /: szöveg keresése
  - n: keresés folytatása (előző szöveggel)
  - q: kilépés

Az `info` parancs

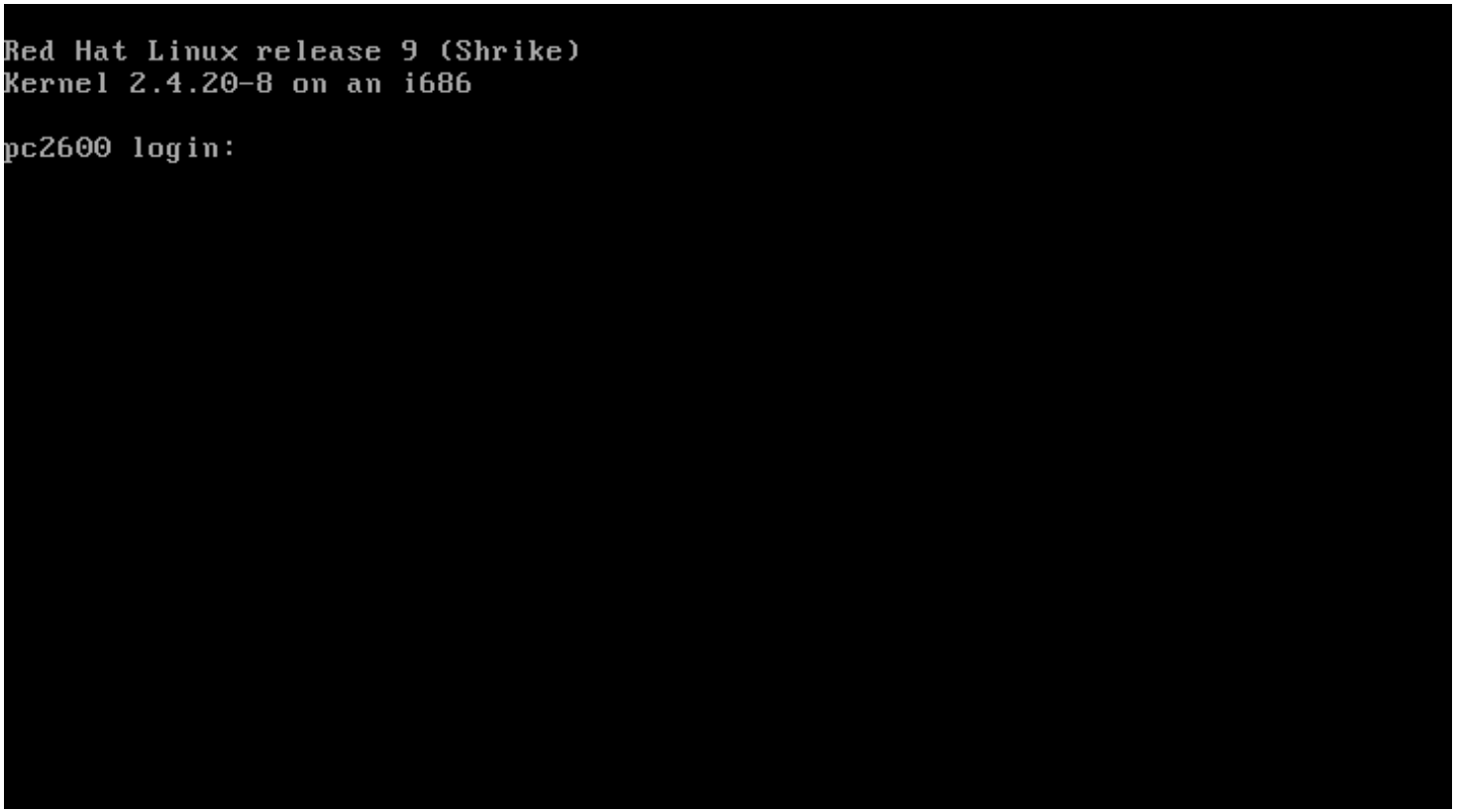
- Hasznos billentyűk:
  - ?: segítség a használható billentyűkről
  - h: oktató leírás a program használatáról
  - szóköz (SPACE), PAGE DOWN: előre egy képernyőnyit
  - BACKSPACE, DEL, PAGE UP: vissza egy képernyőnyit
  - FEL, LE: mozgás vissza-előre egy sorral
  - b: ugrás az első sorra
  - p: ugrás a megelőző témára
  - n: ugrás a következő témára
  - u: ugrás egy szinttel feljebb
  - l („ell”): visszatérés a legutóbbi témára
  - q: kilépés

A Linux használata

- Grafikus és szöveges felületen (ún. **virtuális terminálokon**) keresztül is használható
- Parancsok kiadására használhatjuk:
  - a szöveges módot
  - grafikus módban az ún. **terminál emulációs programot** (ld. később)
- Átváltás grafikus módból szövegesbe: CTRL+ALT+F1, …, CTRL+ALT+F6 (a megadott sorszámú szöveges terminálra)
- Átváltás szöveges módból grafikusba: ALT+F7, …, ALT+F11 (valamelyik)
- Kilépés:
  - `exit`
  - CTRL+D
- Fontos, hogy ha mindkét módban (avagy több szöveges terminálon) be vagyunk jelentkezve, akkor külön-külön ki kell lépnünk minden helyről!

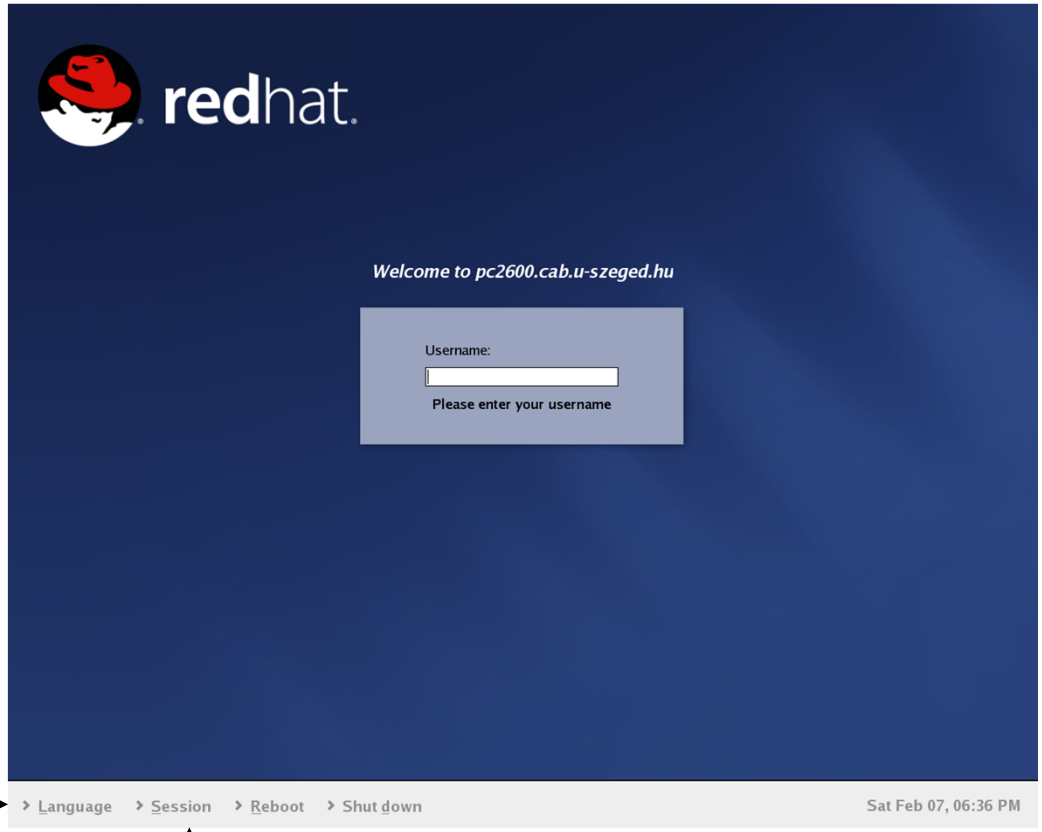
A szöveges mód használata

- 6 egymástól független szöveges ablak (virtuális terminál) áll rendelkezésre
- Átváltás a szöveges terminálok között: ALT+F1, …, ALT+F6



A grafikus mód használata

- Ez is virtuális terminálnak számít (alapesetben a 7. terminál)

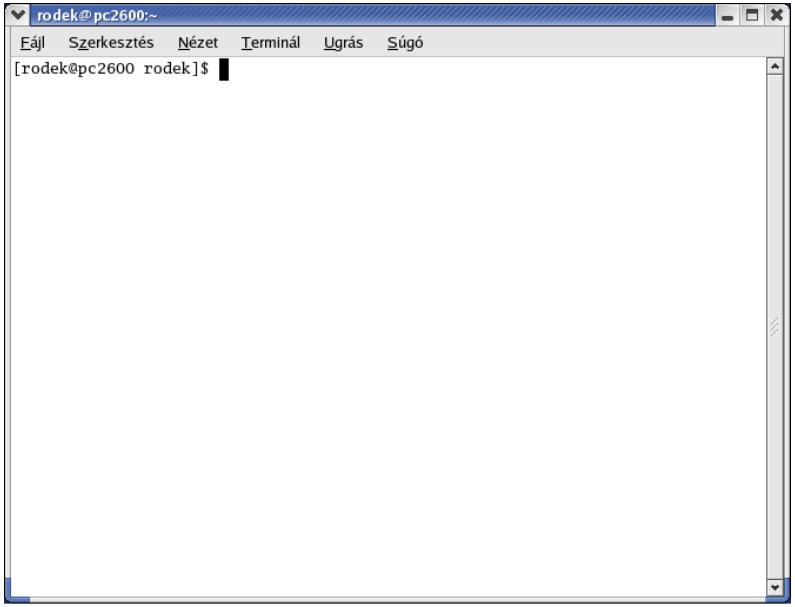


nyelv kiválasztása

grafikus felület kiválasztása (KDE, Gnome)

A terminál emulációs program

- Segítségével ugyanúgy hajthatunk végre parancsokat, mint szöveges módban.
- A terminál emulációs programok neve:
  - `xterm` (nem javasolt)
  - `konsole`
  - `gnome-terminal`
- A végrehajtani kívánt parancsot a **parancssorba** írhatjuk be. Ennek elején, a kurzor előtt látható a dollárjelre végződő **prompt**, ami általában a felhasználó azonosítóját, a számítógép nevét és az aktuális könyvtárat mutatja.

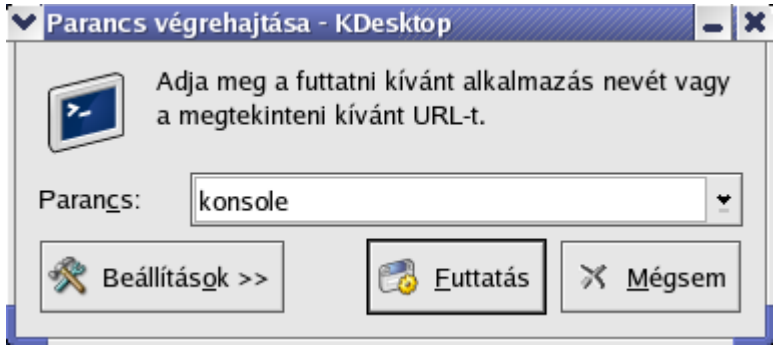


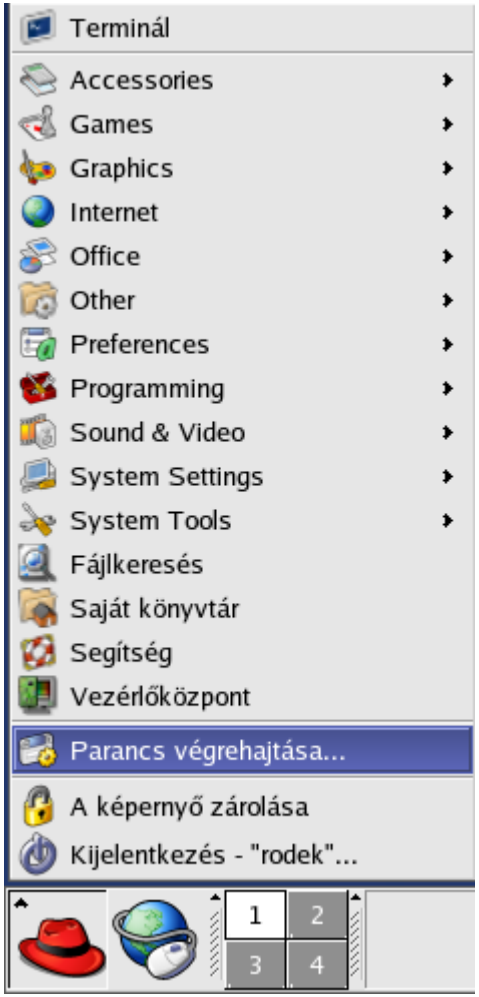
Terminál indítása a KDE grafikus felületen I.

1. A bal alsó sarokban nyíló menüben a „System Tools” / „Terminál” bejegyzést választva

Terminál indítása a KDE grafikus felületen II.

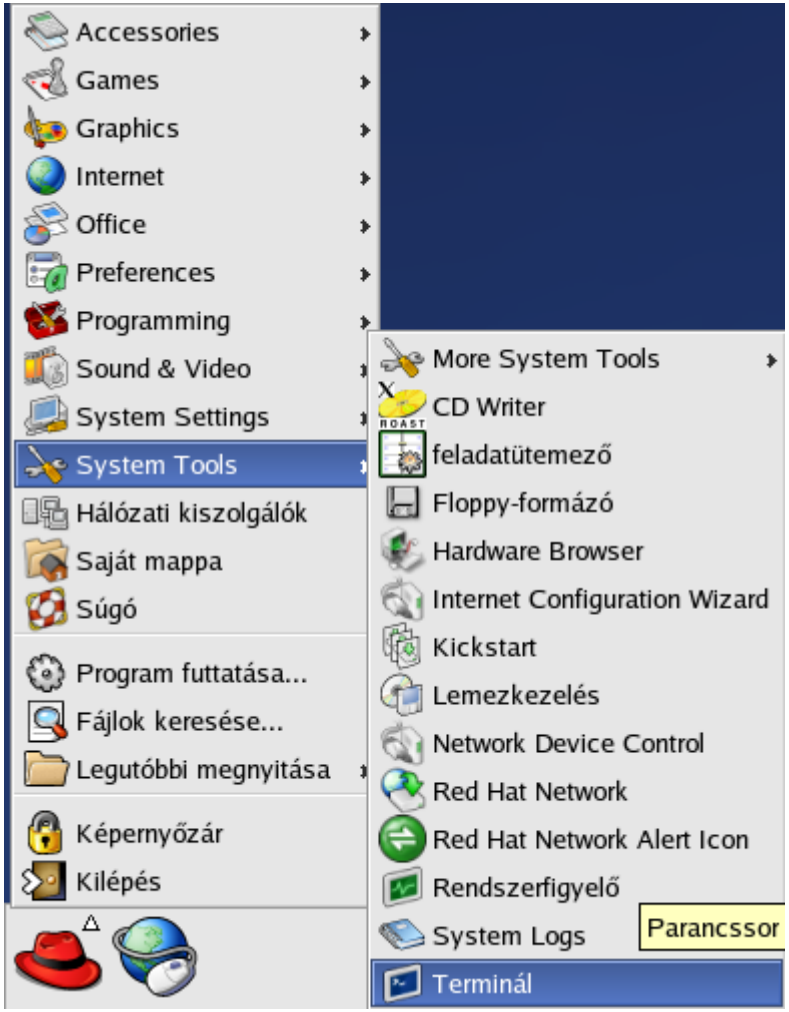
2. A bal alsó sarokban nyíló menüben a „Parancs végrehajtása…” bejegyzést választva, majd a megjelenő párbeszédablakba a program nevét beírva





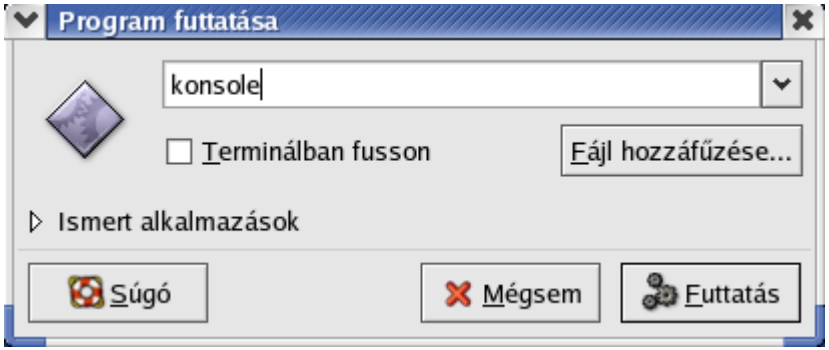
Terminál indítása a Gnome grafikus felületen I.

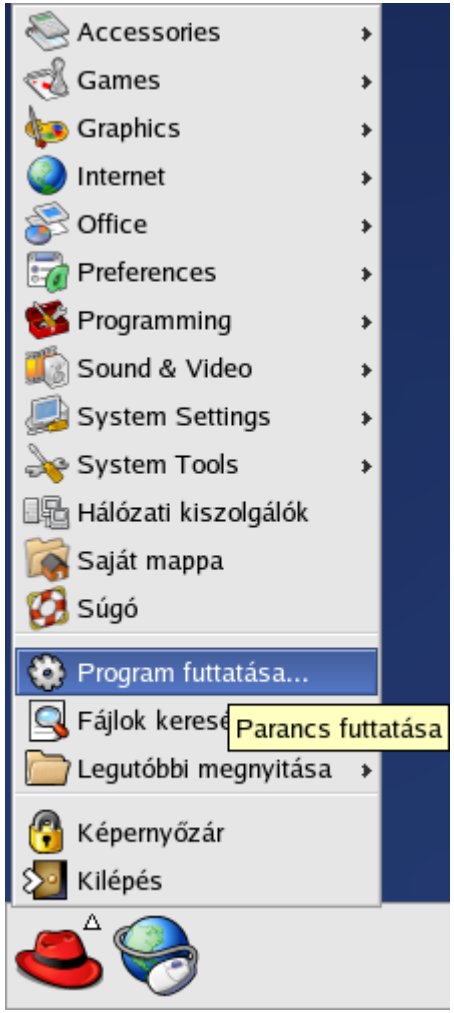
1. A bal alsó sarokban nyíló menüben a „System Tools” / „Terminál” bejegyzést választva



Terminál indítása a Gnome grafikus felületen II.

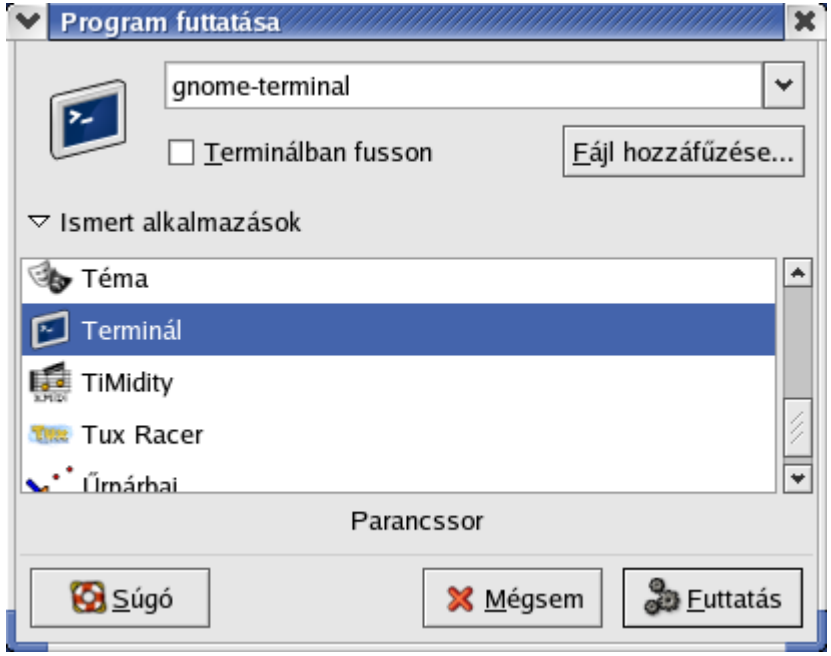
2. A bal alsó sarokban nyíló menüben a „Program futtatása…” bejegyzést választva, majd a megjelenő párbeszédablakba a program nevét beírva

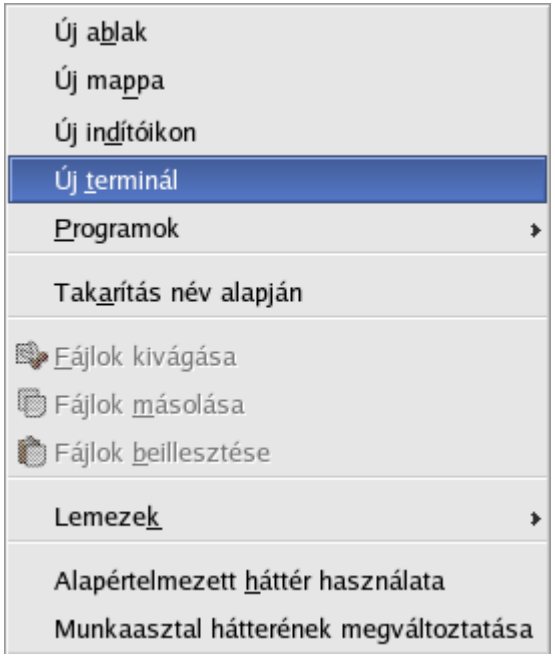




Terminál indítása a Gnome grafikus felületen III.

3. Mint előbb, de a párbeszédablak „Ismert alkalmazások” listáját lenyitva, majd ott a „Terminál” bejegyzést kiválasztva
4. A munkaasztalon a jobb egérgombbal kattintva felugró menüben az „Új terminál” bejegyzést választva





Az állományrendszer felépítése

- Az operációs rendszerek a különféle, összetartozó adatokat **állományokban** avagy **fájlokban** (file) tárolják.
- A UNIX állományok **típusa**:
  - közönséges: struktúrálatlan bájtsorozat
  - speciális: meghatározott szerkezetű, különleges célú
    - **katalógus**, **jegyzék** avagy **könyvtár** (directory)
    - eszköz (device)
    - szimbolikus lánc (symbolic link)
    - nevesített FIFO cső (named pipe, FIFO)
    - kommunikációs végpont (socket)
- Az állományok hierarchikusan (többszintű fastruktúrában) csoportosítva, könyvtárakban vannak elhelyezve. Mindegyik könyvtár tartalmazhat bármilyen állományt, akár újabb könyvtárat is (ezeket hívjuk **alkönyvtáraknak**). Az alkönyvtárat tartalmazó könyvtárat **szülőnek** nevezzük.

Állományok tulajdonságai

- Méret: Van felső korlátja, de ez az adott állományrendszertől függ.
- Típus (ld. előző dia)
- Név: Szinte bármilyen karaktert tartalmazhat (a kivételeket ld. később a shellnél), hossza általában legfeljebb 255 karakter lehet. A kisbetűk és a nagybetűk különbözőnek számítanak!
- Ha a név ponttal (`.`) kezdődik, **rejtett állományról** ill. **rejtett könyvtárról** beszélünk (ld. később az `ls` parancsnál).
- **Tulajdonos**:
  - tulajdonos felhasználó (owner, owner user): megváltoztatás a `chown` paranccsal
  - felhasználói csoport (group): megváltoztatás a `chgrp` paranccsal
- Létrehozás, utolsó hozzáférés ill. utolsó módosítás dátuma és ideje
- **Hozzáférési jogok** (access permissions/mode): megváltoztatás a `chmod` paranccsal (ld. később), de befolyásolja az `umask` parancs is

Hozzáférési jogok

- Jogok:
  - **Olvasási jog** (read permission): az állomány olvasható, ill. a könyvtár tartalma listázható
  - **Írási jog** (write permission): az állomány módosítható, ill. a könyvtárban állományokat lehet létrehozni és törölni
  - **Végrehajtási** avagy **futtatási jog** (execute permission): az állomány programként végrehajtható, ill. a könyvtárban levő állományok/ könyvtárak hozzáférhetőek, be lehet lépni a könyvtárba
  - Létezik még 3 speciális jog is, de ezek számunkra nem fontosak.
- Az előbbi jogok a felhasználók 3 részhalmazára adhatók meg:
  - a fájl tulajdonosának (owner, owner user)
  - a fájl csoportjának (group)
  - mindenki másnak (other users)

A chmod parancs I.

- chmod *JOG ÚTVONAL(AK)*:
  - a megadott állomány(ok) ill. könyvtár(ak) hozzáférési jogainak módosítása
  - -R: a módosítást a megadott könyvtár(ak) összes állományán és az alkönyvtárak teljes tartalmán elvégzi (a jogok rekurzív módosítása)
  - A *JOG* szóközöket *nem tartalmazó* egyetlen szó, és kétféle alakban adható meg: szimbolikus és numerikus alakban.
  - Szimbolikus alak:
    - A *JOG* szerkezete ilyenkor *FELHASZNÁLÓ MŰVELET JOGOK* (persze a szóközök nélkül).
    - *FELHASZNÁLÓ*: u: tulajdonos, g: csoport, o: mindenki más, a: az előző három egyszerre (=ugo). Több betűt is megadhatunk, a sorrend pedig nem számít.
    - *MŰVELET*: +: *JOGOK* engedélyezése a *FELHASZNÁLÓ*-nak, -: *JOGOK* tiltása a *FELHASZNÁLÓ*-nak, =: a *FELHASZNÁLÓ* csak a megadott *JOGOK*-kal fog rendelkezni.

A chmod parancs II.

- *JOGOK*: r: olvasási jog, w: írási jog, x: végrehajtási jog, X: feltételes végrehajtási jog (a végrehajtási jog csak akkor módosul, ha könyvtárról van szó, vagy ha az állomány amúgy is végrehajtható volt). Több betűt is megadhatunk, és a sorrend itt sem számít.
- A *FELHASZNÁLÓ* és a *JOGOK* rész is elhagyható. Előbbi esetben majdnem olyan, mintha a lett volna megadva (az eltérés az umask paranccsal kapcsolatos). A *JOGOK* elhagyásának pedig az = művelet használata esetén van értelme, ugyanis így a *FELHASZNÁLÓ*-nak semmilyen joga sem lesz.

– Numerikus alak:

- A *JOG* ilyenkor egy háromjegyű szám, ahol a jegyek a tulajdonos (első jegy), a csoport (második jegy), ill. mindenki más (harmadik jegy) jogait adják meg abszolút módon. A bevezető nullák elhagyhatók.
- Minden jegy egy 0 és 7 közötti számjegy, amely a következő számok összegeként áll elő: 0: üres, 1: végrehajtási jog, 2: írási jog, 4: olvasási jog. Mindegyik tag legfeljebb egyszer szerepelhet az összegben!

Elérési utak

- Egy **elérési út** vagy **útvonal** (path) egy konkrét állomány/könyvtár helyét adja meg az állományrendszerben.
- Az elérési útban előforduló könyvtárak neveit ill. az esetleg a végén álló állomány nevét a `/` (slash) jel választja el. Ez a jel akkor is kiírható az elérési út végére, ha az állománynév elmarad.
- Speciális elérési utak:
  - `/`: **gyökérkönyvtár** (root directory), az állományrendszerben „legfelül” elhelyezkedő könyvtár (az összes könyvtár őse)
  - `~` (tilde): az aktuális felhasználó saját könyvtára (**home directory**)
  - `~`*`FELHASZNÁLÓ`*: a megadott felhasználó saját könyvtára
  - `.` (pont): **aktuális könyvtár**, munkakönyvtár (working directory)
  - `..` (dupla pont): az aktuális könyvtár szülő könyvtára (parent directory)
- **Abszolút elérési út**: a gyökérhez (`/`) képest megadott hely
- **Relatív elérési út**: az aktuális könyvtárhoz (`.`) képest megadott hely
- Egy elérési út mindig relatív, ha nem a `/` vagy `~` jelekkel kezdődik.

Fontosabb rendszerkönyvtárak

- Leírás: `man 7 hier`
- `/boot`: az operációs rendszer elindulásához szükséges
- `/bin`, `/sbin`, `/usr/bin`, `/usr/sbin`: futtatható állományok gyűjtőhelye
- `/dev`: eszközállományokat tartalmaz
- `/etc`: adminisztrációs állományok, kritikus beállítások
- `/home`: a felhasználói könyvtárakat tartalmazza
- `/lib`: programok által használt függvénykönyvárakat tartalmaz
- `/mail`: az elektronikus levelezéshez
- `/mnt`: külső állományrendszerek gyűjtőhelye
- `/opt`, `/var`: vegyes beállítások, adatok, programok
- `/tmp`: ideiglenesen létrehozott állományok
- `/usr`: felhasználók által elérhető közös adatok, információk, programok

Az állományrendszer fizikai szervezése

- Boot block (nulladik blokk): az ebben levő rövid program tölti be a UNIX-ot
- Superblock (első blokk): az állományrendszer részleteit és a belső táblák adatait tartalmazza
- Inode tábla: az inode-ok adatait tartalmazza
- Az **inode** (index node) egy adott állomány minden fontos adatát tartalmazza: méretet, típust, tulajdonost, a hozzáférési jogokat, a háromféle dátumot, az állományhoz tartozó lemezblokkok sorszámait, valamint a merev láncok számát avagy a **láncszámot** (ld. később).
- Szigorúan véve az inode-okat azonosíthatjuk az állományokkal.
- Minden inode egyedi sorszámot kap.
- Minden könyvtárhoz tartozik egy állomány. Ez a speciális állomány tartalmazza a könyvtárban levő állományok nevét és inode-számát.
- A szimbolikus láncok (ld. később) olyan speciális állományra mutatnak, amelyek a célállomány (eredeti állomány) nevét tartalmazzák.

Munka állományokkal, könyvtárakkal I.

- Gyakran használt parancsok:

	Állományok	*Könyvtárak*
Váltás		`cd`
Létrehozás	`cat`, `touch`, `echo`, szövegszerkesztők	`mkdir`
Másolás	`cp`	`cp`
Átnevezés, mozgatás	`mv`	`mv`
Törlés	`rm`	`rmdir`
Listázás, megjelenítés	`cat`, `less`, `more`, `od`, `xd`	`ls`
Egyéb	`file`, `touch`, `basename`, `dirname`	`du`, `pwd`, `touch`, `basename`, `dirname`

Munka állományokkal, könyvtárakkal II.

- `cd`: az aktuális könyvtár (`.`) beállítása (alapesetben a `~` könyvtárra)
- `pico`, `joe`, `xedit`, `nedit`, `xemacs`: szövegszerkesztők
- `mc` (Midnight Commander): Segédprogram az állományokkal és könyvtárakkal való munkához. Tartalmaz egy szövegszerkesztőt is.
- `cat > `*`ÁLLOMÁNY`*: Új állomány létrehozása. Az állomány tartalma a parancs kiadása után begépelt (akár többsoros) szöveg lesz. A szöveget a CTRL+D billentyű-kombinációval kell lezárni.
- `echo '`*`SZÖVEG`*`' > `*`ÁLLOMÁNY`*: Új állomány létrehozása a megadott szöveggel mint tartalommal. (Hogy miért kellenek az aposztrófok, arra később a shellnél lesz magyarázat.)
- `touch `*`NÉV`*:
  - Új állomány létrehozása üresen, ha az még nem létezik.
  - Létező állomány vagy könyvtár utolsó elérési és utolsó módosítási dátumának/idejének beállítása az aktuálisra.
- `mkdir`: új könyvtár létrehozása

Munka állományokkal, könyvtárakkal III.

- `cp` *FORRÁS CÉL*:
  - állomány másolása (alapesetben könyvtárakat nem másol)
  - `-R`, `-r`: a megadott könyvtár(ak) minden állományának és az alkönyvtárak teljes tartalmának átmásolása (rekurzív másolás)
- `mv`: állomány vagy könyvtár átnevezése vagy új helyre mozgatása (áthelyezése)
- `rm`:
  - állomány törlése (alapesetben könyvtárakat nem töröl)
  - A törlés minden esetben végleges (nem vonható vissza)!
  - `-f`: rákérdezés nélkül töröl
  - `-R`, `-r`: a megadott könyvtár(ak) minden állományának és az alkönyvtárak teljes tartalmának törlése (rekurzív törlés)
- `rmdir`: üres könyvtár törlése
- `cat`: állomány tartalmának kiírása
- `less`, `more`: állomány tartalmának listázása lapozhatóan

Munka állományokkal, könyvtárakkal IV.

- `od`, `xd`: állomány tartalmának listázása (dump) nyolcas (oktális) vagy tizenhatos (hexadecimális) számrendszerben
- `ls`: könyvtár tartalmának listázása (ld. következő dia)
- `file`: állománytípus megállapítása tartalom alapján
- `du`:
  - a lemezen használt terület kiírása 512 bájtos blokkokban
  - `-k`: ugyanez, de 1 kilobájtos egységekben
- `pwd`: az aktuális könyvtár (`.`) nevének (abszolút elérési útjának) kiírása
- `basename` *`ÚTVONAL`*: A könyvtárak neveit eltávolítja a megadott útvonalból (csak az utolsó `/` utáni állománynév marad meg), majd kiírja az eredményt. Nem ellenőrzi az útvonal valódiságát!
- `dirname` *`ÚTVONAL`*: Az állomány nevét eltávolítja a megadott útvonalból (csak az utolsó `/` előtt álló könyvtárak listája marad meg), majd kiírja az eredményt. Ha az útvonal nem tartalmaz `/` jelet, az eredmény a `.` lesz. Nem ellenőrzi az útvonal valódiságát!

Az `ls` parancs I.

- `ls` *`ÚTVONAL(AK)`*:
  - a megadott állomány(ok) jellemzőinek kiírása növekvő ábécé sorrendben
  - jellemzők: alapesetben csak a név
  - Ha könyvtárat adtunk meg, akkor a könyvtárban levő állományok jellemzőit írja ki. A rejtett állományok alapesetben kimaradnak a listából.
  - Ha nem adunk meg útvonalat, akkor az aktuális könyvtár (`.`) tartalmát listázza ki.
  - Több könyvtár megadása esetén, vagy ha állományt és könyvtárat is megadtunk, a könyvtárlista elé egy fejlécsor („*`KÖNYVTÁR`*`:`”) is kiíródik, valamint a listákat egy-egy üres sor fogja elválasztani.
  - `-1` („egy”): minden sorban csak egy név látszik (egyoszlopos mód)
  - `-a`: a listában a rejtett állományok/könyvtárak is megjelennek
  - `-C`: minden sorban több név látszik (többoszlopos mód)
  - `-d`: könyvtár megadása esetén a könyvtárnak mint speciális állománynak a jellemzőit írja ki (nem pedig a könyvtár tartalmát)
  - `-l` („ell”): hosszú avagy bővített listát készít (ld. később)

Az `ls` parancs II.

- `-R`: a megadott könyvtár(ak) minden alkönyvtárának és azok teljes tartalmának listázása (rekurzív listázás)
- `-r`: csökkenő sorrend
- A lista formája:
  - Az `-l` („ell”) opció használata esetén minden sor csak egy bejegyzés jellemzőit tartalmazza (ld. következő dia).
  - Különben a parancs kimenete kétféle alakot ölthet: minden sorban egy vagy több név is kiíródhat. Hogy melyiket alkalmazza, azt az `-1` („egy”) és `-C` opciók határozzák meg. (Értelemszerűen ez a két opció kölcsönösen kizárja egymást.)
  - Ha egyik említett opciót sem adtuk meg, akkor a kimenet többoszlopos lesz, ha a szabványos kimenet (ld. később) a képernyő. Ellenkező esetben – tehát ha a kimenetet átirányítottuk, vagy a parancs csővezetékbe van kötve – az egyoszlopos mód lép érvénybe.

Az `ls -l` parancs I.

- Bővített listázás (a néven kívül egyéb információkat is megjelenít)
- Minden sor egy állomány vagy alkönyvtár adatait mutatja 9 oszlopban (az oszlopokat szóközök tagolják):
  1. állománytípus, hozzáférési jogok
  2. merev láncok száma (láncszám) állományoknál (ld. később); alkönyvtárak száma könyvtáraknál (a `.` és `..` könyvtárakat is beleértve)
  3. tulajdonos felhasználó
  4. tulajdonos csoport
  5. méret bájtokban

  6-8. utolsó módosítás dátuma és időpontja (hónap, nap, év/időpont)

  9. név, szimbolikus lánc neve (ld. később)
- Könyvtárak listázása esetén a legelső bejegyzés előtt egy „`total` *N*” („`összesen` *N*”) tartalmú sor szerepel, ahol *N* a kiírt bejegyzések által a lemezen elfoglalt hely kilobájtokban. Minden könyvtárra újabb ilyen sor íródik ki.

Az `ls -l` parancs II.

- Az állománytípus és a hozzáférési jogok egy 10 karakteres szóval vannak ábrázolva:
  1. típus (`-`: közönséges, `c`: karakteres eszköz, `b`: blokkos eszköz, `d`: könyvtár, `l`: szimbolikus lánc, `p`: FIFO cső, `s`: kommunikációs végpont)

  2., 5., 8. olvasási jog a tulajdonosnak, a csoportnak, ill. mindenki másnak (`-`: tiltott, `r`: engedélyezett)

  3., 6., 9. írási jog a tulajdonosnak, a csoportnak, ill. mindenki másnak (`-`: tiltott, `w`: engedélyezett)

  4., 7., 10. végrehajtási jog a tulajdonosnak, a csoportnak, ill. mindenki másnak (`-`: tiltott, `x`: engedélyezett)

Állomány- és könyvtárnevek megadása I.

- Hasonló felépítésű állomány- vagy könyvtárnevek listájának megadására használhatunk ún. **állománynév mintákat** (filename pattern). Ezek a közönséges karakterek mellett helyettesítő, mintaillesztő avagy Joker-karaktereket is tartalmaznak.
- Eredmény: a mintának megfelelő (mintára illeszkedő) *létező* nevek szóközökkel tagolt rendezett listája
- **Mintaillesztő karakterek**:
  - `*`: tetszőleges karakterekből álló, tetszőlegesen hosszú szó (üres szó is)
  - `?`: egyetlen tetszőleges karakter
  - `[`*`HALMAZ`*`]`: A halmaz bármely karakterének egy példánya. A halmazt a karakterek egymás mellé írásával adhatjuk meg.
  - `[`*`ELSŐ`*`-`*`UTOLSÓ`*`]`: mint előbb, de itt egy tartományt adunk meg
  - `[^`*`HALMAZ`*`]`: a halmazban *nem szereplő* bármely karakter egy példánya

Állomány- és könyvtárnevek megadása II.

- Speciális esetek:
  - Mindig ki kell írni a rejtett állományok/könyvtárak nevének kezdő pont (`.`) karakterét, ill. könyvtárak esetén a könyvtárnév után a `/` jelet.
  - A pont karakter egyéb esetekben nem számít speciálisnak. Néhány program azonban az állománynevekben az utolsó pont utáni részt, az ún. **kiterjesztést** (filename extension) különlegesen kezeli. Ezt általában az állomány tartalma típusának jelzésére használják (pl. kép, video, hang).
- Példák:
  - `*`: az összes nem rejtett állomány és alkönyvtár
  - `*/`: az összes nem rejtett alkönyvtár
  - `*/*`: az összes nem rejtett alkönyvtár teljes tartalma
  - `.*`: az összes rejtett állomány és alkönyvtár
  - `.*/`: az összes rejtett alkönyvtár
  - `*.jpg`: a `.jpg` kiterjesztésű állományok (JPEG formátumú képek)
  - `*.*`: az összes nem rejtett állomány és alkönyvtár, amelynek neve tartalmaz legalább egy pontot

Állomány- és könyvtárnevek megadása III.

- A hosszabb nevek begépelését könnyíti meg az **állománynév-kiegészítés** (filename completion). A név első pár betűjének beírása után üssük le a TAB billentyűt. Ha csak egy állomány neve kezdődik így, akkor a név kiegészül. Különben még egyszer üssük le a TAB-ot, hogy egy listát kapjunk a szóba jöhető nevekről. Ezután folytassuk a gépelést a kívánt karakterrel. Ez a szolgáltatás könyvtár- és programneveknél is működik.

Eszközök I.

- Minden hardvereszköz (periféria), ill. néhány szoftveres erőforrás ún. **eszközállományokon** (device) keresztül érhető el. Az ilyen állományok olvasása vagy írása közvetlenül az adott eszköz elérését fogja jelenteni.
- Eszközök típusai:
  - blokkos eszközök (block device): floppy, merevlemez, CD-ROM, pendrive
  - karakteres eszközök (character device): terminál, nyomtató, egér, szalagos egység, hangkártya
- Példák:
  - `/dev/null`: minden bele írt adatot elnyel („szemetesláda”)
  - `/dev/stdin`, `/dev/stdout`, `/dev/stderr`: szabványos bemenet és kimenetek az aktuális program esetén (ld. később)
  - `/dev/tty`: az éppen használt virtuális terminál

Eszközök II.

- `tty`: az éppen használt virtuális terminál nevének kiírása
- `tset`: terminál alaphelyzetbe hozása, jellemzőinek beállítása és lekérdezése
- `stty`: terminál jellemzőinek beállítása és lekérdezése
- `chvt` *SZÁM*: átváltás a megadott sorszámú virtuális terminálra
- `mount`: külső állományrendszer felcsatolása (bekötése) a jelenlegi állományrendszerbe, ill. a felcsatolt állományrendszerek nevének kilistázása
- `umount`: felcsatolt állományrendszer leválasztása
- `df`: a felcsatolt állományrendszerek szabad tárolóterületének kiírása
- `mknod`: eszközállomány vagy nevesített FIFO cső létrehozása

Láncolás I.

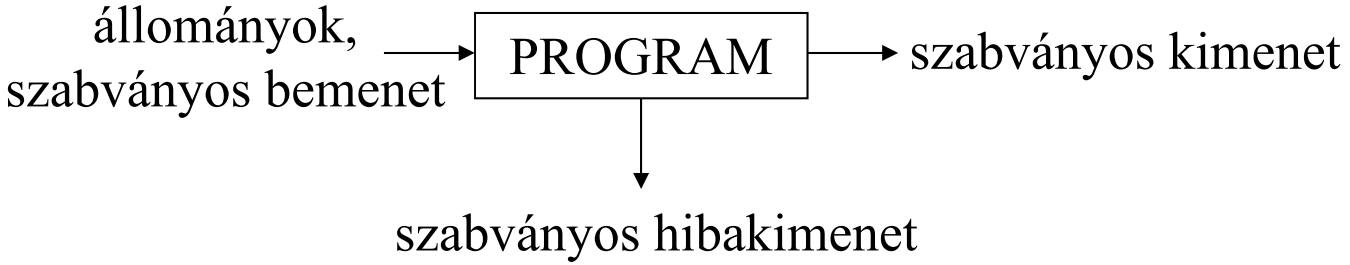
- Az állományrendszer lehetővé teszi, hogy ugyanazt az állományt több néven is elérhessük. Ezt ún. **láncok** avagy **láncszemek** (link) létrehozásával érhetjük el. Ezek olyan új állományok („másolatok”), amelyek az eredeti állományra mutatnak.
- Az eredeti állomány tartalmának megváltozásakor a láncok tartalma is változni fog.
- Két típusuk van:
  - **Merev lánc** (hard link):
    - Megkülönböztethetetlen és független az eredeti állománytól, mert mindkettő ugyanarra az inode-ra mutat.
    - Az `ls -l` parancs által kiírt láncszám értéke az eredeti állománynál és a láncnál is eggyel növekszik.
    - Az eredeti állomány a lánctól függetlenül törölhető, és viszont. Törléskor eggyel csökken a láncszám.
    - Nem használható könyvtárakra és más állományrendszerben elhelyezkedő állományokra.

Láncolás II.

- **Lágy** avagy **szimbolikus lánc** (soft/symbolic link):
  - Egy speciális állomány, amit az `ls -l` parancs `l` típusúnak mutat.
  - Az `ls -l` által kiírt állománynév ilyenkor „*LÁNC* `->` *EREDETI*” alakú (ez tehát plusz 2 oszlopot jelent).
  - A láncszám értéke az eredeti állománynál és a láncnál is változatlan marad.
  - A legtöbb művelet a lánc helyett az eredeti állományon hajtódik végre, kivéve pl. az `mv` és `rm` parancsokat.
  - Magának a szimbolikus láncnak a hozzáférési jogait nem lehet módosítani, mivel mindig az eredeti állomány jogai számítanak.
  - Az eredeti állomány törlésekor a lánc megmarad, de érvénytelenné válik (tehát ilyen szempontból függ az eredeti állománytól).
  - Bármilyen állományra és könyvtárra használható.
- `ln` *EREDETI LÁNC*:
  - merev lánc létrehozása
  - `-s`: szimbolikus lánc létrehozása

A programok kapcsolata a külvilággal

- Minden program rendelkezik egy bemenettel és kettő kimenettel:



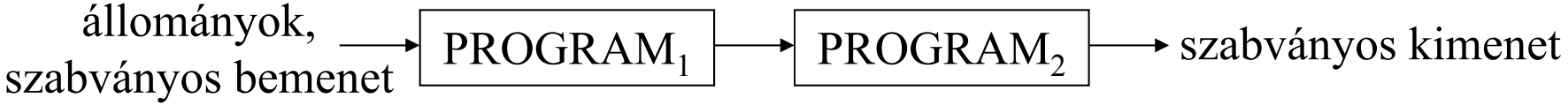
- Ha bemenetként nincs állomány megadva, akkor a program a **szabványos bemenetről** (standard input, **stdin**) olvas. Ez alapesetben a billentyűzet.
- A program által produkált látható eredmény a **szabványos kimenetre** (standard output, **stdout**) íródik ki. Ez alapesetben a képernyő.
- A hibaüzenetek a **szabványos hibakimenetre** (standard error output, **stderr**) lesznek kiírva. Ez alapesetben ugyancsak a képernyő.
- A billentyűzet és a képernyő együtt alkotják a **terminált**.
- Szabványos bemenet esetén a **bemenet** avagy az **adatbevitel végének** (end of stream) jelzése: CTRL+D

Átirányítás

- Mind a bemenet, mind pedig a két kimenet átirányítható egy tetszőleges állományba.
- Az **átirányítás** (redirection) a program számára teljesen átlátszóan történik.
- Az átirányítás jelöléseit a program utolsó paramétere után kell feltüntetni.
- Több átirányítás esetén azok végrehajtása balról jobbra történik.
- `<` *ÁLLOMÁNY*: stdin átirányítása (a megadott fájlból olvas)
- `>` *ÁLLOMÁNY*: stdout átirányítása (a megadott fájlba ír, a létező állomány felülírásával)
- `>>` *ÁLLOMÁNY*: stdout átirányítása (a megadott fájlba ír, a létező állomány végéhez való hozzáfűzéssel)
- `2>` *ÁLLOMÁNY*: stderr átirányítása (a megadott fájlba írja a hibaüzeneteket)
- `&>` *ÁLLOMÁNY*: stdout és stderr átirányítása ugyanabba a fájlba
- `2>&1`: a stderr-t ugyanoda irányítja, ahová a stdout irányítva lett
- `1>&2`: a stdout-ot ugyanoda irányítja, ahová a stderr irányítva lett

A csővezeték

- A **cső** avagy **csővezeték** (pipe, pipeline) $PROGRAM_1$ kimenetét (stdout-ot) $PROGRAM_2$ bemenetére (stdin-re) köti. A második program tehát az első által produkált eredményt tekinti bemenetként:



- Több programból álló csővezeték is létrehozható.
- A programok számára a csővezeték használata is teljesen átlátszó.
- Adatcsere köztes (ideiglenes) állomány használata nélkül
- A cső létrehozása az esetleges átirányítások elvégzése előtt történik.
- Megadása: a két parancsot a `|` (függőleges vonal) jellel elválasztva adjuk ki egy sorban
- `tee` *ÁLLOMÁNY*: Stdin tartalmát változatlan formában kiírja stdout-ra ill. a megadott állomány(ok)ba is (a csővezeték „megcsapolása”).

Felhasználói információk I.

- `who`:
  - az aktuálisan bejelentkezett felhasználók kilistázása
  - minden sorban egy adott felhasználóra vonatkozó információk jelennek meg 6 oszlopban (az oszlopokat szóközök tagolják):
    1. felhasználói azonosító
    2. virtuális terminál neve
    3-5. bejelentkezés dátuma és időpontja (hónap, nap, időpont)
    6. távoli számítógép neve vagy címe (el is maradhat)
- `who am i`:
  - csak az aktuális felhasználóra vonatkozó adatok jelennek meg
  - Bizonyos esetekben az 1. oszlopban a felhasználói azonosító elé egy felkiáltójellel (`!`) elválasztva kiíródik a számítógép neve is.
- `whoami`: az aktuális felhasználó azonosítójának kiírása
- `w`: a `who` parancsnál részletesebb információk kiírása

Felhasználói információk II.

- `groups`: kiírja, hogy mely csoport(ok)ba tartozik az aktuális felhasználó
- `finger`:
  - Ha nem adunk meg paramétert, akkor a jelenleg bejelentkezett felhasználókról jelenít meg egy listát. Ebben minden sorban egy adott felhasználóról jelennek meg különféle információk (pl. azonosító, név).
  - Ha egy felhasználó azonosítója szerepel paraméterként, akkor csak erről a felhasználóról jelenít meg információkat. A kimenet többsoros lesz, és olyan plusz adatokat is kiír, mint a felhasználó saját könyvtárának elérési útja (`~`, home directory), a használt shell neve, telefonszáma, valamint a felhasználó könyvtárában levő `~/.plan` és `~/.project` állományok tartalma (utóbbiból csak az első sor).
  - Ha egy másik számítógépről szeretnénk hasonló információkat szerezni, paraméterként egy `@`*`GÉP`* vagy *`FELHASZNÁLÓ`*`@`*`GÉP`* alakú nevet adjunk meg (ezt a szolgáltatást biztonsági okokból sokszor letiltják).
- `chfn`: a `finger` parancs által kiírt néhány információ megváltoztatása

Felhasználók nyilvántartása

- `/etc/passwd`:
  - felhasználói azonosítók és kritikus adatok nyilvántartása
  - leírás: `man 5 passwd`
  - minden sorban egy adott felhasználó adatai tárolódnak 7 oszlopban (az oszlopokat kettőspontok tagolják):
    1. felhasználói azonosító
    2. kódolt jelszó (sokszor egy másik állományban található)
    3-4. nem fontosak
    5. a felhasználó teljes neve
    6. a felhasználó saját könyvtárának elérési útja
    7. a használt shell neve (elérési útja)
- `/etc/group`:
  - felhasználói csoportok nyilvántartása
  - leírás: `man 5 group`

Felhasználók azonosítása, bejelentkezés

- `login`: bejelentkezés erre a számítógépre
- `rlogin`: bejelentkezés egy távoli számítógépre
- `passwd`:
  - a jelenlegi felhasználó jelszavának beállítása
  - Ha egy felhasználói azonosítót is megadunk paraméterként, akkor az ő jelszavát állíthatjuk be (erre csak a `root` képes).
- `gpasswd`: egy felhasználói csoport jelszavának beállítása/törlése, ill. felhasználók kinevezése csoport-adminisztrátorrá (előbbire a csoport-adminisztrátorok és a `root`, utóbbira csak a `root` képes)
- `newgrp`: az aktuális felhasználót egy másik csoportba lépteti be

Mindenféle segédprogram

- `write`, `talk`, `news`: csevegés, hírek olvasása
- `mail`, `sendmail`, `pine`, `pico`, `from`, `biff`, `xbiff`: elektronikus levelezés (e-mail)
- `ping`, `traceroute`, `telnet`, `ssh`, `ftp`, `sftp`: hálózati diagnosztika, terminálkapcsolat teremtése távoli számítógéppel, állományok átvitele
- `expr`, `bc`, `awk`/`gawk`, `factor`, `seq`: matematikai számítások
- `lpr`, `pr`: nyomtatás
- `grep`/`egrep`/`fgrep`, `awk`/`gawk`: információk keresése állományokban
- `locate`, `find`: állományok keresése név alapján
- `arch` (csak GNU/Linux), `uname`: információ az operációs rendszerről és a hardverplatformról
- `tar`, `zip`, `unzip`, `gzip`, `gunzip`, `bzip2`, `bunzip2`: archiválás, betömörítés, kicsomagolás

Egyéb hasznos parancsok

- `date`:
  - paraméter nélkül futtatva kiírja az aktuális dátumot és időt
  - megfelelően felparaméterezve beállítható vele a dátum és a pontos idő (erre csak a `root` képes)
- `sleep` *SZÁM*: a megadott számú másodpercig várakozik (a GNU/Linuxban törtszámot is megadhatunk)

Üzenetek megjelenítése, kiíratás

- `echo 'SZÖVEG'`:
  - Kiírja a megadott szöveget, majd sortörést végez (a következő sorba teszi a kurzort). Az aposztrófok megadása ajánlott.
  - `-e`: A `\` karakterrel kezdődő escape-szekvenciák is megengedettek a szövegben. Néhány példa: `\\` (közönséges `\`), `\n` (sortörés, így többsoros szöveget is kiírhatunk egy paranccsal), `\t` (tabulátor).
  - `-n`: a kurzor ugyanabban a sorban marad (nincs sortörés)
- `printf FORMÁTUM PARAMÉTEREK`: formázott kiíratás a C programozási nyelv azonos nevű függvényéhez hasonlóan
- `clear`: a képernyő ill. a terminálablak letörlése

Szűrők

- A **szűrő** (filter) egy olyan program, ami az – általában szöveges – bemeneten valamilyen átalakítást, szűrést hajt végre, és ennek eredményét írja ki a kimenetre. Gyakran csővezetékbe kötve alkalmazzuk őket.
- Korábban bemutatott szűrők: `cat`, `tee`, `pr`
- Később bemutatandó szűrők:
  - `grep 'SZÖVEG' ÁLLOMÁNY`:
    - Kiírja a megadott állomány mindazon sorait, amelyekben bárhol előfordul a megadott szöveg (az aposztrófok megadása ajánlott).
    - Ha nem adunk meg állományt, akkor a szabványos bemenetről olvas.
    - A program speciális minták keresésére is alkalmas (ld. később).
  - `awk`: mintakereső és -feldolgozó program saját programozási nyelvvel

A wc szűrő I.

- `wc` *`ÁLLOMÁNY(OK)`*:
  - statisztika készítése
  - Ha nem adunk meg állományt, akkor a szabványos bemenet tartalmáról ír ki egy egysoros statisztikát. Ez 3 oszlopból áll (az oszlopokat szóközök tagolják):
    1. sorok (sortörések) száma
    2. szavak száma
    3. bájtok száma
  - Egy állománynév esetén arról az állományról kapunk statisztikát, de ilyenkor egy negyedik oszlopban az állománynév is kiíródik!
  - Több állomány esetén az állománynevek ugyancsak megjelennek a negyedik oszlopban, továbbá a legvégén (egy plussz sorként) egy összesítést is kapunk! Állománynévként ilyenkor „`total`” („`összesen`”) jelenik meg.

A `wc` szűrő II.

- A megjeleníteni kívánt információkat a következő opciókkal szabályozhatjuk:
  - `-c`: csak a bájtok száma jelenik meg
  - `-l` („ell”): csak a sorok száma jelenik meg
  - `-w`: csak a szavak száma jelenik meg
  - Ezek az opciók nem zárják ki egymást. Ha egynél többet adunk meg közülük, akkor az adatok megjelenési sorrendje: sorok száma, szavak száma, bájtok száma.
  - Az előbbi dián említett plussz oszlop (állománynév) ill. sor (összesítés) a fenti opciók alkalmazása esetén is megjelenik!

A `sort` szűrő

- `sort` *ÁLLOMÁNY(OK)*:
  - A bemenetet soronként növekvő sorrendbe rendezi, majd az eredményt kiírja a kimenetre.
  - Ha nem adunk meg állományt, akkor a szabványos bemenetről olvas.
  - Több állomány esetén azok összesített (egymás után fűzött) tartalmát rendezi le.
  - alapértelmezés: ábécé sorrend (lexikografikus rendezés)
  - `-b`: a sorok elején álló szóközöket és tabulátorokat figyelmen kívül hagyja
  - `-f`: a kisbetűk és a nagybetűk egyenértékűek
  - `-n`: Numerikus rendezés: minden sor első szavát egy előjeles valós számnak tekinti (tizedesponttal vagy -vesszővel), és a sorokat a számok értéke szerint rakja sorrendbe. A pozitív előjelet (`+`) nem szabad kiírni!
  - `-r`: csökkenő sorrend
  - `-u`: a többször előforduló azonos sorok közül csak egyetlen példányt tart meg

A `head` és a `tail` szűrő

- `head` *ÁLLOMÁNY*:
  - a bemenet elejét írja ki
  - Alapesetben a bemenet első 10 sora jelenik meg.
  - Ha nem adunk meg állományt, akkor a szabványos bemenetről olvas.
  - `-c` *SZÁM*: az első *SZÁM* bájtot írja ki
  - `-n` *SZÁM*: az első *SZÁM* sort írja ki
- `tail` *ÁLLOMÁNY*:
  - a bemenet végét írja ki
  - Alapesetben a bemenet utolsó 10 sora jelenik meg.
  - Ha nem adunk meg állományt, akkor a szabványos bemenetről olvas.
  - `-c` *SZÁM*: az utolsó *SZÁM* bájtot írja ki
  - `-n` *SZÁM*: az utolsó *SZÁM* sort írja ki
  - Ha a *SZÁM* egy `+` jellel kezdődik, akkor a bemenet elejétől számított *SZÁM*-adik bájttól vagy sortól kezdődő tartalmat írja ki.

Egyéb hasznos szűrők

- `tac`: a bemenet sorait fordított sorrendben írja ki
- `cmp`: két tetszőleges állomány tartalmának összehasonlítása
- `diff`: két szöveges állomány tartalmának összehasonlítása, az összes eltérés kiírásával
- `cut`: a bemenet minden sora adott részének kiírása (kivágása)
- `tr`: a bemenetben előforduló bizonyos karakterek törlése vagy lecserélése másik karakterre
- `col -b`: A `man` parancs kimenetére érdemes használni, ha a szöveget állományban szeretnénk eltárolni. Az így szűrt szövegek más operációs rendszerben is helyesen fognak megjelenni.
- `uniq`: a bemenetben egymás után többször szereplő azonos sorokat kiszűri
- `dos2unix`: A DOS formátumú szöveget UNIX formájúvá alakítja. A sortörést a UNIX-ban az ASCII 10-es kódú karakter jelenti, míg DOS-ban a 13-as és 10-es kódú karakterek alkotta párost használják erre.
- `unix2dos`: a UNIX formátumú szöveget DOS formájúvá alakítja

A shell

- A burok avagy héj (**shell**) egy olyan rendszerprogram, amely a kernel és a felhasználó között közvetít. Ily módon egyrészt hozzáférést biztosít a kernel egyes funkcióihoz, másrészt különféle kényelmi szolgáltatásokat is nyújt. Többek közt lehetővé teszi programok indítását, így sokszor **parancsértelmezőnek** (command interpreter) is hívják.
- Sokféle shell létezik:
  - `/bin/sh`: Bourne SHell (ez a legősibb shell)
  - `/bin/csh`: C SHell
  - `/bin/ksh`: Korn SHell
  - **`/bin/bash`**: Bourne Again SHell (ezzel fogunk foglalkozni)
- `chsh` *`SHELL`*: használni kívánt shell beállítása az aktuális felhasználónak
- Kilépés a shellből:
  - `exit`
  - CTRL+D (mint a bemenet végének jelzése)

A shell feladatai

- Parancssor kezelése
- Munkafolyamatok (job) kezelése (ld. később)
- Átirányítások elvégzése
- Csővezeték létrehozása
- Helyettesítő nevek (alias) értelmezése (ezzel nem foglalkozunk)
- Mintaillesztő karakterek értelmezése
- Állománynév-kiegészítés végrehajtása
- Parancsok kötegelt végrehajtása (shell scriptek)
- Vezérlési szerkezetek értelmezése
- Személyes (felhasználótól függő) beállítások kezelése:
  - környezet (environment), környezeti változók
  - `~/.profile`: Ha létezik ez az állomány a felhasználó saját könyvtárában, akkor ennek tartalma minden bejelentkezéskor (login) végrehajtódik, mint egy script.

A futtatókörnyezet és az alshellek

- A shellben végrehajtott parancsok működését befolyásoló belső jellemzők összessége alkotja a **futtatókörnyezetet** (execution environment). Ez pl. a következőket tartalmazza:
  - a szabványos bemenetre és a két kimenetre alkalmazott átirányítások, ill. a csővezeték használata
  - az aktuális könyvtár (`.`) elérési útja
  - a környezet, környezeti változók (ld. később)
- A shellből indított újabb shellt, ill. szűkebb értelemben annak futtatókörnyezetét **alshellnek** (subshell) nevezzük. A shellből végrehajtott programok általában egy-egy külön alshellben futnak.
- Az alshell mindig a szülő futtatókörnyezetét örökli, de sosem módosíthatja azt (csak a sajátját). Az öröklés alól van azonban néhány kivétel:
  - A futtatáskor alkalmazott átirányítások, ill. a csővezeték használata módosíthatják és kiegészíthetik az örökölteket.
  - A környezeti változók öröklődése speciálisan zajlik (ld. később).

A `bash` jellemzői

- Leírás: `man bash`, `info bash`
- Két üzemmódja van:
  - interaktív mód:
    - A shell egy **parancssort** (command line) jelenít meg, majd a szabványos bemenetről parancsok végrehajtására várakozik. Amint a parancs végrehajtása befejeződik, a parancssort újra visszakapjuk.
    - A parancssor elején látható, dinamikusan változó felirat a **prompt**. Sokszor dollárjelre végződik, és általában a felhasználó azonosítóját, a számítógép nevét és az aktuális könyvtárat mutatja.
    - A korábban végrehajtott parancsokat az **előzmények listája** (command history) tartalmazza. Előhívása: a FEL és LE gombokkal.
  - script avagy neminteraktív mód:
    - Sem parancssor, sem prompt nem jelenik meg.
    - A shell a végrehajtandó parancsokat egy szöveges állományból olvassa (kötegelt végrehajtás). Az állomány végének elérésekor a shell befejeződik.

A `bash` speciális karakterei

- Sok írásjelnek és szimbólumnak a shell számára különleges jelentése van (néha megkettőzve is):
  - korábban megismert **speciális karakterek**: `|`, `<`, `>`, `>>`, `~`, `?`, `*`
  - további speciális karakterek: `<<`, `(`, `)`, `((`, `))`, `[`, `]`, `{`, `}`, `||`, `&`, `&&`, `$`, `#`, `\`, `` ` ``, `'`, `"`, `!`, `;`, `;;`, szóköz, tabulátor, sortörés
  - csak parancsnévként speciális karakterek (a paramétereken belül ezek továbbra is közönségesek): `.`, `:`, `=`, `^`, `[[`, `]]`
  - Az előbbi kategóriát kivéve a többi speciális karakter a sor bármelyik részén speciálisan viselkedik.
- A speciális karakterek értelmezése a shell feladata, ebből a többi program semmit sem vesz észre. Az ilyen karaktereket tehát a programok nem látják (pl. ha azok valamelyik paraméterben fordultak elő), csak a hatásukat (eredményüket) kapják meg.
- Sem az útvonalak megadásához használt `/`, `.` és `..` jelölések, sem az opciókat bevezető mínuszjel (`-`) *nem speciális* a fenti értelemben!

Speciális jelentés elnyomása (quoting)

- `\`*KARAKTER*:
  - a `\` (backslash) után írt karaktert közönségesként értelmezi
  - A sor végére írt `\` lehetővé teszi többsoros parancs végrehajtását (ld. következő dia).
- `'`*SZÖVEG*`'`: Az aposztrófpár közé zárt *bármely* karaktert közönségesként értelmezi (még a `\`-t is). A szöveg nyilván nem tartalmazhat aposztrófot.
- `"`*SZÖVEG*`"`:
  - Az idézőjelpár közé zárt szövegben csak `\`, `$` és `` ` `` tartja meg speciális jelentését.
  - A szövegben a `\` csak `$`, `` ` ``, `"` és `\` előtt számít speciálisnak. Ebből következik, hogy a szövegbe egy közönséges idézőjelet szúrhatunk be a `\"` karakterpárossal.
  - A sor végére írt `\` itt is használható többsoros parancs kiadására.
  - Az idézőjelpár közé zárt szövegben a shell nem végzi el a szavakra bontást (ld. következő dia), így tehát mindig *egyetlen szót* kapunk. Lásd még: parancs-, változó-, paraméter- és aritmetikai-behelyettesítés.

Parancsok végrehajtása

- Szóköz, tabulátor:
  - A parancs végrehajtása előtt a shell elvégzi a sor **szavakra bontását** (word splitting), azaz a sort parancsnévre, opciókra és paraméterekre bontja. **Szóhatárolónak** egymás után álló egy vagy több szóköz ill. tabulátor számít. Néhány esetben a sortörés is szóhatároló lesz.
  - Az eredményben a szavakat pontosan egy szóköz fogja tagolni!
- Sortörés (ENTER, újsor):
  - parancs végrehajtása
  - Ha a parancs neve nem tartalmaz könyvtárnevet ill. / jelet, akkor a PATH változóban (ld. később) felsorolt könyvtárakban keresi a programot a shell.
- \: Ha a sort a \ karakterrel zárjuk (közvetlenül a sortörés előtt), akkor a parancsot a következő sorban folytathatjuk (többsoros parancs). Ekkor az ún. másodlagos prompt jelenik meg a parancssor elején.
- #: A sor végéig tartó **megjegyzés** (comment) kezdetét jelzi. A # jelet az előtte levő szótól egy szóhatárolóval kell elválasztani!

Kilépési státusz

- A programok befejeződésükkor egy különleges értékkel, az ún. **kilépési státusszal** (exit status) jelzik lefutásuk sikeres vagy sikertelen voltát. Ez egy előjeles egész szám, ahol 0 jelzi a sikeres (hibamentes) lefutást. Tekinthetjük tehát egyfajta hibakódnak is.
- Fontos, hogy a kilépési státusz *nem része* a szabványos kimenetre kerülő kimenetnek! Kezelését a kernel végzi (nem pedig a shell), de értékét a shellben is felhasználhatjuk.
- `exit` *SZÁM*:
  - kilépés a shellből (alshellből), a kilépési státusz *SZÁM* lesz
  - A szám elhagyása esetén a legutóbb végrehajtott parancs kilépési státuszát használja.
- `!` *PARANCS*: A parancs kilépési státuszának logikai tagadása (nemzérusból 0, nullából 1 lesz). A felkiáltójelet az előtte és utána levő szavaktól egy-egy szóhatárolóval kell elválasztani!
- A legutóbb végrehajtott parancs kilépési státuszát a `$?` speciális paraméter tárolja (ld. később).

Összetett parancsok I.

- *PARANCS*$_1$ `|` *PARANCS*$_2$ `|` …: A megadott parancsok végrehajtása csővezetékként (csak emlékeztetőül). Mindegyik parancs alshellben lesz végrehajtva, a kilépési státusz az utolsó parancs státusza lesz.
- **Parancslisták**:
  - *PARANCS* `&`: A megadott parancs végrehajtása egy alshellben a háttérben (ld. később). A shell nem várja meg a parancs befejeződését, a kilépési státusz pedig 0 lesz. A bemenet alapesetben a `/dev/null` lesz.
  - *PARANCS*$_1$ `;` *PARANCS*$_2$ `;` …: A megadott parancsok végrehajtása egymás után a megadott sorrendben (mintha a pontosvesszők helyén sortörés állna). A kilépési státusz az utolsó parancs státusza lesz.
  - *PARANCS*$_1$ `&&` *PARANCS*$_2$ `&&` …: *PARANCS*$_2$ végrehajtása akkor és csak akkor, ha *PARANCS*$_1$ kilépési státusza 0 (ÉS-lista).
  - *PARANCS*$_1$ `||` *PARANCS*$_2$ `||` …: *PARANCS*$_2$ végrehajtása akkor és csak akkor, ha *PARANCS*$_1$ kilépési státusza nemzérus (VAGY-lista).
  - Mindegyik *PARANCS* egy csővezeték is lehet.

Összetett parancsok II.

- Parancsok csoportosítása:
  - `{ PARANCSLISTA; }`: A parancslista végrehajtása az aktuális (!) shellben. A zárójeleket az előttük ill. mögöttük álló szavaktól egy-egy szóközzel (szóhatárolóval) kell elválasztani! A listát lezáró pontosvessző kiírása ugyancsak kötelező, de helyette sortörés is alkalmazható!
  - `( PARANCSLISTA )`: a parancslista végrehajtása egy alshellben
  - A zárójelek mindkét esetben a parancsoktól különválasztva, másik sorba is kerülhetnek.
  - A csoportosítás egyik előnye, hogy az ilyen összetett parancsra is alkalmazhatók az átirányítások és a csővezeték.
  - A csoport kilépési státusza megegyezik a lista státuszával.

Parancs-behelyettesítés

- `` `PARANCS` ``:
  - A shell a kifejezést a fordított aposztrófok (backquote) közé írt parancs kimenetével (a szabványos kimenetre írt eredménnyel) helyettesíti. Ezt hívjuk **parancs-behelyettesítésnek** (command substitution). A parancs, amely akár összetett parancs is lehet, egy alshellben lesz végrehajtva.
  - A fordított aposztrófpár közé zárt szövegben csak `\` tartja meg speciális jelentését, de csak `$`, `` ` `` és `\` előtt számít speciálisnak. Ebből következik, hogy a szövegbe egy újabb parancs-behelyettesítést ágyazhatunk be a `` \` `` karakterpárossal.
  - A parancs végrehajtása után annak kimenetét szavakra bontja a shell, továbbá a mintaillesztő karaktereket (`*`, `?`) is ekkor fejti ki. Ráadásul ebben az esetben a kimenetben előforduló sortörés is szóhatárolónak számít. Ha ezeket el akarjuk kerülni, az egész kifejezést zárjuk idézőjelpár közé!
- `$(PARANCS)`: Alternatív jelölés, de itt még a `\` sem speciális.

A környezet I.

- A **környezet** (environment) nem más, mint név-érték párok halmaza. Az elemek alakja *NÉV*=*ÉRTÉK*, ahol *NÉV* egy **környezeti változót** avagy **shell változót** (environment variable) azonosít.
- A változók neve betűket, számokat és aláhúzásjelet (_) tartalmazhat, és nem kezdődhet számjeggyel. A shell a változók értékét mindig szövegesen kezeli.
- *NÉV*=*ÉRTÉK*:
  - Értékadás egy létező változónak, vagy új változó létrehozása. Az egyenlőségjel elé tilos szóközt vagy tabulátort írni!
  - Ha az értéket elhagyjuk, akkor a változó értéke az **üres szó** (`""`) lesz.
  - Az *ÉRTÉK* csak *egyetlen szó* lehet! Lásd még: idézőjel használata.
  - Az *ÉRTÉK* tartalmazhat parancs-, változó-, paraméter- és aritmetikai-behelyettesítést (ld. lent ill. később) is. Szavakra bontás itt nem történik.
- $*NÉV*:
  - a megadott nevű **változó** aktuális értékének **behelyettesítése**
  - A behelyettesített szövegben itt is megtörténik a szavakra bontás és a mintaillesztő karakterek kifejtése. Bővebben: előző dia.

A környezet II.

- `unset` *NÉV*: változó megszüntetése
- `set`, `printenv`: a változók név szerint rendezett listájának kiírása
- `read` *NÉV*:
  - A billentyűzetről (szabványos bementről) beolvasott szöveg az ENTER leütése után a megadott változó új értéke lesz.
  - Ha nem adunk meg nevet, akkor a beolvasott szöveg sehova se lesz eltárolva. Hasznos, ha pl. ENTER leütésére akarunk várakozni.
- `export` *NÉV*, `export` *NÉV*`=`*ÉRTÉK*: az újonnan létrehozott változót az alshell is látni fogja (örökli)
- Egy alshell, ill. egy külön futtatókörnyezetben végrehajtott parancs a környezetből csak azokat a változókat örökli, amelyeket
  1. az aktuális shell is örökölt, továbbá
  2. azokat az új változókat, amiket az `export` paranccsal megjelöltünk.
- Az előbbi szabály *nem vonatkozik* a csoportok (`(` *PARANCSLISTA* `)`) és a parancs-behelyettesítés alshelljére! Ezek futtatókörnyezete az `export` használata nélkül is örököl minden változót.

Fontosabb környezeti változók

- Néhány fontosabb környezeti változó:
  - `HOME`: az aktuális felhasználó saját könyvtárának (`~`) elérési útja
  - `LANG`: a használni kívánt nyelv és karakterkódolás
  - `MAIL`: az aktuális felhasználó elektronikus levelesládájának elérési útja
  - `PATH`: A programok végrehajtásához használt keresési útvonalak listája. Az útvonalakat kettőspontok választják el egymástól. Alapesetben nem tartalmazza az aktuális könyvtárt jelző `.` könyvtárt.
  - `PS1`, `PS2`: az elsődleges és másodlagos promptok szövege
  - `SHELL`: a használt shell elérési útja
  - `TERM`: A (virtuális) terminál típusa. Ha valamelyik program erre hivatkozva nem futna, akkor állítsuk pl. „`vt100`”-ra.
  - `USER`: az aktuális felhasználó azonosítója

Shell scriptek

- Az interaktív mód mellett a shell képes arra is, hogy a felhasználó beavatkozása nélkül, automatikusan hajtson végre parancsokat egy *szöveges* állományból. Ezt a módszert **kötegelt végrehajtásnak** vagy **feldolgozásnak** (batch processing) hívjuk, a hozzá szükséges állományokat pedig **shell scripteknek** (röviden script) nevezzük.
- Scriptek tartalma:
  - `#! /bin/bash`: A script első sora tartalmazhatja ezt a speciális megjegyzést, az ún. **parancsértelmező fejlécet**. A `#` jelnek *közvetlenül* a sor elején kell elhelyezkednie! A `#!` páros angol neve: sha-bang.
  - bármilyen parancs, program végrehajtása: Az állomány minden sora külön-külön parancsként lesz végrehajtva (persze többsoros parancsok is lehetnek). Az üres sor szintén megengedett.
  - vezérlési szerkezetek: Ezeket a parancssorban is lehet használni, de igazán itt van értelmük és hasznuk. Alkalmazásuk a shellt programozási nyelvhez hasonlóvá teszi.

Scriptek futtatása

- Script futtatása egy alshellben:
  1. `bash` *SCRIPT PARAMÉTEREK*
  2. Script futtatása végrehajtható állományként (ez javasolt):
     1. A script első sorában a parancsértelmező fejlécet kell használni.
     2. `chmod +x` *SCRIPT*: futtatási jog engedélyezése
     3. `./`*SCRIPT PARAMÉTEREK*: Az elérési útban az aktuális könyvtárt is mindig ki kell írni, mert ebben alapesetben – biztonsági okokból – a shell nem keres futtatható állományokat (ld. PATH környezeti változó).
- Script futtatása az aktuális (!) shellben:
  3. `. ./`*SCRIPT PARAMÉTEREK*: Az első `.` a shell egy beépített parancsaként szerepel.
  4. `source ./`*SCRIPT PARAMÉTEREK*
- Ha a scriptet futtathatóvá tesszük, és a 2. módszert követjük, akkor ezzel létrehoztunk egy paraméterezhető saját parancsot!

Pozícionális paraméterek

- **Pozícionális** (sorszám alapján azonosított) **paraméterek**:
  - `$1`, `$2`, …, `$9`: a script indításakor annak neve után megadott 1., 2., …, 9. **paraméter** értékének **behelyettesítése** (parameter expansion)
  - `${`*`SORSZÁM`*`}`: Hatása megegyezik az előzőkkel, de itt többjegyű sorszámot is megadhatunk.
- A behelyettesített szövegben itt is megtörténik a szavakra bontás és a mintaillesztő karakterek kifejtése. Ez a szabály a `$` minden később bemutatandó használatára is igaz. Bővebben: „Parancs-behelyettesítés”.
- `shift` *`SZÁM`*:
  - Ha nem adunk meg paramétert, vagy az 1-et használjuk, akkor mindegyik pozícionális paraméter eggyel kisebb sorszámúvá lesz átnevezve (léptetve), a korábban legelső paraméter értéke pedig elveszik. (Azaz a korábban `$2` értéket most `$1` alatt fogjuk elérni.) Felfogható úgy is, hogy kitörli a script első paraméterét. Ezenfelül eggyel csökkenti a `$#` speciális paraméter értékét (ld. következő dia).
  - Egynél nagyobb szám megadása esetén többször léptet.

Speciális paraméterek

- **Speciális paraméterek**:
  - `$*`: az összes megadott pozícionális paraméter szóközzel tagolt listája az eredeti sorrendben
  - `$#`: a megadott pozícionális paraméterek száma
  - `$0` (dollárjel és nulla): A script neve és elérési útja. Ha nem scripten belül használjuk, akkor a shell nevét és elérési útját tartalmazza.
  - `$?`: a legutóbb végrehajtott parancs kilépési státusza
  - `$$`: az aktuális shell vagy script processz-azonosítója (ld. később)

Változók és paraméterek értékének behelyettesítése

- `$`*`NÉV`*: A megadott nevű környezeti változó aktuális értékének behelyettesítése (csak emlékeztetőül). Ha a változó nem létezik, üres szót kapunk (azaz a kifejezés egyszerűen törölve lesz).
- `${`*`NÉV`*`}`: Hatása megegyezik az előzővel, de ez akkor is használható, ha közvetlenül a kifejezés után betű, számjegy vagy aláhúzásjel áll (máskülönben azt a név részének tekintené a shell).
- `${!`*`NÉV`*`}`: A megadott nevű változó értékét egy változónévnek ($NÉV_2$) tekinti, és a kifejezést $NÉV_2$ értékével helyettesíti (indirekció).
- *`PAR`*: környezeti változó neve, poz. paraméter sorszáma, spec. paraméter jele
- `${`*`PAR`*`:-`*`ÉRTÉK`*`}`: Ha `$`*`PAR`* üres, a kifejezést *`ÉRTÉK`*-kel helyettesíti. Különben a kifejezés értéke `$`*`PAR`*. (Alapértelmezett érték használata.)
- `${`*`PAR`*`:+`*`ÉRTÉK`*`}`: Ha `$`*`PAR`* üres, a kifejezést az üres szóval helyettesíti. Különben a kifejezés értéke *`ÉRTÉK`*. (Alternatív érték használata.)
- `${`*`NÉV`*`:=`*`ÉRTÉK`*`}`: Ha `$`*`NÉV`* üres, a változó értékét *`ÉRTÉK`*-re állítja. Az előző feltételtől függetlenül, de már az esetleges értékadás elvégzése után, a kifejezés értéke `$`*`NÉV`* lesz. (Alapértelmezett érték beállítása.)

Feltételes kifejezések I.

- Néhány vezérlési szerkezet (ld. később) alkalmazásánál szükség lehet különféle **feltételek** megadására. Ez általában egy olyan parancs(lista) végrehajtását jelenti, amely kilépési státuszában jelzi a feltétel teljesülését vagy nemteljesülését. Megegyezés szerint 0 jelzi az igazat (teljesülést), nemzérus pedig a hamisat.
- `grep 'SZÖVEG' ÁLLOMÁNY`: A korábban már említett `grep` szűrő is kilépési státuszában jelzi, talált-e olyan sort a bemenetben, amely a megadott szöveget tartalmazza.
- `test KIF`:
  - Kiértékeli a megadott *KIF* **feltételes kifejezést** (conditional expression), majd a kilépési státuszban jelzi annak logikai igazságértékét. Látható kimenetet nem produkál.
  - A kifejezésben előforduló operátorokat ill. operandusokat szóközzel (szóhatárolóval) kell egymástól elválasztani!
  - A kifejezés tartalmazhat parancs-, változó-, paraméter- és aritmetikai-behelyettesítést (ld. később) is.

Feltételes kifejezések II.

- Összetett kifejezések:
  - `(` *KIF* `)`: csoportosítás (műveleti sorrend felülbírálása)
  - `!` *KIF*: Logikai igazságérték tagadása (negáció). A legerősebb művelet.
  - KIF_1 `-a` KIF_2: Logikai ÉS (konjunkció). Gyengébb a negációnál.
  - KIF_1 `-o` KIF_2: Logikai MEGENGEDŐ VAGY (diszjunkció). Gyengébb a konjunkciónál.
- Állományjellemzők vizsgálata:
  - `-a` *NÉV*, `-e` *NÉV*: igazak, ha *NÉV* egy létező, tetszőleges típusú állomány neve
  - `-d` *NÉV*: igaz, ha *NÉV* egy létező könyvtár neve
  - `-f` *NÉV*: igaz, ha *NÉV* egy létező közönséges állomány neve
  - `-h` *NÉV*: igaz, ha *NÉV* egy létező szimbolikus lánc neve
  - `-r` *NÉV*: igaz, ha *NÉV* egy létező olvasható állomány neve
  - `-w` *NÉV*: igaz, ha *NÉV* egy létező írható állomány neve
  - `-x` *NÉV*: igaz, ha *NÉV* egy létező végrehajtható áll. neve

Feltételes kifejezések III.

- Szöveges összehasonlítás (mindegyik *SZÖVEG* csak *egyetlen szó* lehet!):
  - `-z` *SZÖVEG*: igaz, ha *SZÖVEG* az üres szó
  - `-n` *SZÖVEG*: igaz, ha *SZÖVEG* nem az üres szó
  - $SZ\ddot{O}VEG_1$ `==` $SZ\ddot{O}VEG_2$: igaz, ha a két szöveg megegyezik
  - $SZ\ddot{O}VEG_1$ `!=` $SZ\ddot{O}VEG_2$: igaz, ha a két szöveg eltérő
- Numerikus összehasonlítás (csak előjeles egész számokkal):
  - $SZÁM_1$ `-eq` $SZÁM_2$: igaz, ha $SZÁM_1 = SZÁM_2$
  - $SZÁM_1$ `-ne` $SZÁM_2$: igaz, ha $SZÁM_1 \neq SZÁM_2$
  - $SZÁM_1$ `-lt` $SZÁM_2$: igaz, ha $SZÁM_1 < SZÁM_2$
  - $SZÁM_1$ `-le` $SZÁM_2$: igaz, ha $SZÁM_1 \leq SZÁM_2$
  - $SZÁM_1$ `-gt` $SZÁM_2$: igaz, ha $SZÁM_1 > SZÁM_2$
  - $SZÁM_1$ `-ge` $SZÁM_2$: igaz, ha $SZÁM_1 \geq SZÁM_2$

- `[` *KIF* `]`: Alternatíva a `test` parancs helyett. A szögletes zárójeleket az előttük ill. utánuk álló szavaktól egy-egy szóhatárolóval kell elválasztani!

Aritmetikai kifejezések I.

- `expr` *`KIF`*:
  - Kiértékeli a megadott *`KIF`* **aritmetikai kifejezést** (arithmetic expression), majd az eredményt a szabványos kimenetre írja. Számolni csak előjeles egész számokkal tud, így az eredmény is egész lesz.
  - A kifejezésben előforduló operátorokat ill. operandusokat szóközzel (szóhatárolóval) kell egymástól elválasztani!
  - A kifejezés tartalmazhat parancs-, változó-, paraméter- és aritmetikai-behelyettesítést (ld. következő dia) is.
  - `(` *`KIF`* `)`: csoportosítás (műveleti sorrend felülbírálása)
  - *`SZÁM`*$_1$ *`MŰVELET`* *`SZÁM`*$_2$: A megadott művelet elvégzése a két szám között. Használható műveletek: `+`, `-`, `*`, `/`, `%` (osztási maradék).
  - *`SZÁM`*$_1$ *`RELÁCIÓ`* *`SZÁM`*$_2$: A két szám összehasonlítása. Az eredmény igaz esetén 1, különben 0 lesz. Használható relációk: `<`, `<=`, `>`, `>=`, `==`, `!=`.
  - Vigyázzunk, mert a fenti karakterek közül sokat a shell speciálisan kezel!

Aritmetikai kifejezések II.

- `$((KIF))`:
  - **aritmetikai kifejezés** értékének **behelyettesítése** (arithmetic evaluation/ expansion)
  - Nem ekvivalens az `expr` paranccsal! Mivel bizonyos szempontból többet tud az említett parancsnál, így ezt érdemesebb használni.
  - A kifejezésen belül a `$` és `` ` `` megtartja speciális jelentését, minden más karakter közönségesnek számít (a `\` is). Így a kifejezés tartalmazhat parancs-, változó-, paraméter- és újabb aritmetikai-behelyettesítést is.
  - A kifejezés használhatja a C programozási nyelv operátorait, valamint a környezeti változókat (akár `$` nélkül is). Szóközökre nincs szükség.
  - Az operandusok csak előjeles egész számok lehetnek. Az eredmény szintén egy előjeles egész lesz.
  - Műveletek: `++`, `--`, `+`, `-`, `*`, `/`, `%`, `**` (hatványozás), `<<`, `>>`, `<`, `<=`, `>`, `>=`, `==`, `!=`, `!`, `~`, `&`, `^`, `|`, `&&`, `||`, `?:`, `=` és társai, `,` (vessző), `(`, `)`. A relációk eredménye igaz esetén 1, különben 0 lesz.

Vezérlési szerkezetek I.

- **Vezérlési szerkezeteket** (control structure) gyakran használunk a programozás során az utasítások végrehajtási sorrendjének módosítására.
- A parancsokat a shell alapesetben szekvenciálisan (természetes sorrendben) hajtja végre.
- A következő konstrukcióknál – pár kivétellel – a pontosvesszőkre (`;`) csak akkor van szükség, ha az elválasztott utasításrészeket ugyanabba a sorba írnánk.
- `for` *NÉV* `in` *LISTA*`;`
  `do` *PARANCSLISTA*`;`
  `done`:
  - A *NÉV* nevű környezeti változó (mint ciklusváltozó) sorban felveszi *LISTA* elemeinek értékét, miközben minden alkalommal végrehajtódik a *PARANCSLISTA* (diszkrét ismétléses vezérlés).
  - A *LISTA* tetszőleges szöveges értékek listája, ahol az elemeket szóközök (szóhatárolók) tagolják. Tartalmazhat parancs-, változó-, paraméter- és aritmetikai-behelyettesítést, ill. mintaillesztő karaktert is.

Vezérlési szerkezetek II.

- `for ((`KIF_1`;`KIF_2`;`KIF_3`));`
  `do` *PARANCSLISTA*`;`
  `done`:
  - Teljesen úgy működik, mint a C programozási nyelv hasonló vezérlési szerkezete (számlálásos ismétléses vezérlés). Először KIF_1 (kezdeti értékadás) lesz kiértékelve, majd következnek az iterációk. Ha KIF_2 (kilépési feltétel) értéke nemzérus (igaz), a *PARANCSLISTA* egyszer végrehajtódik, majd KIF_3 (ciklusváltozó módosítása) is ki lesz értékelve. Utána ismét KIF_2 értékétől függően vagy új iteráció kezdődik (ha nemzérus), vagy befejeződik a végrehajtás (ha 0).
  - A három *KIF* egy-egy aritmetikai kifejezés (ld. `$((…))`) lehet, nem pedig feltételes kifejezés! Ennek megfelelően a C nyelv műveleteit használhatjuk, környezeti változók is előfordulhatnak (akár `$` nélkül is), szóközökre pedig nincs szükség. A kifejezéseket tagoló pontosvesszők (`;`) kiírása kötelező! A kifejezéseket nem fontos a `$((` és `))` jelekkel közrezárni.
  - Bármelyik kifejezés elhagyása esetén annak értéke 1-nek számít.

Vezérlési szerkezetek III.

- `while` *`FELTÉTEL`*`;`
  `do` *`PARANCSLISTA`*`;`
  `done`:
  - A *`PARANCSLISTA`* ismételt végrehajtása addig, amíg a *`FELTÉTEL`* igaz (előfeltételes ismétléses vezérlés). A végrehajtás akkor fejeződik be, ha a *`FELTÉTEL`* hamis.
  - A *`FELTÉTEL`* tetszőleges parancs, amely kilépési státuszában jelzi egy feltétel igazságértékét (pl. `test`). Bővebben: „Feltételes kifejezések”.
- `until` *`FELTÉTEL`*`;`
  `do` *`PARANCSLISTA`*`;`
  `done`: A `while` ellentéte, azaz a *`PARANCSLISTA`* végrehajtása akkor fejeződik be, ha a *`FELTÉTEL`* igaz. Vigyázat, ez is előfeltételes vezérlés!
- `break`, `continue`: Kilépés a ciklusból, ill. rátérés a ciklus következő iterációjára (`for`, `while` és `until` esetén használhatók). Mindig az őket körbevevő *legbelső ciklusra* vonatkoznak!

Vezérlési szerkezetek IV.

- `if` *FELTÉTEL*$_1$`;`
  `then` *PARANCSLISTA*$_1$`;`
  `elif` *FELTÉTEL*$_2$`;`
  `then` *PARANCSLISTA*$_2$`;`
  …
  `else` *PARANCSLISTA*$_0$`;`
  `fi`:
  - Egyszeres vagy többszörös szelekciós vezérlést valósít meg. Ha *FELTÉTEL*$_1$ igaz, akkor *PARANCSLISTA*$_1$ végrehajtódik. Különben, ha *FELTÉTEL*$_2$ igaz, akkor *PARANCSLISTA*$_2$ hajtódik végre. És így tovább a többi feltétel esetén is. Végül, ha mindegyik feltétel hamis volt, akkor *PARANCSLISTA*$_0$ lesz végrehajtva.
  - Természetesen sem az `else` ág, sem az `elif` ágak megadása nem kötelező.
  - Feltételek: mint a `while` és `until` ciklusok esetén.

Vezérlési szerkezetek V.

- `case` *SZÓ* `in`
  MINTA$_1$`)` *PARANCSLISTA*$_1$`;;`
  MINTA$_2$`)` *PARANCSLISTA*$_2$`;;`
  …
  `*)` *PARANCSLISTA*$_0$`;;`
  `esac`:
  - Esetszétválasztásos szelekciós vezérlést valósít meg. Ha *MINTA*$_1$ illeszkedik *SZÓ*-ra, akkor *PARANCSLISTA*$_1$ végrehajtódik. Különben, ha *MINTA*$_2$ illeszkedik *SZÓ*-ra, akkor *PARANCSLISTA*$_2$ hajtódik végre. És így tovább a többi minta esetén is. Végül, ha egyik minta sem illeszkedett, akkor *PARANCSLISTA*$_0$ lesz végrehajtva (ezt az esetet nem fontos megadni). A dupla pontosvesszők (`;;`) kiírása kötelező!
  - A *SZÓ* tetszőleges szöveges érték, de csak *egyetlen szó* lehet. Tartalmazhat parancs-, változó-, paraméter- és aritmetikai-behelyettesítést is.

Vezérlési szerkezetek VI.

  - A minták az állománynevek megadásához használt mintaillesztő karaktereket is tartalmazhatják. Több minta összekapcsolható a `|` jellel, ami itt logikai MEGENGEDŐ VAGY-ot jelez, nem pedig csővezetéket.
  - Fontos, hogy az illeszkedés vizsgálata az egyes esetek sorrendjében történik, továbbá csak a *legelső* illeszkedő eset parancslistája lesz végrehajtva! Ebből következik, hogy a C programozási nyelv `switch` szerkezetétől eltérően a parancslistákban *nem kell alkalmazni* a `break` parancsot (főleg, hogy az csak cikluson belül lenne használható)!
- `exit` *`SZÁM`*:
  - A korábban már többször bemutatott `exit` parancs alkalmazható a scriptből való kilépésre is.
  - A *`SZÁM`* itt is a kilépési státusz megkívánt értéke lehet.
  - igazából nem vezérlési szerkezet, hanem vezérlőparancs

Reguláris kifejezések

- Sok program (főleg szűrők) használ mintaillesztést (pattern matching), mintakeresést (pattern scanning) és mintafeldolgozást (pattern processing). Ilyen esetekben a – legtöbbször szöveges – bemeneti adatok azon részével fog dolgozni a program, amely egy megadott mintának megfelel, azaz a **mintára illeszkedik** (vagy amire a minta illeszkedik). Az ilyen komplex minták egyik gyakran alkalmazott formája a **szabályos** avagy **reguláris kifejezés** (regular expression, regexp, RE).
- A reguláris kifejezésekkel mélyebben a formális nyelvek elmélete (theory of formal languages) foglalkozik.
- Vigyázat! Bár a filozófiájuk hasonló, de a reguláris kifejezéseket *nem szabad összekeverni* az állományneveknél használható mintákkal és mintaillesztő karakterekkel! Ott egy létező állomány/könyvtár nevét adjuk meg, itt viszont egy szöveg valamely részét választjuk ki. Ráadásul ugyanazon karaktereknek itt más a jelentése.

A reguláris kifejezések tulajdonságai

- Leírás: `man 7 regex`, `man grep`, `info grep`, `man awk`/`gawk`, `info gawk`
- Egy reguláris kifejezés a szövegnek mindig a *legkorábban* elkezdődő, és ezen belül a *leghosszabb* részére illeszkedik. Ez a részkifejezésekre is igaz. Az illeszkedő rész a szövegen belül *bárhol* – akár egy szó belsejében is – előfordulhat, kivéve néhány esetet (pl. `^` és `$`, ld. következő dia).
- Alapesetben a kisbetűk és nagybetűk különbözőnek számítanak illesztéskor.
- A reguláris kifejezésekben néhány karakternek speciális jelentése van. Mivel ezek közül sokat a shell is speciálisan kezel, így a parancssorban megadott reguláris kifejezést érdemes aposztrófok közé zárni.
- A reguláris kifejezések nem „mindenhatóak”, nem lehet velük minden feltételt leírni! Így pl. belátható, hogy nem létezik olyan reguláris kifejezés, amely csak olyan szövegre illeszkedik, amely pontosan *N* db „a” betűt és ugyancsak *N* db „b” betűt tartalmaz, minden pozitív *N*-re. (Egy konkrét *N* esetén egy borzasztó hosszú reguláris kifejezést ugyan meg lehet adni, de az általános esetben ez nem lehetséges.)

A reguláris kifejezések felépítése I.

- Elemi kifejezések (atomok):
  - `(`*KIF*`)`: csoportosítás (műveleti sorrend felülbírálása), *KIF*-re illeszkedik
  - `()`: az üres szóra illeszkedik
  - `[`*HALMAZ*`]`: A halmaz bármely karakterének egy példányára illeszkedik. A halmazt a karakterek egymás mellé írásával adhatjuk meg.
  - `[`*ELSŐ*`-`*UTOLSÓ*`]`: mint előbb, de itt egy tartományt adunk meg
  - `[^`*HALMAZ*`]`: a halmazban *nem szereplő* bármely karakter egy példányára illeszkedik (a sortörést kivéve)
  - `.`: bármilyen karakter egy példányára illeszkedik (a sortörést kivéve)
  - `^`: a sor elejére illeszkedik
  - `$`: a sor végére illeszkedik
  - `\`*KARAKTER*: a `\` után írt speciális jelentésű karaktert közönségesként kezeli
  - *KARAKTER*: bármely közönséges karakter saját maga egy példányára illeszkedik

A reguláris kifejezések felépítése II.

- Összetett kifejezések:
  - $KIF_1 KIF_2$ (két kifejezés egymás mellé írása): Összefűzés, **konkatenáció** (concatenation). Olyan szövegre illeszkedik, amelynek első fele KIF_1-re, második fele KIF_2-re illeszkedik. Több kifejezést is összefűzhetünk.
  - KIF_1 | KIF_2 | …: Logikai MEGENGEDŐ VAGY (diszjunkció), **alternáció** (alternation). Olyan szövegre illeszkedik, amely legalább az egyik kifejezésre (**alternatívára**) illeszkedik.
  - ismételt illesztés, ismétlésszám megadása, **iteráció** (repetition, iteration):
    - *KIF*`*`: *KIF* akárhány egymást követő példányára illeszkedik (0 is)
    - *KIF*`+`: *KIF* legalább 1 egymást követő példányára illeszkedik
    - *KIF*`?`: KIF 0 vagy 1 példányára illeszkedik (azaz *KIF* opcionális)
    - *KIF*`{`*I*`}`: *KIF* pontosan *I* egymást követő példányára illeszkedik
    - *KIF*`{`*I*`,}`: *KIF* legalább *I* egymást követő példányára illeszkedik
    - *KIF*`{`*I*`,`*J*`}`: mint előbb, de legfeljebb *J* példányra illeszkedik (*I* ≤ *J*)
  - Műveleti erősség csökkenő sorrendben: iteráció, konkatenáció, alternáció

A `grep` szűrő I.

- `grep 'REGKIF' ÁLLOMÁNY(OK)`:
  - leírás: `man grep`, `info grep`
  - Kiírja a megadott állomány(ok) mindazon sorait, amelyek illeszkednek a *REGKIF* reguláris kifejezésre. Szűrőnek tekinthető.
  - A korábban említettek miatt az aposztrófok kiírása ajánlott.
  - Ha nem adunk meg állományt, akkor a szabványos bemenetről olvas.
  - `-c`: Az illeszkedő sorok tartalma helyett csak azok darabszáma jelenik meg. A `-v` opció esetén a nem illeszkedő sorok száma íródik ki.
  - `-E`: Teljes értékű, kibővített (extended) kifejezések használata. Ha ezt elhagyjuk, akkor a reguláris kifejezéseknek egy régebbi változatát kell megadnunk. Ez utóbbi jelentősen eltér a korábban bemutatottól!
  - `-e 'REGKIF'`: Akkor kell használni, ha a reguláris kifejezés `-` jellel kezdődik. Közvetlenül a *REGKIF* előtt kell állnia!
  - `-F`: *REGKIF*-ben minden karaktert közönségesként értelmez
  - `-f KIFFÁJL`: *KIFFÁJL* minden sorát egy-egy *REGKIF*-nek tekinti. Ilyenkor a bármelyik kifejezésre illeszkedő sorok jelennek meg.

A `grep` szűrő II.

  - `-i`: a kisbetűket és a nagybetűket azonosnak tekinti
  - `-n`: az illeszkedő sorok tartalma elé a sorszámukat is kiírja
  - `-o`: a sorokból csak az illeszkedő részt jeleníti meg
  - `-R`, `-r`: Ha könyvtárat adtunk meg, akkor a keresés az alkönyvárakban és azok teljes tartalmában történik (rekurzív keresés).
  - `-v`: illeszkedés helyett nem-illeszkedést vizsgál (inverzió)
  - `-w`: Csak olyan sort ír ki, amelyben legalább egy egész szó (nemcsak egy részlet) illeszkedik a reguláris kifejezésre.
- `egrep`, `fgrep`: a `grep -E` ill. `grep -F` paranccsal ekvivalensek
- A gyakorlatban a `-E` opció vagy a vele ekvivalens `egrep` parancs használata ajánlott! Ha nem így tennénk, vegyük figyelembe, hogy ezek nélkül a reguláris kifejezéseknek egy régebbi (basic) változatát kell használnunk, ahol pl. a `(` és `)` közönséges karakterek, a csoportosításra pedig a `\(` és `\)` jelölések szolgálnak (tehát a korábban látotthoz képest pont fordítva működnek). Ugyanez érvényes a `{`, `}`, `|`, `?` és `+` karakterekre is.

Az awk szűrő

- `awk 'PROGRAM' ÁLLOMÁNY(OK)`:
  - leírás: `man awk`/`gawk`, `info gawk`
  - mintakereső és -feldolgozó program saját programozási nyelvvel (AWK)
  - Sorban beolvassa a bemeneti állomány(ok) tartalmát, miközben az AWK nyelven írt *PROGRAM*-ban leírt műveleteket végrehajtja. Szintén szűrő.
  - Ha nem adunk meg állományt, akkor a szabványos bemenetről olvas.
  - A forrásprogram szövegét érdemes aposztrófok közé zárni, hogy a benne szereplő karaktereket a shell ne tekintse speciálisnak.
  - `-f` *PROGFÁJL*: a végrehajtandó programot *PROGFÁJL*-ból olvassa
- `gawk`: Az eredeti `awk` program GNU változata, GNU/Linux alatt ezt használhatjuk. Jóval többet tud elődjénél.
- `#! /bin/awk -f`: Ha az AWK forrásprogramot állományban tároljuk el, az állomány első sorába ezt a megjegyzést (parancsértelmező fejlécet) írjuk, valamint futtathatóvá tesszük az állományt, akkor az AWK programot a shell scriptek mintájára a `./`*PROGFÁJL ÁLLOMÁNY(OK)* paranccsal is lefuttathatjuk.

Az AWK forrásprogram felépítése I.

- Minden AWK forrásprogram **szabályok** (rule) sorozata. Minden szabály tartalmazhat egy **mintát** (pattern) és egy hozzá tartozó **tevékenységet** avagy **akciót** (action). Az akciót különféle **utasításokból** (statement) állíthatjuk össze.
- A szabályok alakja: *MINTA*`{`*AKCIÓ*`}`
- A szabályokat egymástól sortöréssel vagy pontosvesszővel lehet elválasztani (ld. következő dia).
- A feldolgozás során a bemenet tartalmát **rekordokra** (record) bontja, ezek alapesetben a bemenet sorai lesznek. A rekordokat szintén továbbontja **mezőkre** (field), amiket alapesetben az illető sor szavai képviselnek.
- A bemenet feldolgozása rekordonként történik. Minden rekordot megpróbál illeszteni sorban az összes szabály mintájára, az első szabálytól kezdve. Ha a rekord illeszkedett egy szabály mintájára, akkor végrehajtódik a hozzá tartozó akció. Végül az összes szabály ellenőrzése után rátér a következő rekord feldolgozására.
- A szabályok sorrendje fontos, hiszen a mintákra való illeszkedés ellenőrzése, s így az akciók végrehajtásának sorrendje ettől függ!

Az AWK forrásprogram felépítése II.

- Hiányzó minta esetén az illető akció minden rekord esetén lefut.
- A szabályokból az akciót is el lehet hagyni a kapcsos zárójelekkel együtt. A hiányzó akció ekvivalens a `{print}` akcióval, ami kiírja az egész rekord tartalmát (ld. később).
- Vigyázat! A `{}` páros az üres akciót jelöli, tehát *nem egyezik meg* az előbb említett esettel (ti. az akció elhagyásával)!
- Bármely mintát vagy utasítást folytathatjuk a következő sorban, ha az aktuális sort a `\` jellel zárjuk.
- Az akciók utasításlistája akár több sorból is állhat. Egy sorba több utasítást is írhatunk, ha őket pontosvesszővel (`;`) választjuk el egymástól. Hasonlóan, a pontosvessző használatával több szabályt is írhatunk egy sorba.
- Szóközöket és tabulátorokat tetszés szerint használhatunk a műveleti jelek, operandusok, utasítások, paraméterek, stb. között. Üres sorok szintén megengedettek.
- `#`: A sor végéig tartó **megjegyzés** (comment) kezdetét jelzi.
- Az AWK is különbséget tesz a kisbetűk és nagybetűk között!

Az AWK minták felépítése I.

- Minden minta egy logikai feltételt fogalmaz meg. Ha a feltétel teljesül egy konkrét rekord esetén, akkor azt mondjuk, hogy a rekord illeszkedik a mintára. Fontos, hogy olyan feltételt is megfogalmazhatunk, amely nem (vagy nemcsak) a rekord tartalmától függ, hanem pl. valamely változótól!
- Elemi minták:
  - `(`*`MINTA`*`)`: csoportosítás (műveleti sorrend felülbírálása), *MINTÁ*-ra illeszkedik
  - `!`*`MINTA`*: logikai tagadás (negáció)
  - `/`*`REGKIF`*`/`: igaz, ha az egész rekord illeszkedik a reguláris kifejezésre
  - *`KIF`*`~/`*`REGKIF`*`/`: igaz, ha a *KIF* kifejezés (ld. később) mint szöveg illeszkedik a reguláris kifejezésre
  - *`KIF`*`!~/`*`REGKIF`*`/`: igaz, ha a kifejezés *nem illeszkedik* a *REGKIF*-re
  - relációs kifejezések: tetszőleges kifejezés, amely relációs jelet tartalmaz
  - `BEGIN`: csak a bemenet feldolgozása előtt teljesül (ld. következő dia)
  - `END`: csak a bemenet feldolgozása után teljesül (ld. következő dia)

Az AWK minták felépítése II.

- Összetett minták:
  - *MINTA*$_1$`&&`*MINTA*$_2$: logikai ÉS (konjunkció)
  - *MINTA*$_1$`||`*MINTA*$_2$: logikai MEGENGEDŐ VAGY (diszjunkció)
  - *MINTA*$_1$`,`*MINTA*$_2$: Rekordok tartományára illeszkedik, kezdve egy olyan rekorddal, amely *MINTA*$_1$-re illeszkedik, egészen egy olyan rekordig, amely *MINTA*$_2$-re illeszkedik. Nem kombinálható semmilyen más mintával!
- A `BEGIN` és `END` mintákhoz mindig meg kell adni az akciót is! Továbbá ezek a speciális minták nem kombinálhatók semmilyen más mintával, valamint nem alkalmazható rájuk a csoportosítás és a negáció sem!
- A `BEGIN` mintához tartozó akció *pontosan egyszer* hajtódik végre, mégpedig a legelső bemeneti rekord feldolgozása előtt. Ez akkor is így történik, ha több bemeneti állományt adtunk meg.
- Hasonlóan, az `END` mintához tartozó akció is *pontosan egyszer*, az utolsó bemeneti rekord feldolgozása után hajtódik végre. Ezt az `awk` program befejeződése követi.

Konstansok használata az AWK-ban

- **Szám** avagy **numerikus konstansok** (numeric constant):
  - egész számok (pl. `12`)
  - valós törtszámok tizedesponttal (pl. `25.3`)
  - egész vagy valós szám hatványkitevővel (pl. `1.234e+2`=`123.4`)
- **Szöveges** avagy **sztring konstansok** (string constant):
  - `"SZÖVEG"`
  - `""`: **üres sztring** (0 karakter hosszúságú szöveg)
  - A szövegben a `\` speciális (az ún. escape-karakter), így használhatók pl. a következő escape-szekvenciák: `\\` (közönséges `\`), `\"` (közönséges idézőjel), `\n` (sortörés), `\t` (tabulátor).
- **Konstans reguláris kifejezések** (regular expression constant):
  - `/REGKIF/`
  - A reguláris kifejezésen belül a `\` speciális, így használhatók a `\\` (közönséges `\`) és `\/` (közönséges `/`) karakterpárosok.

Változók használata az AWK-ban I.

- Az AWK-ban a változók élettartama *dinamikus*: az első használatkor automatikusan létrejönnek (nem kell őket deklarálni).
- A változók neve betűket, számokat és aláhúzásjelet (_) tartalmazhat, és nem kezdődhet számjeggyel.
- Változók típusai:
  - numerikus változók (valós számokat tárolnak)
  - szöveges változók avagy sztringek (string)
  - egydimenziós tömbök (ld. később)
- A tömböket kivéve minden változó típusa *dinamikus*, azaz a használattól függően változik! Ez a tömbelemekre is vonatkozik (ld. később).
- Egy változó típusát *nem lehet* tömbről numerikusra vagy sztringre változtatni, és viszont!
- A változók értékét az `awk` automatikusan konvertálja számmá vagy szöveggé, szintén a használati módtól (művelettől, függvénytől) függően. Ha a szöveget nem lehet számmá konvertálni (mert nem egy érvényes alakú számot tartalmaz), nullát kapunk.

Változók használata az AWK-ban II.

- Manuális konverzió:
  - szövegből szám: adjunk hozzá `0`-t
  - számból szöveg: fűzzük hozzá az üres sztringet (`""`)
- *NÉV*`=`*ÉRTÉK*:
  - Értékadás egy létező változónak, vagy új változó létrehozása. A változó típusa *ÉRTÉK* típusa lesz.
  - A C programozási nyelv egyéb értékadó, növelő és csökkentő műveletei is használhatók (ld. később).
  - Az *ÉRTÉK* természetesen nemcsak konstans, hanem kifejezés is lehet.
  - Többszörös értékadás ($NÉV_1$`=`$NÉV_2$`=`*ÉRTÉK*) is megengedett.
- *NÉV*:
  - a változó aktuális értékét jelöli
  - Definiálatlan (ti. amelyiknek eddig nem adtunk értéket) változó értéke az üres sztring (`""`) ill. `0`.

Az AWK beépített változói I.

- Az `awk` program indulásakor már létezik jónéhány különleges, **beépített változó** (built-in variable). Ezek neve egységesen csupa nagybetűből áll, és tartalmuk egyrészt a felhasználónak szóló fontos információkat hordoz, másrészt némelyikük az `awk` program működését ill. a bemenet feldolgozásának módját vezérli.
- `FILENAME`: Az aktuális bemeneti állomány neve, ill. `-` a szabványos bemenet esetén. A `BEGIN` minta akcióján belül definiálatlan.
- `FNR`: az aktuális rekord sorszáma az aktuális bemeneti állományon belül
- `FS`: **bemeneti mezőhatároló** karakter (input field separator, ld. később), kezdetben a szóköz
- `IGNORECASE`: Ha értéke nemzérus, akkor a sztringek összehasonlítása ill. a reguláris kifejezések illesztése nem különbözteti meg a kisbetűket a nagyoktól. Alapesetben értéke definiálatlan (effektíve nulla).
- `NF`: az aktuális rekord mezőinek száma (number of fields)
- `NR`: Az aktuális rekord sorszáma az eddig feldolgozott bemenet tekintetében. Egy bemeneti állomány ill. a szabványos bemenet esetén egyenlő az `FNR`-rel.

Az AWK beépített változói II.

- `OFS`: **Kimeneti mezőhatároló** (output field separator, ld. később), kezdetben a szóköz. Értéke tetszőleges szöveg lehet, nemcsak egy karakter.
- `ORS`: **Kimeneti rekordhatároló** (output record separator, ld. később), kezdetben a sortörés. Ez is tetszőleges szöveget tartalmazhat.
- `RS`: **bemeneti rekordhatároló** karakter (input record separator, ld. később), kezdetben a sortörés

Egy egyszerű awk program

- Maximum és minimum értékek megkeresése az első oszlopban

```
NR == 1 {m=$1 ; p=$1}
$1 >= m {m = $1}
$1 <= p {p = $1}
END { print "Max = " m, " Min = " p }
```

Tömbök használata az AWK-ban I.

- Lehetőség van egydimenziós tömbök (vektorok) használatára is. Fontos, hogy a tömb méretét *nem kell* előre lerögzíteni, továbbá a tömbelemek indexe *tetszőleges szöveg* lehet (a számokat is szöveggé konvertálja)! Az ilyen tömböket **asszociatív tömböknek** (associative array) nevezik.
- A tömbök nevét a változónevek mintájára adhatjuk meg.
- A tömb vegyesen tartalmazhat numerikus és szöveges elemeket is!
- *NÉV*`[`*INDEX*`]=`*ÉRTÉK*:
  - Értékadás egy létező tömbelemnek, vagy új elem beszúrása. Az elem típusa *ÉRTÉK* típusa lesz. A tömb is létrejön, ha még nem létezett.
  - A C programozási nyelv egyéb értékadó, növelő és csökkentő műveletei is használhatók (ld. később).
  - Az *INDEX* és az *ÉRTÉK* konstans és tetszőleges kifejezés is lehet.
- *NÉV*`[`*INDEX*`]`:
  - a megadott indexű tömbelem aktuális értékét jelöli
  - Definiálatlan elem értéke az üres sztring (`""`) ill. `0`.

Tömbök használata az AWK-ban II.

- *INDEX* `in` *NÉV*: Ez a logikai reláció csak akkor igaz, ha a tömbnek van *INDEX* indexű eleme. Lásd még: `for`, `while`, `do`…`while`, `if` utasítások.
- `delete` *NÉV*`[`*INDEX*`]`: a megadott indexű tömbelem kitörlése
- `delete` *NÉV*: A tömb összes elemének kitörlése. Vigyázat, a tömb továbbra is létezni fog, csak üres lesz!

Az AWK kifejezések felépítése I.

- A minták és az utasítások megadásához használhatunk különféle **kifejezéseket** (expression). Az ezeket felépítő építőkövek: konstansok, változók, műveleti jelek, függvények, segédjelek (pl. zárójelek, vessző).
- `(KIF)`: csoportosítás (műveleti sorrend felülbírálása)
- Műveletek, relációk:
  - Aritmetika valós számokon: `+` (előjel és összeadás is), `-` (előjel és kivonás is), `^` (hatványozás), `*`, `/`, `%` (osztási maradék)
  - Növelés (increment), csökkentés (decrement): `++`, `--` (mindkettő prefix és postfix használatban is)
  - Sztring összefűzés, konkatenáció: egymás mellé írás, illetve szóköz
  - Mező értékének használata (mezőhivatkozás): `$KIF` (ld. később)
  - Értékadás (assignment): `=`, `+=`, `-=`, `*=`, `/=`, `%=`, `^=`
  - Összehasonlító relációk: `<`, `<=`, `>`, `>=`, `==`, `!=`
  - Mintaillesztő relációk: `KIF~/REGKIF/`, `KIF!~/REGKIF/` (ld. következő dia)
  - Tömbelem létezésének vizsgálata: `INDEX in NÉV`

Input bekérése

- **`getline nev < "-”`**
  - Standard inputról olvas a nev változóba

- **Példa**

```
BEGIN {
      printf ”Irja be a nevet:"
      getline nev < "-"
      printf "%s kora:",nev
      getline kor < "-"
      print  nev, ", kovetkezo evben " kor + 1
  ”eves lesz”
  }
```

Az AWK kifejezések felépítése II.

  - Logikai műveletek: `!` (negáció), `&&` (konjunkció), `||` (diszjunkció)
  - Feltételes kifejezés: KIF_1`?`KIF_2`:`KIF_3 (mint a C prog. nyelvben)
- Az összehasonlítás csak akkor történik numerikusan, ha a reláció mindkét oldalán szám konstans, numerikus változó vagy mezőhivatkozás áll. Máskülönben az értékek szövegesen (lexikografikusan, azaz az ábécé rendet követve) lesznek összehasonlítva!
- A logikai műveletek, a feltételes kifejezés és a vezérlési szerkezetek szempontjából **hamisnak** (false) minősül az üres sztring (`""`) és a nulla. Minden más érték **igaznak** (true) számít.
- A relációk numerikus értéke igaz esetén `1`, különben `0`. Ez az összehasonlító és mintaillesztő relációkra, továbbá az `in` relációra és a logikai műveletekre is vonatkozik.
- A mintaillesztő relációk igazak, ha a bal oldali kifejezés mint szöveg illeszkedik (`~`) ill. nem illeszkedik (`!~`) a jobb oldali reguláris kifejezésre.
- A feltételes kifejezésben először KIF_1 lesz kiértékelve. Ha igaz, akkor KIF_2, különben KIF_3 lesz kiszámolva, s ők adják a kifejezés értékét is.

Az AWK kifejezések felépítése III.

- Numerikus függvények:
  - Trigonometria: `sin(`*KIF*`)`, `cos(`*KIF*`)`
  - `sqrt(`*KIF*`)`: négyzetgyökvonás
  - Exponens, logaritmus: `exp(`*KIF*`)`, `log(`*KIF*`)`
  - `int(`*KIF*`)`: egésszé konvertálás csonkolással (truncation)
- Szöveges függvények:
  - `index(`*SZÖVEG*`,`*RÉSZ*`)`: A *RÉSZ* szöveg legelső előfordulásának pozíciója *SZÖVEG*-ben. Ha nincs ilyen rész, akkor nullát ad vissza.
  - `length(`*SZÖVEG*`)`: a megadott sztring hossza karakterekben
  - `split(`*SZÖVEG*`,`*TÖMB*`,`*HAT*`)`: *SZÖVEG*-et a *HAT* határolójel mentén darabokra bontja, a darabokat a megadott tömbben eltárolja, majd visszaadja a darabok számát. A *SZÖVEG* változatlan marad. A tömb elemei a darab sorszámával (pont nélkül) lesznek indexelve. *HAT* reguláris kifejezés is lehet.

Az AWK kifejezések felépítése IV.

- `substr(`*SZÖVEG*`,`*IND*`)`: a szöveg *IND* sorszámú karakterén kezdődő részét adja vissza
- `substr(`*SZÖVEG*`,`*IND*`,`*HOSSZ*`)`: mint előbb, de legfeljebb *HOSSZ* karakterből álló részt ad vissza
- `tolower(`*SZÖVEG*`)`: visszaadja a *SZÖVEG* kisbetűssé konvertált értékét
- `toupper(`*SZÖVEG*`)`: visszaadja a *SZÖVEG* nagybetűssé konvertált értékét

Mezők elérése az AWK-ban I.

- A bemenet rekordokra bontását, ill. azoknak mezőkre bontását két beépített változó vezérli. Az `RS` változó tartalma egy karakter (alapesetben sortörés), ez jelzi a rekordokat elválasztó karaktert. Hasonlóan, az `FS` változó tartalma (alapesetben szóköz) határozza meg, mi határolja a mezőket a rekordokon belül. Ha az `FS` értéke a szóköz (alapeset), akkor a mezőket legalább egy szóköz vagy tabulátor választja el.
- Az aktuális rekord mezőinek a számát az `NF` beépített változó tárolja.
- A mezők típusa ugyancsak numerikus vagy szöveges lehet, az aktuális használattól függően. Összehasonlításkor a mezők tartalmát számnak tekinti az `awk`, ha az valóban egy érvényes számot tartalmaz, továbbá ha a másik tag szám konstans, numerikus változó vagy mezőhivatkozás (ld. következő dia).

Mezők elérése az AWK-ban II.

- `$KIF`:
  - Az aktuális rekord megadott sorszámú mezőjének tartalma. Ezt a jelölést **mezőhivatkozásnak** (field reference) nevezzük.
  - Tetszőleges kifejezést is használhatunk, pl. `$(2*3)` a hatodik mezőt jelzi. Természetesen a negatív értékek nem megengedettek.
  - `$NF`: az aktuális rekord utolsó mezőjének tartalma
  - `$0` (dollárjel és nulla): az aktuális rekord teljes tartalma
- `$KIF=ÉRTÉK`:
  - egy adott mező – ill. `KIF` = 0 esetén a rekord – értékének módosítása
  - Ha `$0` tartalmát változtatjuk meg, akkor minden mező új értéket kap. Ha viszont egy mező tartalmát módosítjuk, akkor `$0` értékét az `awk` újraépíti oly módon, hogy a mezőket az `OFS` értéke határolja majd el.
  - Ha `KIF` > `NF`, akkor a mezők számát kibővíti, és `NF`-et is módosítja. Szükség szerint a közbülső helyekre új mezőket szúr be, ezek értéke az üres sztring (`""`) lesz. Végül pedig `$0` tartalmát is újraszámítja az előbb leírt módon.

Az AWK akciók felépítése I.

- A szabályok akcióját alkotó utasítások építőelemei:
  - a korábban már látott `delete` utasítás
  - értékadó, növelő és csökkentő kifejezések
  - vezérlési szerkezetek (ld. lent)
  - egyéb utasítások (ld. következő dia)
- `{`*UTASÍTÁSOK*`}`: összetett utasítás, utasításblokk/-lista
- `if (`*FELTÉTEL*`)` *UTASÍTÁS* `else` *UTASÍTÁS*$_0$: szelekciós vezérlés
- `while (`*FELTÉTEL*`)` *UTASÍTÁS*: előfeltételes ismétléses vezérlés
- `do` *UTASÍTÁS* `while (`*FELTÉTEL*`)`: végfeltételes ismétléses vezérlés
- `for (`*KIF*$_1$`;`*KIF*$_2$`;`*KIF*$_3$`)` *UTASÍTÁS*: számlálásos ismétléses vezérlés
- `for (`*INDEX* `in` *NÉV*`)` *UTASÍTÁS*: Diszkrét ismétléses vezérlést valósít meg. Az *INDEX* változó sorban felveszi a *NÉV* nevű tömb elemeinek indexét, miközben a megadott utasítás végrehajtódik.

Az AWK akciók felépítése II.

- `break`, `continue`: Kilépés a ciklusból, ill. rátérés a ciklus következő iterációjára (`for`, `while` és `do`…`while` esetén használhatók). Mindig az őket körbevevő *legbelső ciklusra* vonatkoznak!
- `exit`: A bemenet feldolgozásának azonnali befejezése. Ha nem az `END` minta akciójában használjuk, akkor az esetleges `END` minta akciója végrehajtódik, különben az `awk` rögtön befejezi működését.
- `print` *`LISTA`*: Kiírja a vesszővel tagolt kifejezéslista tagjainak értékét, majd az `ORS` tartalmát (alapesetben egy sortörést). A kiírt értékek közé az `OFS` tartalma kerül (alapesetben egy szóköz).
- `print`: ekvivalens a `print $0` utasítással (az aktuális rekord teljes tartalmát kiírja)
- `printf` *`FORMÁTUM`*`,` *`LISTA`*: formázott kiíratás (mint a C prog. nyelvben)
- `next`: Azonnal nekikezd a következő bemeneti rekord feldolgozásához, a legelső szabály mintáját tesztelve. Ha nincs több rekord, akkor az esetleges `END` minta akciójával folytatja.

Processzek és munkafolyamatok I.

- A **folyamat** avagy **processz** (process) nem más, mint egy saját adatterülettel rendelkező futó programpéldány. Minden processz kap egy egyedi **processz-azonosítót** (process ID, PID), ami egy pozitív egész szám.
- A shellből indított programokat (processzeket) **munkafolyamatnak** (job) nevezzük. Ha több parancsot csővezetékbe kapcsolunk, akkor azok ugyanahhoz a munkafolyamathoz fognak tartozni. A munkafolyamatokat is egy egyedi pozitív egész szám azonosítja.
- Minden **processz állapota** (state) a következők valamelyike lehet:
  - **előtérben futó** (foreground): Pontosan 1 processz lehet előtérben. Csak ez a folyamat képes a billentyűzetről olvasni vagy a képernyőre írni.
  - **háttérben futó** (background): Több processz is futhat a háttérben. Ők nem érhetik el sem a billentyűzetet, sem a képernyőt. Ha ezt mégis megkísérlik, azonnal felfüggesztett állapotba kerülnek.
  - **felfüggesztett** avagy megállított (suspended, stopped): Az ilyen folyamatok futása ideiglenesen félbeszakadt. Később még folytatódhatnak, de csak külső beavatkozásra (nem maguktól)!

Processzek és munkafolyamatok II.

- Fontos, hogy a shellből csak a munkafolyamatok állapotát befolyásolhatjuk, a többi folyamatot legfeljebb leállítani tudjuk.
- A valamely processz által indított újabb folyamatot **gyerek-processznek** (child process) nevezzük. Így pl. minden alshell egy gyerek-processz.
- `$$`: Az aktuális shell vagy script processz-azonosítóját tartalmazza ez a speciális paraméter.
- `ps`: a processzlista és a folyamatok állapotának megjelenítése
- `jobs`: az aktuális shellhez tartozó munkafolyamatok listájának megjelenítése
- `%`*`SZÁM`*: Bármelyik munkafolyamatra ezzel a jelöléssel hivatkozhatunk, ahol *`SZÁM`* a munkafolyamat azonosítója.
- CTRL+Z: az előtérben futó munkafolyamat futásának felfüggesztése
- `fg %`*`SZÁM`*: egy háttérben futó vagy felfüggesztett munkafolyamatot előtérbe hoz
- `bg %`*`SZÁM`*: egy felfüggesztett munkafolyamatot háttérbe küld
- *`PARANCS`* `&`: a megadott parancs elindítása a háttérben

Processzek és munkafolyamatok III.

- CTRL+C: az előtérben futó program futásának befejezése (munkafolyamat leállítása)
- `kill` *AZON*:
  - A megadott azonosítójú processz vagy munkafolyamat futásának befejezése (folyamat leállítása).
  - Az *AZON* alakja *SZÁM* (processz) vagy `%`*SZÁM* (munkafolyamat) lehet.
  - A `root` felhasználó bármilyen processzt képes leállítani. A többi felhasználó viszont csak az őhozzá tartozókat, azaz a saját maga által indítottakat és az azok által indított gyerek-processzeket, továbbá pl. a bejelentkezésnél elinduló shellt tudja leállítani.
- Előfordulhat, hogy az illető processz – általában programhiba miatt – nem reagál sem a CTRL+C billentyű-kombinációra, sem a `kill` parancsra. Ilyenkor erőszakosabb módszerhez kell folyamodni (ld. következő diák).
- A processzek befejeződésükkor leállítják az általuk indított gyerek-processzeket is. Ha ez valamiért nem sikerülne, akkor ún. halott avagy zombi (dead, zombie) processzek keletkezhetnek.

Szignálok I.

- Bizonyos kritikus események bekövetkezése esetén a processzek jelzéseket avagy **szignálokat** (signal) kapnak. Ezek legtöbbször a kerneltől származnak, de a felhasználói programok is küldhetnek szignálokat.
- Leírás: `man 7 signal`
- Minden szignált egy név és egy sorszám azonosít.
- Néhány fontosabb szignál (zárójelben a szignál sorszáma):
  - `SIGINT` (2): processz futásának befejezése (mint a CTRL+C)
  - `SIGKILL` (9): processz futásának erőszakos befejezése (ld. következő dia)
  - `SIGTERM` (15): processz futásának befejezése (ld. következő dia)
  - `SIGCONT` (18): felfüggesztett processz háttérbe küldése (mint a `bg` parancs)
  - `SIGSTOP` (19): processz futásának felfüggesztése (mint a CTRL+Z)
- A `SIGCONT` és a `SIGSTOP` szignálok fenti sorszáma bizonyos UNIX változatokban eltérő lehet! GNU/Linux alatt viszont ezek érvényesek.

Szignálok II.

- `kill -`*`SZIGNÁL AZON`*:
  - szignál küldése a megadott azonosítójú processznek
  - A *`SZIGNÁL`* mind sorszám, mind név formájában megadható.
  - A `-`*`SZIGNÁL`* elhagyása esetén egy `SIGTERM` szignált küld.
  - Az *`AZON`* egy processz vagy munkafolyamat azonosítóján kívül `-1` (kötőjel és egy) is lehet. Ilyenkor a szignál az összes processznek el lesz küldve (ld. még a következő megjegyzést).
  - A `kill` parancsnál korábban elmondottak érvényesek a szignálok küldésére is. A `root` bármelyik processznek küldhet szignált, a többi felhasználó viszont csak az őhozzá tartozóknak.
- Bármelyik processzt leállíthatjuk, ha `SIGKILL`-t küldünk neki. Sem ezt, sem a `SIGSTOP` szignált nem hagyhatja figyelmen kívül egyetlen processz sem.
- `killall -`*`SZIGNÁL PARANCS`*: A megadott parancsot futtató összes processznek szignált küld.

Egyéb parancsok a processzek felügyeletéhez

- Programok módosított futtatása:
  - `chroot`: program futtatása másik gyökérkönyvtárral (`/`)
  - `env`: program futtatása módosított környezetben (új környezeti változókkal, stb.)
  - `nice`: program futtatása módosított ütemezési prioritással
  - `nohup`: Az így indított program akkor sem fejeződik be, ha az őt indító felhasználó kijelentkezik.
  - `renice`: futó program ütemezési prioritásának módosítása
  - `su`: shell futtatása más felhasználóként
  - `sudo`: tetszőleges program futtatása más felhasználóként
- `at`: program futtatása egy adott későbbi időpontban
- `cron`, `crontab`: program futtatása rendszeres időközönként
- `fuser`: Kiírja, hogy mely processzek használnak egy adott állományt, könyvtárat vagy kommunikációs végpontot (socket-et).
- `time`: program futtatása, majd annak befejeződése után a futási idő kiírása


Vége