Operációs rendszerek alapjai

1. gyakorlat

Tematika

- Történelem, linux alapok
- Terminál tartós használata
- ITK-s szerverek, ssh, scp
- Shell script
- Szövegkezelés: grep, sed, awk
- Terminál - internet interakció
- Automatizálás, ütemezés
- Windows, cmd, PowerShell
- Esettanulmányok
- *

Gyakorlat célja

- Kényelmes működés terminálban (platformtól függetlenül)
- Alapvető műveletek elvégzése és automatizálása script segítségével
- Ismerkedés néhány nagyobb hangvételű feladattal
- Programozás és fordítás terminálban

Követelmények

- Beadandók
  - Három darab: 10, 15, 15 pontért
- ZH
  - Félév végén: 60 pontért
- Szorgalmi?
- Ponthatárok
  - 0-59 elégtelen
  - 60-69 elégséges
  - 70-79 közepes
  - 80-89 jó
  - 90-100+ kiváló

Mai gyakorlat

- Történelem
- Linux alapok
  - fájlrendszer
  - alapvető parancsok
  - átirányítások, pipe-ok

Unix

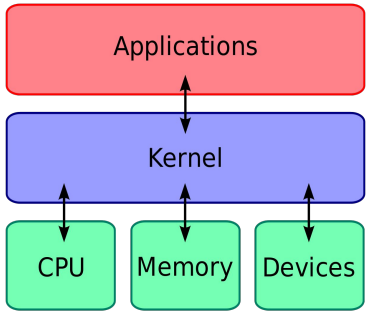
- 1970 körül assembly-ben írták
  - UNICS - Uniplexed Operating and Computing System
  - rendszermag (kernel), héj (shell), szövegszerkesztő és assembler
  - 1973-ban átírták C-re a könnyebb portolás érdekében
  - ma már licenszelhető védjegy, meg kell felelni a specifikációknak
    - pl. Mac OS X
- Moduláris design - a programok interakciója a fontos
- Fontos emberek
  - Ken Thompson
  - Dennis Ritchie
  - Brian Kernighan

GNU - GNU’s not Unix

- 1980-as években Richard Stallman hirdette meg a GNU Project-t
  - Unix-szerű operációs rendszer: nagyjából teljesíti a specifikációkat, de ingyenes
- Programok
  - Emacs - szövegszerkesztő
  - GCC - GNU Compiler Collection (C, C++, Objective C, Fortran stb.)
  - GIMP - GNU Image Manipulation Program
  - GNOME - GNU Network Object Model Environment grafikus felület
  - GRUB - Grand Unified Boot Loader
- Mit jelent, hogy szabad szoftver?
  - szabadon használható, terjeszthető, módosítható
  - (L)GPL - (Lesser) General Public License

Linux

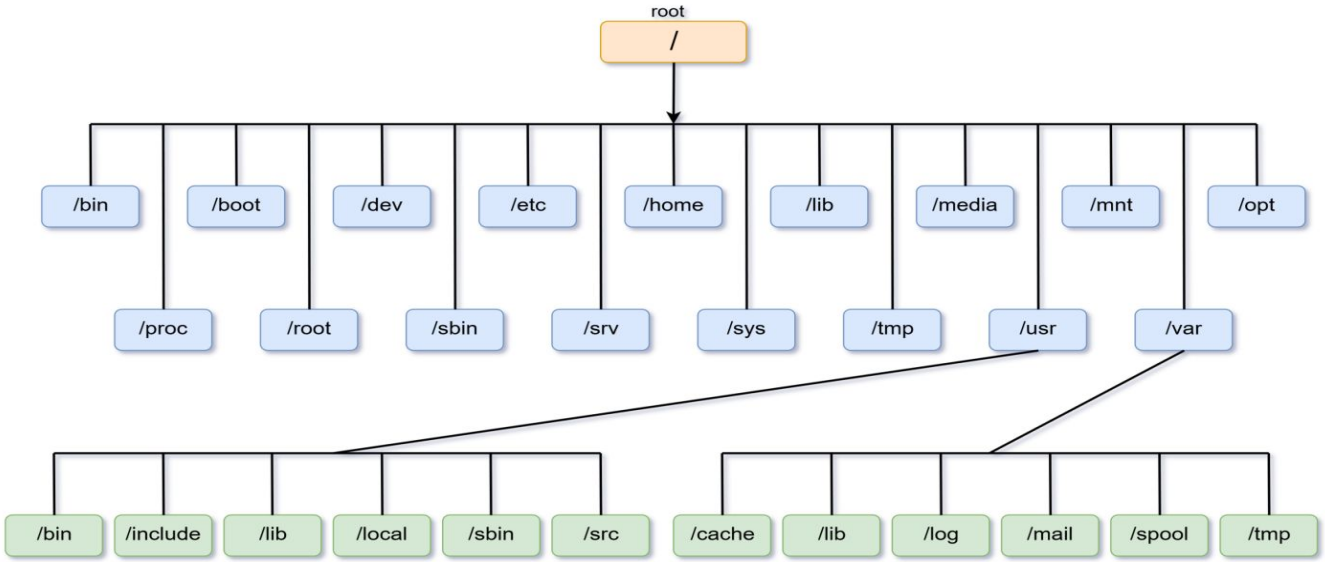
- 1991-ben Linus Torvalds írta főleg C-ben
  - nem voltak elérhető kernelek (GNU kernel is csak akkoriban készült el)
- Open-source GPL-2.0 licenc alatt github link
  - még ma is több ezren szerkesztik Linus menedzselésével
- Személyi operációs rendszernek szánta
  - minden területen domináns, kivéve a személyi felhasználást :(
- Linux kernel felé rengeteg rendszert írnak
  - Android
  - disztribúciók



Linux disztribúciók

- Debian
  - ez van az ITK-s gépeken
- Ubuntu
  - Debian alapú, könnyű Windows után
- Mint
  - Ubuntu alapú, flugi ezt használja (úgy tudom)
- Elementary OS
  - könnyű Mac OS X után
- Arch
  - hardcore
- Manjaro
  - gyakorlatilag egy konfigurált Arch

Linux fájlrendszer - alapvető struktúra



Linux fájlrendszer - fontosabb mappák

/	virtuális könyvtár gyökere
/bin	bináris fájlok
/boot	boot-hoz szükséges fájlok
/dev	eszközmeghajtó fájlok (merevlemezek)
/etc	konfigurációs fájlok
/home	felhasználók könyvtárai
/lib	library-k
/media	hordozható eszközök

Linux fájlrendszer - fontosabb mappák

/opt	third-party szoftverek (MATLAB, Qt stb.)
/root	rendszergazda könyvtára
/run	futási idejű adatok
/sys	hardware információk
/tmp	ideiglenes fájlok
/usr	rendszerszintű programok
/var	gyakran változó fájlok, pl. log

Linux fájlrendszer - inode (index-node)

- Minden fájlhoz tartozik egy inode
  - `stat` paranccsal elérhetjük az inode információit
  - POSIX - Portable Operating System Interface szabvány
    - eszköz ID
    - sorszám
    - linkek száma
    - engedélyek
    - user ID, group ID
    - méret
    - létrehozás, hozzáférés, módosítás időbélyege
    - IO blokkméret
    - elfoglalt blokkok száma

Linux fájlrendszer - linkek

- Hard link
  - adott inode-ra (lemezterületre) mutató link
  - link számláló: ha egy inode-ra nulla hard link mutat, akkor törlésre kerül
  - létrehozás:
    - `link oldfile newfile`
    - `ln oldfile newfile`
- Symbolic link
  - elérési útvonal akár könyvtárra is
  - nem növeli a link számlálót
  - lehet a mutatott objektumot kitörölni és újra létrehozni
  - létrehozás:
    - `ln -s oldfile newfile`

Linux fájlrendszer - fájl típusok

- `ls` parancs
  - aktuális mappa listázása
  - `-l` (long) kapcsolóval több infót is láthatunk (engedélyek, tulajdonos, fájlméret byte-ban stb.)
  - `-h` (human) kapcsolóval ember által értelmezhető fájlméret
- Fájl típusa az első karakter
  - sima fájl: –
  - mappa: `d`
  - symlink: `l`
  - pipe: `p` (később)
  - socket: `s`
  - blokk eszköz: `b` (random access eszközhöz, pl. /dev/sda lemez partícióhoz)
  - karakter eszköz: `c` (soros bemenet vagy kimenet eszközhöz, pl. /dev/null nulleszközhöz)

Linux fájlrendszer - fájl engedélyek

- Következő 3x3 karakter
  - tulajdonos engedélyei
  - csoport engedélyei
  - többi felhasználó engedélyei
- Összesen 9 bit
  - `r`: olvasható
  - `w`: írható
  - `x`: futtatható
- `chmod` - change mode paranccsal módosítható
  - `chmod +x file: -rw-rw-r--` **=>** `-rwxrwxr-x`
  - `chmod 777 file`: bármi **=>** `-rwxrwxrwx`

Navigálás a fájlrendszerben

- `ls` - list
  - néhány kapcsoló:
    - `-a (-A)` - rejtett fájlok kiírása (`.` és `..` kihagyásával)
    - `-R` - rekurzívan listáz
- `pwd` - print working directory
- `cd` - change directory
  - abszolút vagy relatív elérési útvonal
  - speciális mappák
    - aktuális mappa: `.`
    - eggyel fentebbi mappa: `..`
    - felhasználó otthoni mappája: ~

Fájlkezelés

- `touch`
  - időbélyeg módosítása
  - ha nem létezik a fájl, akkor létrehozza
- `cp` - copy
  - `-r` kapcsolóval rekurzívan másol
- `mv` - move
  - átnevezésre is alkalmas
- `mkdir` - make directory
- `rm` - remove
  - `-r` kapcsolóval rekurzívan töröl
- `cat` - concatenate
  - fájlok összefűzése és kiírása

Átirányítások

- `echo` paranccsal kiírathatunk terminálra
- `<, >`
  - felülír fájlokat
  - `echo heloszia > helo.txt`
- `<<, >>`
  - hozzáfűz fájlokhoz
  - `echo heloszia megint >> helo.txt`

Fájl deskriptorok

- 0 - standard input
- 1 - standard output
- 2 - standard error
- Ezekbe és ezekből lehet átirányítani
  - hibák logolása: `command 2>> errors.log`
  - parancs csendes futtatása (nincs kimenet): `command 1> /dev/null`
  - kimenet hibaként megjelenítése: `command 1>&2`

Pipe

- Parancsok/processzek kimenetét beköthetjük más parancsok bemenetére a `|` karakterrel
  - szinkron kommunikáció
  - így elég komplex one-liner programokat is lehet írni
- Egyszerű példa: tegyük fel, hogy az első öt objektum érdekel csak az aktuális mappában
  - `head -n`
    - kiírja egy fájl első n sorát, alapból n=10
    - bemenetet is elfogad
  - `ls -l | head -5`
- Probléma: mi van, ha aszinkron kommunikáció kell?
  - megoldási lehetőség egy ideiglenes fájl használata az üzenet elmentésére és beolvasására
  - merevlemezre írunk, úgyhogy ez lassú és elég fapados

Elnevezett pipe

- Processzek közti aszinkron kommunikációt könnyíti meg
  - segítségével nem kell csinálni egy ideiglenes fájlt, ami tárolja az adatot, hiszen a memóriát használja
- Létrehozás
  - `mkfifo my_pipe`
- Bemenet (több is lehetséges!)
  - `ls -l > my_pipe`
- Kimenet (egyszeri kiolvasás, minden adatot megkapunk)
  - `cat my_pipe`
- Törlés
  - `rm my_pipe`

Hasznos parancsok

- `man command`
  - adott parancs manuál oldalának megnyitása
  - online is elérhető pl. itt
- `apropos text`
  - keresés parancsok összefoglalójában
- `whatis command`
  - parancs összefoglalója

1	parancsok
2	rendszerhívások
3	könyvtári függvényhívások (C nyelv)
4	speciális fájlok (/dev)
5	fájlformátumok és konvenciók
6	játékok
7	egyéb
8	rendszerkarbantartó parancsok
9	kernel rutinok

Következő gyakorlat

- Linux telepítése (akár Windows mellé)
- ITK-s szerverek, ssh
  - otthonról géptermi elérés
- Shell script-k írása
- Terminál alapú szövegszerkesztők:)
- (Környezeti) változók, főleg PATH
- Binárisok

Érdekességek, olvasnivalók

- Real Programmers Don't Use PASCAL
- Linus Torvalds Q&A at Aalto University
- Linus Torvalds Q&A at DebConf14
- Operating Systems: from 0 to 1
  - prerequisites
    - basic concepts of electricity: atoms, electron, proton, neutron, current flow, Ohm’s law
    - C programming
    - Linux basics
      - navigation
      - invoke commands with options
      - piping

Operációs rendszerek alapjai

2. gyakorlat

Mai gyakorlat

- Otthoni Linux használat
- Ssh és ITK-s szerverek
- Shell script alapok
  - (környezeti) változók
  - string, egész aritmetika
  - adatszerkezetek

Linux telepítése

- Rendes telepítéshez szükséges
  - egy számítógép
  - egy ISO lemezkép-fájl
  - egy formázható pendrive (vagy CD)
- Virtuális géphez szükséges (lehet egyébként Linux-ból telepíteni virtuális Windows-t!)
  - egy számítógép
  - például VirtualBox
  - egy ISO lemezkép-fájl

Linux telepítése - számítógép előkészítése

- Ha megválnál a Windows-tól, akkor készen is vagyunk
- Ha nem állsz készen a Windows nélküli életre
  - készítsd elő a merevlemezed partícióit, ahogy szeretnéd
  - javaslat: a Windows és Linux partíció mellett legyen egy közös, amiben közös fájlok vannak (ez a közös partíció legyen NTFS fájlrendszer, a Windows csak azt tudja használni)
  - kapcsold ki Windows-on a gyors indítást

Linux telepítése - lemezkép-fájl

- Lemezkép-fájl
  - a kedvenc disztribúciód honlapjáról le tudod tölteni
  - ha először telepítesz, akkor érdemes LTS (Long-Term Support) verziót
- Telepítő pendrive
  - fontos a pendrive tárhelyének mérete, például Ubuntu esetén legalább 4 GB kell
  - Rufus programmal formázd a pendrive-t és égesd rá a lemezkép-fájlt

Linux telepítése

- A pendrive behelyezése után indítsd újra a gépet
- Lépj be a BIOS-ba
  - amikor kiír valami olyasmit, hogy “to interrupt normal setup, press Enter”, akkor kell az adott gombot megnyomni
- A bootolható eszközök közül válaszd ki a pendrive-odat
- Csináld, amit a telepítő mond
- Ha bármelyik lépés gondot okoz, akkor keress rá a problémára például azzal az információval kiegészítve, hogy milyen géped van

VirtualBox használata

- A program telepítése és egy lemezkép-fájl beszerzése után csináld azt, amit a User Manual mond
- Telepíthető Windows-ra, Linux-ra és Mac-re is
- A megfelelő lemezkép-fájllal telepíthető vele Windows, Linux és Mac is

PuTTY - ha nem telepítenél

- TTY - teletypewriter
- Ingyenes SSH és Telnet kliens Windows-ra
- Letöltés után tetszőleges szerverrel tudsz SSH kapcsolatot létesíteni

SSH - Secure Shell (RFC 4252-4256)

- Régebbi protokollok titkosítás nélkül kommunikáltak a szerverrel
  - így könnyen lehallgathatók volt az egész kommunikáció (felhasználók, jelszavak, fájlok stb.)
  - például: Telnet, rlogin (remote login), rsh (remote shell)
- TCP/IP protokollt használ alapértelmezetten a 22-es porton
- Adat csomagokra bontása, csomagok tömörítése (opcionális) és titkosítása
- Titkosítás fajtája a szervertől és klienstől függ, általában több lehetőség van
  - Nyilvános kulcsú titkosítás
- Ezen felül egy kapcsolati réteg létrehoz egy vagy több folyamatosan nyitott csatornát
- Lehetőség van kapcsolatok továbbítására is
- Lehetőség van grafikus adatokat is továbbítani, így nem “csak” egy terminál adott (X forwarding opciót kell bekapcsolni, Windows-hoz kell X server)

ITK-s szerverek

- users.itk.ppke.hu
  - web szerver, bárki csinálhat rá weboldalt
  - kb. 5 éve nem frissítettek rajta semmit de vannak 10 éves dolgok is (PHP 5, Python 2.6.6, GCC 4.4.5)
- cortex.itk.ppke.hu
  - ezen érdemes futtatni bármilyen kódot (különösen a hosszan futó programokat)
  - Python 3.4.2, GCC 4.9.2 és alapvetően mindenből frissebb van (bár ezek is gyakran 5+ éves verziók)
- medea.itk.ppke.hu
  - Windows-os profilokat tárolja, elérhető rajta minden fájl
- tempus.itk.ppke.hu
  - Linux-os profilokat tárolja

Szövegszerkesztők

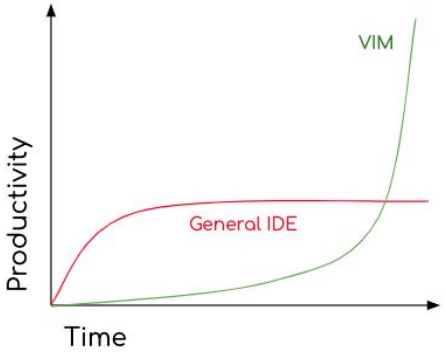
- `gedit`
  - grafikus alapú, szintaxis kiemeléssel
  - C-ben és Python-ban írták
  - GNOME Core Applications része
- `nano`
  - GNU Project része
  - terminál alapú, szintaxis kiemeléssel
  - C-ben írták

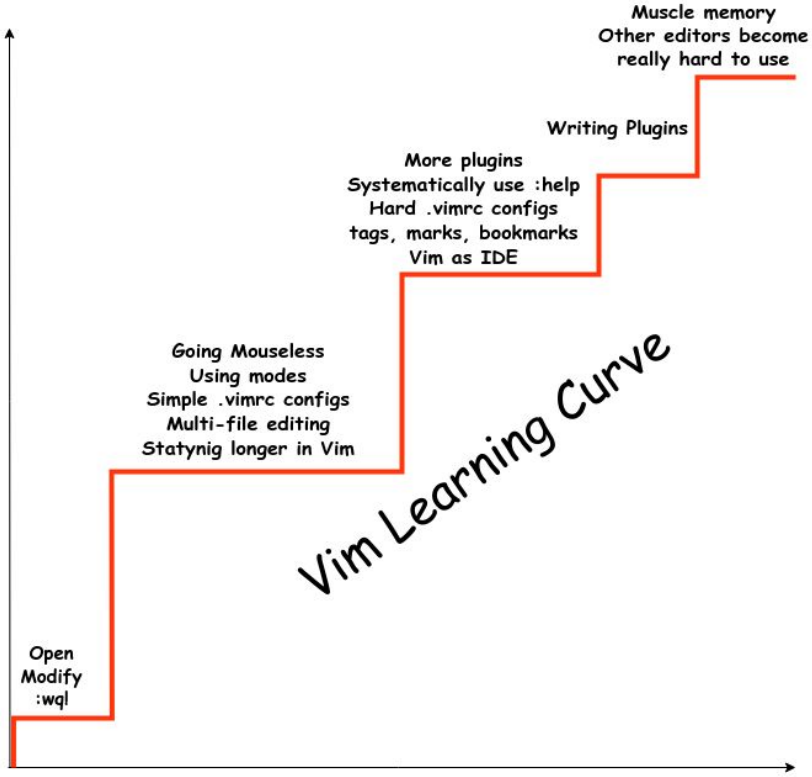
Vim - A szövegszerkesztő

- Terminál alapú, szintaxis kiemeléssel és mindenféle plugin-al
- C-ben és Vim script-ben írták
- Magas testreszabhatóság
- Mindenhol megtalálható (JetBrains, Matlab, Atom, PDF olvasók, böngészők, 9GAG stb.)
- Filozófia
  - kódolás közben a fájlok közti navigálás, inspekció és szerkesztés ne vigyen el sok időt
  - előnyös tanulási görbe
  - különböző módok a különböző fázisokhoz
    - normal mód - navigálás és inspekció
    - insert és replace módok - szerkesztés
    - visual és select módok - kijelölés
    - command-line mód - belső és külső parancsok futtatása

Vim - tanulás

- `vimtutor` (25-30 perc)
- Rendszeres használat minden feladatra





Shell

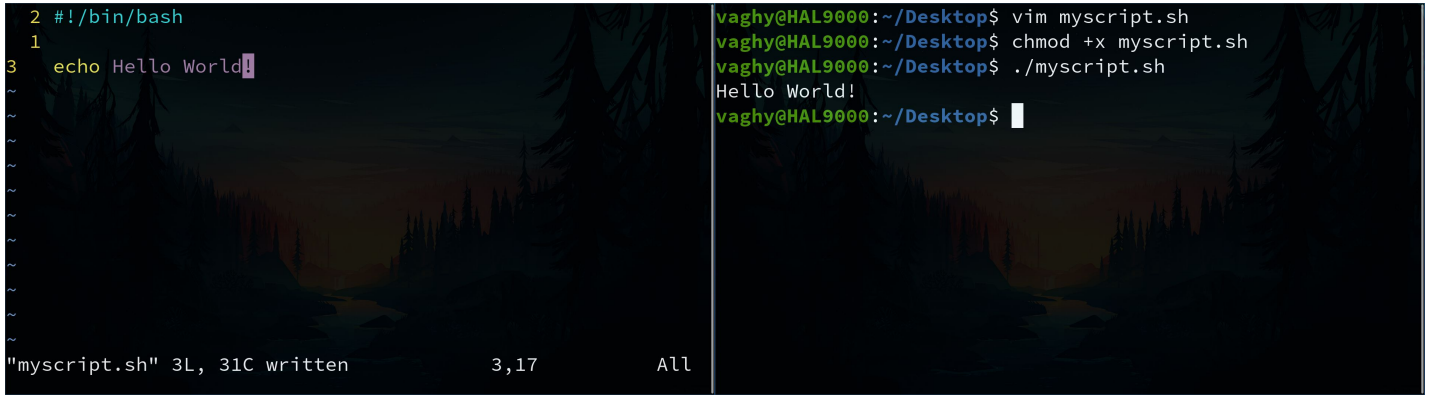
- `sh` - Bourne Shell
  - 1979-ben írta Stephen Bourne Unix-hoz
- `bash` - Bourne Again Shell
  - Bourne Shell továbbfejlesztett verziója, 1989-ből
  - alapértelmezett a legtöbb Linux disztribúcióban
- `csh` - C Shell
  - C-szerű nyelvi elemeket tartalmaz
- `zsh` - Z shell
  - alapértelmezett macOS Catalina-n
- `fish` - Friendly Interactive Shell
  - autocomplete
  - könnyű konfigurálni és script-elni

Az első script

- Nyissunk meg egy szövegszerkesztőt és írjuk bele a kedvenc parancsunkat
- `bash myscript.sh`

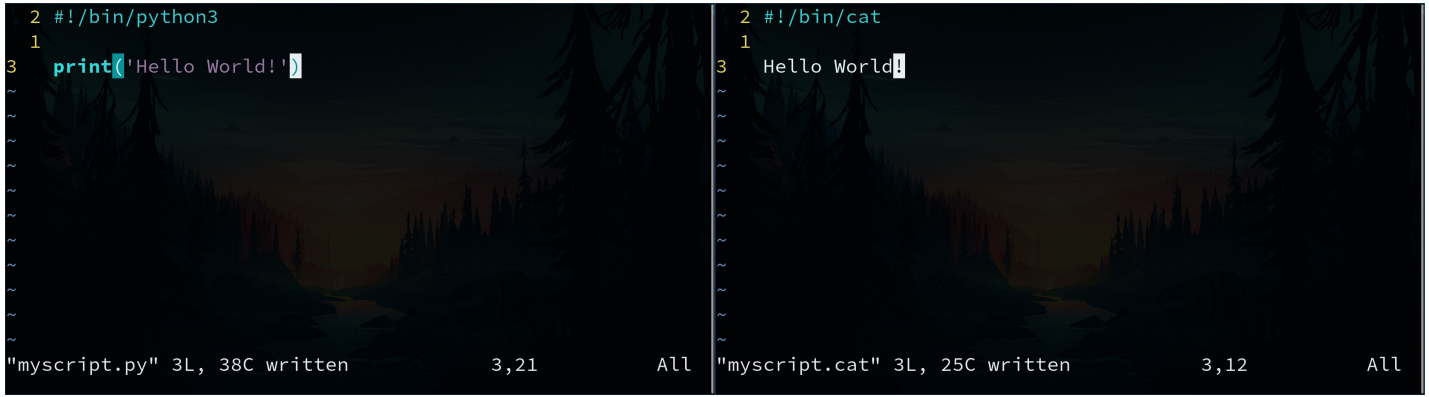
Script futtatása

- Az imént közvetlenül meghívtuk a shellt a futtatáshoz
- Remek lenne, ha az operációs rendszer tudná, hogy milyen programmal futtasson



Shebang (#!)

- Az első sor tartalmát a shell interpreter direktívaként parse-olja
- Szükséges a használatához, hogy futtatható legyen a fájl



Környezeti változók

- Gyakorlatilag globális változók, melyet a különböző programok felhasználnak és akár módosíthatnak
- `echo $SHELL`
- `echo $HOME`
- `echo $PWD`
- `echo $PATH`
  - a shell itt keresi a futtatni kívánt programot (futtatható vagy bináris)
  - `which -a command` parancs megmutatja az összes helyet a PATH-ban, ahol megtalálható

Fájlkiterjesztések

- Mi legyen a script-jeim kiterjesztése?
- A fájlkiterjesztés egy Windows-os dolog, a valóságban kétféle fájl létezik: szöveges és bináris
  - ami szöveges, azt lehet értelmesen szövegszerkesztőben szerkeszteni
  - ami bináris, ahhoz speciális szerkesztő kell (például .docx és Microsoft Word)
  - ami futtatható (x kapcsoló), az futtatható, ha benne van a PATH-ban, vagy az aktuális mappában
- Tehát értelmes keretek között mindegy, hogy mi a script kiterjesztése
  - természetesen célszerű felismerhető kiterjesztést használni

Változók

- Untyped változók, dinamikusan változhat a típus (string vagy egész szám)
- Létrehozás
  - `bar=5`
  - `let foo="Hello World!"`
  - `declare -r var1` - írásvédett változó
  - `declare -i var2` - egész változó
- Elérés
  - `let bar=$bar+1`
  - `echo $foo` - kitörli a tabulátorokat és az új sorokat
  - `echo "$foo"` - megőrzi a formázást
  - `echo '$foo'` - string literál
- Törlés
  - `unset var`

Command substitution

- `command1 $(command2)`
  - `command2` lefuttatása egy subshell-ben
  - eredmény behelyettesítése
  - `command1` lefuttatása az új argumentummal
- `foo=$(command)`
  - `command` lefuttatása
  - eredmény értékadása

Egész aritmetika

- `let expression`
  - `+, -, /, *, %, ++, --, **, <<, >>, &, ^, |, &&, ||`
  - `=, +=, -=, /=, *=, <<=, >>=, &=, ^=, |=`
- `echo $(expr operand1 operator operand2)`
  - `+, -, /, %, \*`
- `echo $((expression))` - arithmetic expansion
  - `+, -, /, *, %, ++, --, **, <<, >>, &, ^, |, &&, ||`
  - `=, +=, -=, /=, *=, <<=, >>=, &=, ^=, |=`
  - `$` nélkül visszatérési érték nélkül hajtja végre a műveletet
    - például változó inkrementálása `((var++))`

String aritmetika I.

- összefűzés
  - `echo $string1$string2`
- értékadás
  - `string2=$string1`
  - `string2+=$string1`
- string hossza
  - `echo ${#string}`
  - `echo $(expr length $string)`

String aritmetika II.

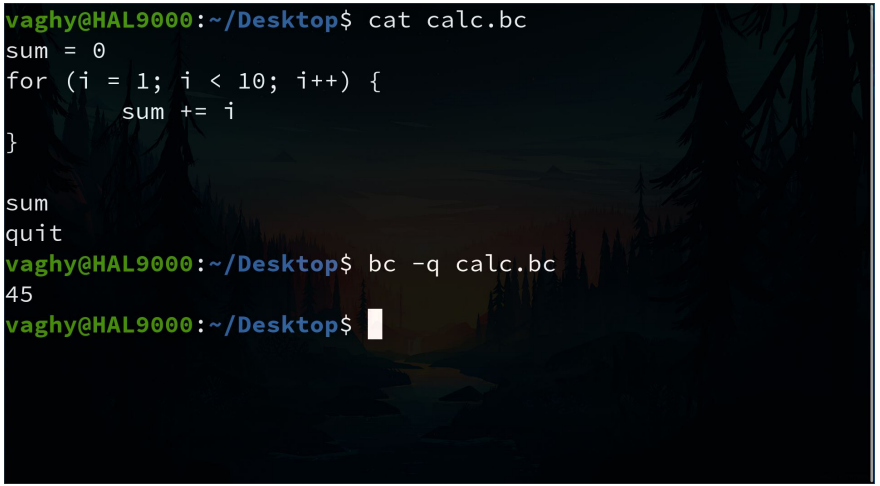
- substring
  - `echo ${string:pos:len}`
  - `echo $(expr substr $string pos len)`
- regex
  - `echo $(expr match $string $regex)` - visszaadja a találat utolsó indexét
  - `echo $(expr match $string \($regex\))` - visszaadja a találatot
- replace
  - `echo ${string/old/new}` - első találat helyettesítése
  - `echo ${string//old/new}` - összes találat helyettesítése

String aritmetika III.

- removal
  - `echo ${string#regex}` - kitörli a legrövidebb találatot
  - `echo ${string##regex}` - kitörli a leghosszabb találatot
  - `echo ${string%regex}` - kitörli a legrövidebb találatot hátulról
  - `echo ${string%%regex}` - kitörli a leghosszabb találatot hátulról

Float aritmetika

- `bc` - basic (best) calculator
  - `echo expression | bc`
  - lehet fájlban is számolni vele
    - változók
    - vezérlési szerkezetek
    - saját és beépített függvények
  - `man bc`



G02F01

- Egészítsük ki az előző `bc` script-t interpreter direktívával!
- Írjuk át a script-t úgy, hogy argumentumban fogadja a szummázás felső határát!
  - `bar=read()`
- Figyeljük meg, hogy az argumentumot lehet pipe-olni is!
- Próbáljuk ki elnevezett pipe-al is!

Adatszerkezetek

- Tömb
  - `bar=(1 2 3 4)`
  - `declare -a bar=(1 2 3 4)`
  - `bar+=(5)`
  - `echo ${bar[*]}`
  - `echo ${#bar[*]}`
- Asszociatív tömb
  - `foo=([key1]=val1 [key2]=val2)`
  - `declare -A foo=([key1]=val1 [key2]=val2)`
  - `foo[key3]=val3`
  - `echo ${foo[*]}`
  - `echo ${!foo[*]}`
  - `echo ${#foo[*]}`

Következő gyakorlat

- Archívumok, tömörítések
- Fájlkezelés
- Vezérlési szerkezetek
- Argumentumok kezelése
- Függvények

Érdekességek, olvasnivalók

- 7 Habits For Effective Text Editing 2.0
- How do I exit the Vim editor?
- Luke Smith: vimtutor Let's Play
- MIT Missing Semester: vim

Operációs rendszerek alapjai

3. gyakorlat

Mai gyakorlat

- Vezérlési szerkezetek
- Argumentumok kezelése
- Függvények

Pattern matching a terminálban

- Gyakran van arra szükség, hogy regex-t használjunk
  - adott kiterjesztésű fájlok törlése, listázása, tömörítése, átnevezése stb
  - pattern listás műveletekhez az `extglob` shell beállítás kell (`shopt -s extglob`)

`*`	bármilyen karakterből bármennyi
`?`	bármilyen karakterből pontosan egy darab
`[characters]`	felsorolt karakterek közül pontosan egy darab
`[[:class:]]`	megadott osztályból pontosan egy darab (`alnum`, `alpha`, `digit`, `lower`, `upper`)
`pattern-list`	`pattern1|pattern2|...`

`?(pattern-list)`	nulla vagy egy találat
`*(pattern-list)`	bármennyi találat
`+(pattern-list)`	legalább egy találat
`@(pattern-list)`	negálás
`!(pattern-list)`	negálás

`test` parancs - I.

- Visszatérési érték a `$?` változóban (`0`: true, `1`: false)
- `test flag file`
  - `-a`, `-e` kapcsoló: fájl létezik-e
  - `-f`, `-d`, `-h`, `-p`, `-S`, `-b`, `-c` kapcsoló: fájl létezik és sima fájl, mappa, szimbolikus link, pipe, socket, blokk eszköz, karakter eszköz
  - `-r`, `-w`, `-x` kapcsoló: fájl létezik és olvasható, írható, futtatható
- `test string1 operator string2`
  - `==` (`=`), `!=`, `<`, `>`
- `test operand1 operator operand2`
  - `-eq`, `-ne`, `-lt`, `-le`, `-gt`, `-ge` operátor: egyenlő, nem egyenlő, kisebb, kisebb egyenlő, nagyobb, nagyobb egyenlő
- `!`, `-a`, `-o` kapcsoló: kifejezések negálás, konjunkciója, diszjunkciója

`test` parancs - II.

- Használható terminálban, illetve elágazásokban
- Helyettesíthető a `[]` blokkal
  - `test flag file` helyett `[ flag file ]`
- Bash-ben
  - `[[]]` blokk
    - `test flag file` helyett `[[ flag file ]]`
    - `[[ string =~ regex ]]`
    - `-a` és `-o` helyett `&&` és `||`
  - `(())` blokk
    - egész számokra kapcsolók helyett szokásos operátorok
    - `-eq` helyett `==`
    - `-a` és `-o` helyett `&&` és `||`

Vezérlési szerkezetek - `if`

```
if <condition>; then
     <commands>
elif <condition>; then
     <commands>
else
     <commands>
fi
```

G03F01

- Olvass be két string-t!
- Regex segítségével döntsd el, hogy egész számok-e!
  - nemnegatív egész szám: `$string =~ ^[0-9]+$`
  - egész szám: `$string =~ ^(-)?[0-9]+$`
- Ha egész számok, akkor írd ki az összegüket!
- Ha nem egész számok, akkor jelezd a felhasználókat, hogy egészeket kell megadni!

Vezérlési szerkezetek - `case`

```
case <variable> in
     <pattern>)
          <commands>
          ;;
     *)
          <default commands>
          ;;
esac
```

G03F02

- Olvass be egy string-t!
- Ha egész számot adott meg a felhasználó, akkor írd ki a négyzetét!
  - regex `if`-hez: `^-?[0-9]+$`
  - pattern `case`-hez: `?(-)+([[:digit:]])`
- Ha nem egész valós szám, akkor írd ki az egészrészét!
  - regex `if`-hez: `^-?[0-9]+\.?[0-9]+$`
  - pattern `case`-hez: `?(-)+([[:digit:]])[.]+([[:digit:]])`
  - egészrész utolsó indexe (.-al együtt!) megkeresése: `expr match $number '.*[.]'`
  - ne felejtsd el, hogy például `[5.6]=5`, de `[-5.6]=6`!
- Ha nem szám, akkor jelezd a felhasználókat, hogy számot kell megadni!
- Ne felejtsd el beállítani az extended globbing-ot a `shopt -s extglob`paranccsal!

Ciklusok - `while`

```
while <condition>; do
    <commands>
done
```

- Amíg igaz a feltétel, addig végrehajtja a parancsokat
- Van végtelen ciklus
  - `while true; do`
  - `while :; do`
- `break`, `continue` szokásosan

Ciklusok - `until`

```
until <condition>; do
    <commands>
done
```

- Amíg hamis a feltétel, addig végrehajtja a parancsokat
- Van végtelen ciklus
  - `until false; do`
- `break`, `continue` szokásosan

G03F03

- Olvass be egy string-t!
- Ha nemnegatív egész számot adott meg a felhasználó, akkor írd ki az osztóit!
- Ha nem nemnegatív egész számot adott meg a felhasználó, akkor ezt jelezd, majd lépjen ki a script az `exit` paranccsal!
- Próbáld a ciklus feltételét megadni [[]] blokkal és (()) blokkal is!
- Próbáld ki while és until ciklussal is!

Ciklusok - `for`

```
for <variable> in <words>; do
     <commands>
done

for (( expr1; expr2; expr3 )); do
     <commands>
done
```

- Utóbbi ciklusban `expr1` inicializál és a ciklus addig fut, amíg `expr2` nem nulla, ekkor `expr3` kiértékelődik és `<commands>` parancsok futtatásra kerülnek (hiányzó kifejezés értéke 1)
- `break`, `continue` szokásosan

Brace expansion

- String(tömb) generálására alkalmas, `for` ciklus ezt fel tudja dolgozni
- `{string1,string2,...}`
  - megadott string-k kifejtése
  - tud konkatenálni, illetve a disztributív szabály is érvényes
    - `{a,b}{c,d}` eredménye `ac ad bc bd`
- `{x..y[..incr]}`
  - `x` és `y` egész szám, vagy karakter
  - `incr` egész szám
  - `x`-től `y`-ig generál incr ugrással
    - `{0..5}` eredménye `0 1 2 3 4 5`
    - `{a..z..5}` eredménye `a f k p u z`

G03F04

- Olvass be egy string-t!
- Ha nemnegatív egész számot adott meg a felhasználó, akkor írd ki az osztóit!
- Ha nem nemnegatív egész számot adott meg a felhasználó, akkor ezt jelezd, majd lépjen ki a script az `exit` paranccsal!
- Próbáld ki hagyományos és C-szerű ciklussal is!

Fájlkezelés I.

- Fájl nevét a mappa neve nélkül a `basename` paranccsal kaphatjuk meg
  - ebből tudjuk a kiterjesztést megtudni

```
name=$(basename file)
ext=${name##*.}
name_no_ext=${name%.*}
```

- `read` paranccsal olvashatunk standard inputról, fájl deskriptorból és fájlból
  - `-p` kapcsoló: üzenet az olvasáshoz
  - `-t` kapcsoló: timeout beállítása
  - `-s` kapcsoló: nem látszik a terminálon gépelés közben (jelszavakhoz)
  - `-r` kapcsoló: backslash-t nem escape karakterként értelmezi
  - `-d` kapcsoló: elválasztó beállítása (alapértelmezetten új sor)

Fájlkezelés II.

- Fájl beolvasása soronként

```
while read -r line; do
    echo "$line"
done < filename
```

- Fájl beolvasása tetszőleges karakterenként (például .csv estén vesszőnként)
  - `-d` kapcsoló addig olvas, amíg van elválasztó
  - `field1,field2` struktúra esetén csak az első mezőt olvassa be
  - IFS-t (Internal Field Separator) kell használni

```
while IFS=, read -r f1 f2; do
    echo $f1 $f2
done < filename
```

Fájlkezelés III.

- .csv fájl soronkénti beolvasása tömbbe

```
while IFS=, read -a array; do
    echo ${array[*]}
done < filename
```

- Fájl egészének beolvasása tömbbe (minden elem egy egész sor)

```
readarray array < filename
```

Argumentumok

- Meghívott scriptnek adhatunk át paramétereket
  - `int main(int argc, char* argv[])` megfelelője
  - `$0` - script neve, ahogy meghívtuk (például `./myscript.sh`)
  - `$#` - argumentumok száma
  - `$n` - n-edik argumentum
  - `$@` - argumentumok egy tömbben
- Indirect parameter expansion
  - `${!var}` esetén a var változó értékével jelölt változót érhetjük el
  - például ha `var` értéke 3, akkor a harmadik argumentumot kapjuk vissza

G03F05

- Írd ki tömbként a script argumentumait!
- Módosítsd a scriptet, hogy `for` ciklust használjon!
- Próbáld ki hagyományos és C-szerű ciklussal is!
    - utóbbihoz szükség lesz indirect parameter expansion-re

Kapcsoló kezelés

- Flag-eket lehetne argumentumként is feldolgozni, de elég kényelmetlen lenne
- `getopts` parancs segítségével jól kezelhetőek a kapcsolók és az argumentumok
- `getopts <flag_structure> flag`
  - beolvas egy kapcsolót a flag változóba a megadott struktúra szerint
    - egyszerűen fel kell sorolni a kapcsolókat
    - `:` karakter jelzi, hogy az adott kapcsoló argumentumot vár, ez elérhető az `$OPTARG` változóban
    - például `ab:` azt jelenti, hogy az `a` kapcsoló nem vár argumentumot, a `b` igen
  - kiírja, ha nem létező argumentumot próbál alkalmazni a felhasználó
  - kiírja, ha nem adott argumentumot adott kapcsolónak
  - `while` ciklusban érdemes használni, `case` elágazással pedig feldolgozni

G03F06

- Írj egy scriptet, ami tetszőleges számú paramétert fogad a következő struktúrával!
  - `-a` kapcsoló: a kapcsoló után megadott parancsra apropos lefuttatása
  - `-m` kapcsoló: a kapcsoló után megadott parancsra man lefuttatása
  - `-w` kapcsoló: a kapcsoló után megadott parancsra whatis lefuttatása
  - `-v` kapcsoló: apropos, man és whatis parancsok verziójának kiírása

Függvények

```
<function_name> () {
    <commands>
}

function <function_name> {
    <commands>
}
```

- A két szintaxis között nincs különbség
- Argumentumok nincsenek jelölve a függvény szignatúrában
- Argumentumok átadása és elérése script-hez hasonlóan
  - `function_name arg1 arg2 …`
  - `$0` - függvény neve
  - `$#` - argumentumok száma
  - `$n` - n-edik argumentum
  - `$@` - argumentumok egy tömbben

Visszatérési értékek, láthatóságok

- Nincs hagyományos visszatérési érték
  - `return` kulcsszóval nemnegatív egész számot beállíthatunk visszatérési status-ként
    - 0-255 között
  - ez a `$?` változóban érhető el
  - ez függvények és script-ek esetén is működik
- Szokás egyszerűen `echo`-val visszaadni az eredményt
- Alapértelmezetten minden változó globális
  - ezzel könnyen lehet visszatérési értéket szimulálni, ugyanis függvényben létrehozott változó elérhető a függvény lefutása után
  - `local` kulcsszóval létrehozhatunk lokális változókat
- Függvények elfedik a parancsokat
  - például ha definiálok egy `ls` nevű függvényt, akkor az `ls` parancs `command ls` néven érhető el

G03F07

- Olvass be egy string-t!
- Ha nemnegatív egész számot adott meg a felhasználó, akkor írd ki a faktoriálisát!
- Próbáld ki az eredmény átadását globális és lokális változóval is!
  - figyeld meg a `return` viselkedését nagy számok esetén
- Próbáld ki iteratívan és rekurzívan is!
  - rekurzióval interpretált nyelv lévén nem túl hatékony, nem ajánlott a használata (de jó gyakorlás)

Következő gyakorlat

- Archívumkezelés, tömörítések
- Alapvető programnyelvek fordítása terminálból
  - C, C++
    - röviden make és cmake is
  - Python
  - LaTeX

Érdekességek, olvasnivalók

- forkbomb
- Bash Reference Manual
- Luke Smith: Bash script without if statements

Operációs rendszerek alapjai

4. gyakorlat

Mai gyakorlat

- Archívumkezelés, tömörítés
- Alapvető programnyelvek fordítása terminálból
  - C, C++
    - röviden make és cmake is
  - Python
  - LaTeX

Archívumok

- Egy vagy több fájl vagy mappa és metaadatok tárolására alkalmas
- Önmagában nem tömörítenek, de hatékonyabb a tárolás
  - 1 byte-os fájl elhasznál (legalább) egy IO blokkot (jellemzően 4 kB)
  - 1000 ilyen fájl elhasznál 4 MB-t
  - archívumot létrehozva szekvenciálisan tárolja, így elég 1 kB (nagyjából, plusz metaadatok)
  - természetesen ez nagyobb fájloknál nem számottevő, de kényelmes egy fájlt használni sok helyett
- Legfontosabb archívum fájltípusok
  - zip - általában tömörít is a deflate algoritmussal (Lempel-Ziv egy fajtája és Huffman kódolás)
  - tar

Tar - tape archive

- Az elkészült archívumot gyakran tarball-nak hívják
- A tarball-t tömöríti sokféle választható algoritmussal
  - ezek az algoritmusok önmagukban is léteznek, a tar parancs implementálja ezeket az algoritmusokat
- Választható algoritmusok:
  - gzip - .gz
    - LZ77 (g a GNU-ra utal, ugyanis a zip nem szabad szoftver)
  - bzip2 - .bz2
    - Burrows-Wheeler transzformáció után Huffman kódolás
  - xz - .xz
    - LZMA (Lempel-Ziv-Markov chain Algorithm)
  - Zstandard - .zst
    - hardcore algoritmus, 2016-ban fejlesztette Yann Collet (Facebook)
    - gyorsabb mindennél és jellemzően a tömörítés is hatékonyabb

Tömörítés terminálban - zip

- `zip archive.zip files`
  - `-r` kapcsoló: rekurzívan tömörít mappát
  - `-0` ... `-9` kapcsoló: tömörítés hatékonyságát állítja be (0 esetén csak archívum fájlt készít)
  - -@ kapcsoló: standard inputról fogadja a fájlneveket (tehát lehet pipe-olni is)
- `unzip archive.zip`
  - `-l` kapcsoló: archívum listázása
  - `-f` kapcsoló: freshen, tehát csak azokat a fájlokat csomagolja ki, amiből már létezik egy régebbi verzió
  - `-u` kapcsoló: update, olyan mint a -f, csak nem eddig létezett fájlokat is kicsomagol

Tömörítés terminálban - tar

- `tar` paranccsal történik minden
  - `-c` kapcsoló: tömörítés
  - `-x` kapcsoló: kitömörítés
  - `-z` kapcsoló: gzip
  - `-j` kapcsoló: bzip2
  - `-J` kapcsoló: xz
  - `--zstd` kapcsoló: Zstandard
  - `-v` kapcsoló: verbose mód, tehát sok információt kiír
  - `-f` kapcsoló: fájlok megadása
- Gyakorló feladatok megoldásai gyakorlatonként .tar.gz tömörítéssel vannak feltöltve
  - `tar -xzvf Gxx.tar.gz`

C, C++ terminálban

- `gcc` - .c és .cpp fájlokat fordít, C és C++ kódként kezelve őket
- `g++` - .c és .cpp fájlokat fordít, C++ kódként kezelve őket
- Alapvető használat
  - `g++ files -o binary`
- Kapcsolók
  - `-std=<standard>`, például `-std=c++11`
  - `-Werror`: minden warning error legyen
  - `-Wall`: (majdnem) minden warning bekapcsolása
  - `-Wextra`: maradék warning bekapcsolása
  - `-pedantic`: csak ISO C és ISO C++ kódok fordítása

Feladat

- Készíts egy Hello World! programot C++-ban!
- Fordítsd le `g++`-al!
  - `g++ main.cpp -o hello`

Feladat

- Készíts egy Hello World! függvényt külön fájlban (.hpp és .cpp)!
- Készíts egy main függvényt, ahol meghívod a függvényt!
- Fordítsd le `g++`-al!
  - `g++ main.cpp func.cpp -o hello`
  - `g++ *cpp -o hello`

Object files

- Gépi kóddá fordított forráskódot tartalmaz
- `-c` kapcsolóval a .cpp fájlokból .o fájlokat gyárthatunk
- Lehet .o fájlokat linkelni a bináris elkészítéséhez
  - akár kombinálhatjuk is a .o és .cpp fájlokat
  - például egy garantáltan jól megírt kódbázist érdemes előre lefordítani, így meggyorsítva a későbbi fordításokat
- Object file-ok és header-k lehetnek más mappában is
  - `-Idir` kapcsolóval hozzáadhatjuk a dir mappát az include path-hoz

Feladat

- Készíts object file-t a Hello World! függvényből!
  - `g++ -c func.cpp`
- Linkeld a .o és .cpp fájlt a bináris elkészítéséhez!
  - `g++ main.cpp func.o -o hello`
- Próbált ki más mappából is!
  - például csinálj egy include mappát, ahol csak header-k vannak
  - `g++ main.cpp func.o -o hello -Iinclude`

Könyvtárak

- Static library
  - .a fájl Linux-on és Mac-n, .lib fájl Windows-on
  - a használt rutinok gépi kódja bekerül a binárisba
  - ha változtatsz a library-n, akkor újra kell fordítani a library-t és az azt használó programokat is
- Shared library
  - .so fájl Linux-on és Mac-n, .dll fájl Windows-on
  - a használt rutinok információi bekerülnek a binárisba, de maga a kód nem
  - futás időben az operációs rendszer betölti a kódot (dynamic linking)
  - kisebb tárhelyet igényel, hiszen a shared library-t több program is használhatja
  - ha változtatsz a library-n, akkor csak a library-t kell újra fordítani

Könyvtárak linkelése

- `-lexample` kapcsolóval libexample.a vagy libexample.so fájlt linkeljük
- Könyvtárak lehetnek más mappában is
  - `-Ldir` kapcsolóval hozzáadhatjuk a dir mappát a linker path-hoz
- .a létrehozása
  - `ar` paranccsal
  - `d`, `p`, `r` opciókkal törölhetünk, listázhatunk, beilleszthetünk archívumba
  - `ar r libfunca.a func.o`
- .so létrehozása
  - `g++ -fpic func.cpp -shared -o libfuncso.so`
  - lehet .o fájlból is, ehhez eleve Position Independent Code kapcsolóval kellett fordítani
    - `g++ -c -fpic func.cpp`
  - elérhetőséghez hozzá kell adni a helyét az $LD_LIBRARY_PATH változóhoz

Feladat

- Hozz létre static library-t és shared library-t az object file-ból!
  - `g++ -c -fpic func.cpp`
  - `ar r libfunca.a func.o`
  - `g++ -fpic func.o -shared -o libfuncso.so`
- Próbáld fordítani és futtatni a binárist static és shared library-vel is!
  - `g++ main.cpp -o hello -Iinclude -L. -lfunca`
  - `g++ main.cpp -o hello -Iinclude -L. -lfuncso`

make I.

- Láthatjuk, hogy nem egyszerű munka a header-k behúzása és a library-k linkelése
  - gyorsan eszkalálódik a végrehajtandó `g++` parancs mérete
- make segítségével könnyen lehet akár több target-re build-elni
  - például .o fájlok és bináris készítése is
- Makefile nevű fájlban vannak leírva a szabályok
  - alapvető struktúra

    ```
    target: pre-req-1 pre-req2 …
          command
    ```
  - `clean` szabályt is megadhatunk a binárisok törlésére
  - változókat is használhatunk
  - `make install` - PATH-be teszi a binárist

make II.

```
hello: func.cpp main.cpp
     g++ -o hello func.cpp main.cpp -Iinclude
```

make III.

```
all: hello

hello: func.o main.cpp
	g++ -o hello func.o main.cpp -Iinclude

func.o: func.cpp
	g++ -c func.cpp

clean:
	rm func.o hello
```

make IV.

```
CC=g++
CFLAGS=-Iinclude

all: hello

hello: func.o main.cpp
	$(CC) -o hello func.o main.cpp $(CFLAGS)

func.o: func.cpp
	$(CC) -c func.cpp
```

make V.

```
CC=g++
CFLAGS=-Iinclude -Llib -lfunc

all: hello

hello: main.cpp
	$(CC) -o $@ $^ $(CFLAGS)
```

cmake I.

- Láthatjuk, hogy nem egyszerű munka a make használata
  - gyorsan eszkalálódik a végrehajtandó `Makefile` mérete
- cmake segítségével könnyen lehet Makefile-t generálni
- CMakeLists.txt nevű fájlban vannak leírva a szabályok
- Szokás létrehozni egy build mappát
  - ekkor a fordítás menete a következő
    - `cd build`
    - `cmake ..` - cmake elkészíti a Makefile-t
    - `make`

cmake II.

```
project(Hello)
add_executable(hello main.cpp func.cpp)
target_include_directories(hello PUBLIC include)
```

cmake III.

```
project(Hello)
add_executable(hello main.cpp)
target_include_directories(hello PUBLIC include)
target_link_directories(hello PUBLIC lib)
target_link_libraries(hello func)
```

Feladat

- Töltsd le a bc forráskódját!
- Csomagold ki!
- Hívd meg a `configure` script-t!
  - beállít rendszerfüggő változókat és generál Makefile-t
- Futtasd le a `make` parancsot!

Python terminálból

- `python`, illetve `python3` paranccsal futtatható
  - fájlkiterjesztés nem számít
  - ha nem adunk meg fájlt, akkor elindul interaktív módban
    - ekkor is figyelni kell a megfelelő indentálásra
    - ez gyors teszteléshez nagyon hasznos
  - használható interpreter direktívaként futtatható fájlban

LaTeX terminálból

- Donald Knuth írta 1978-ban
  - 2014 óta nincs frissítés, aktuális verziószám: 3.14159265
  - verziószám változáskor π újabb számjegyét hozzáírják
- Több fordító közül lehet választani
  - MiKTeX
  - **TexLive** - legteljesebb, legnagyobb projekt, legkönnyebb használni
  - MacTex
- `pdflatex filename`
  - `-draftmode` kapcsolóval nem használ képeket és nem készít kimenetet
  - tartalomjegyzék frissítéséhez kétszer kell futtatni (elsőre elkészül a .toc fájl)
  - irodalomjegyzék készítése: `pdflatex`, `bibtex` futtatása a .aux fájlra, újra `pdflatex`

Markup nyelvek

- Markdown - .md fájlok
  - gyakran használt, például git-n beágyazott README.md
  - Pandoc nevű programmal ajánlott fordítani (tud LaTeX-t is egyébként)
- groff - GNU troff
  - (t)roff Unix rendszerre írt szövegformázó program
  - különböző makrók
    - man - manual oldalak készítéséhez
    - me - cikkek írásához
    - ms - könyvek, jelentések
  - sokan preferálják a LaTeX-el szemben, mert sokkal egyszerűbb és gyorsabb

groff man

- groff_man(7)
- `groff -man filename`
  - standard outputra ír, alapvetően PostScript formátumban (.ps fájl)
    - ezt lehet fájlba irányítani, vagy profibb olvasóba pipe-olni (például zathura)
  - `-Tdev` paranccsal más formátumban is, például `-Tpdf` és `-Thtml`
    - ehhez általában fel kell telepíteni a `groff` csomagot
      - gitről letölthető a forráskód, `bc`-hez hasonlóan kell eljárni
- Megfelelő formátumú fájlt megnyithatunk a `man -l filename` paranccsal
  - szokás odaírni a manual jellegét is, például `ls.1`
- `gzip`-el való tömörítés után `man filename`
  - .gz fájlt `/usr/share/man/manx` mappába helyezve nem kell elérési útvonalat sem megadni
  - .gz fájl tartalmának kiírása `zcat` paranccsal

HF01 - 10 pont

- Másold a könyvtáradba /home/vagmi/Desktop mappából HF01.tar.gz fájlt (tempus szerveren)
  - Unix vagy Linux rendszerre töltsd le a HF01.tar.gz fájlt a honlapról
- Csomagold ki!
- HF01.1.gz manual oldalnak megfelelően írd meg a `HF01.sh` script-t!
- Tesztelhetsz a `test.sh` script-el, vagy manuálisan is!
- `bc` nincs egyik szerveren se és még cortex-n sincsenek a compile-hoz megfelelő könyvtárak
  - a fenti mappában van egy `bc` nevű python script, ami fogad `a+b` alakú kifejezéseket
  - a kifejezés értékét kiírja (command substitution-el elmenthető)
- Figyelj a megfelelő adatszerkezetek, vezérlési szerkezetek és függvények használatára!
- Leadás: töltsd fel a /home/vagmi/Desktop mappába NEPTUN.sh néven!
  - határidő: 2020.10.14. 23.59.59

Következő gyakorlat

- Szövegkezelés

Érdekességek, olvasnivalók

- Donald Knuth: The Art of Computer Programming
- Bisqwuit: How to Create a Compiler
- suckless: software that sucks less
- Scipiades Ármin: A Code::Blocks és az Eclipse káráról - avagy miért ne használjunk IDE-ket a programozásoktatásban

Operációs rendszerek alapjai

5. gyakorlat

Mai gyakorlat

- Szövegkezelés
- Feladatokhoz szükséges: G05.tar.gz

grep

- Globally search for a regular expression and print matching lines
- Több regex-t tud keresni több fájlban is, soronként
- Bemenet lehet pipe-olva
- `grep [OPTION...] PATTERNS [FILE...]`
  - `-c` kapcsolóval találatok számát írja ki
  - `-n` kapcsolóval sorok számának kiírása is
  - `-r` kapcsolóval rekurzívan keres
    - `-R` kapcsolóval szimbolikus linkeket is követ
- Példa
  - csak fájlok listázása: `ls -l | grep ^-`
  - `myfunc` függvény használatainak száma .cpp fájlokban: `grep -c myfunc *cpp`

G05F01

- Nézd meg, hogy milyen csomagok vannak feltelepítve
  - `dpkg` - Debian package
- Szűrj rá mondjuk a python szót tartalmazó csomagokra!
  - `dpkg -l | grep python`
  - `dpkg --get-selection | grep python`

G05F02

- Terminálban a nyilak használatával vissza lehet nézni a lefuttatott parancsokat
- `history` paranccsal kiírhatjuk az összeset (vagyis az utolsó n darabot)
- Szűrj rá például a `cd` parancsra!

grep

- A találat értelmezéséhez gyakran szükség van kontextusra
- Jó lenne, ha a találat körüli néhány sort is kiírná
  - például az előzmények szűrésekor
- `-A`, `-B`, `-C` kapcsolókkal találat előtt, után, előtt és után kiír n sort
  - `history | grep -C5 make`

sort

- Fájl sorait rendezi standard kimenetre
  - kimenet átirányítható
  - bemenet lehet pipe-olva
- Not-in-place rendez, önmagába irányított rendezés nem működik
- Néhány fontos kapcsoló
  - `-r`, `--reverse` fordítva rendez
  - `-R`, `--random-sort` random rendez
  - `-c` ellenőrzi a rendezést
  - `-t` elválasztó megadása
  - `-k` hanyadik oszlop szerint rendezzen

G05F03

- A points.txt fájlban a kedvenc tárgyad házi eredményeit láthatod .csv formátumban
  - NEPTUN,HF1,HF2
- A grades.txt fájlban az érdemjegyeket láthatod .csv formátumban
  - JEGY,NEPTUN
- Rendezd a points.txt fájlt Neptun kód szerint!
  - `sort points.txt`
- Rendezd a grades.txt fájlt Neptun kód szerint!
  - `sort -k 2 -t , grades.txt`

join

- Join lines of two files on a common field
- Rendezett fájlokra működik
- `join file1 file2`
- Néhány fontos kapcsoló
  - `-t` elválasztó megadása
  - `-1` első fájlban hanyadik oszlop szerint
  - `-2` második fájlban hanyadik oszlop szerint
  - `-o` kimeneti formátum megadása, például `1.1,1.2,2.1,2.2`

G05F04

- Fűzd össze a points.txt és grades.txt fájl tartalmát!
  - ne felejtsd el rendezni mindkét fájlt
- `join -2 2 -t , <(sort points.txt) <(sort -t , -k 2 grades.txt)`
- `join -22 -t, <(sort points.txt) <(sort -t, -k2 grades.txt)`
- Mentsd el az eredményt a class.txt fájlba!

cut

- Remove sections from each line of files
- Bemenet lehet pipe-olva is, kimenet standard outputra érkezik
- Fontosabb kapcsolók
  - `-b`, `-c`, `-f` kapcsolóval byte, karakter, vagy mezők kiválasztása
  - `-d` kapcsolóval elválasztó megadása (alapértelmezetten tab)
  - `--complement` kapcsolóval a komplemens kiválasztása
- Objektumok kiválasztása
  - N-edik objektum (egytől indexel): `N`
  - N-edik objektumtól: `N-`
  - N-edik objektumtól az M-edik objektumig (inkluzív): `N-M`
  - M-edik objektumig: `-M`
  - vesszővel elválasztva lehet felsorolni is

G05F05

- G05F04 feladat után class.txt fájlban megtalálhatok a pontszámok és a jegyek is
- Jelenítsd meg csak a Neptun kódokat!
  - `cut -d, -f1 class.txt`
  - `cut -c-6 class.txt`
  - `cut -b-6 class.txt`
- Jelenítsd meg csak a pontokat és a jegyeket!
  - `cut -d, -f2- class.txt`
  - `cut -d, -f1 --complement class.txt`

Egyéb szöveges parancsok

- `uniq`
  - ismételt sorokat keres, szűr
  - `uniq input.txt output.txt`
- `paste`
  - merge lines of files
  - `paste 1.txt 2.txt`
- `tr`
  - translate or delete characters
  - `tr [a-z] [A-Z] input.txt > output.txt`
  - `tr áÁéÉíÍóÓöÖőŐúÚüÜűŰ aAeEiIoOoOoOuUuUuU < input.txt`
- `shuf`
  - random permutációt ír ki (gyorsabb, mint a random sort)

comm

- Compare two sorted files line by line
- `comm input1.txt input2.txt`
- Kimenet három oszlopban
  - első oszlop: csak az első fájlban szereplő sor
  - második oszlop: csak a második fájlban szereplő sor
  - harmadik oszlop: mindkettő fájlban szereplő sor
- Fontosabb kapcsolók
  - `-1`, `-2`, `-3` kapcsolóval az első, második, harmadik oszlop mellőzése
  - `-i` kapcsolóval case-sensitivity kikapcsolható
  - `--output-delimiter` kapcsolóval állítható az oszlopok közti elválasztó

diff

- Compare files line by line
- Nem csak a fájlok közötti különbséget írja ki, azt is hogy milyen változtatás kell az egyezéshez
  - módtól függ, hogy hogyan írja ki
- Vegyük elő a múlt heti C++ példát!
  - old nevű mappában Hello World! main.cpp
  - new nevű mappában külön .hpp-ben és .cpp-ben Hello World! függvény
  - brief mód: `diff old new -q`

```
Only in new: func.cpp
Only in new: func.hpp
Files old/main.cpp and new/main.cpp differ
```

diff - normal mód (alapértelmezett)

```
vaghy@HAL9000:~/Desktop/G04$ diff old new
Only in new: func.cpp
Only in new: func.hpp
diff old/main.cpp new/main.cpp
1c1
< #include <iostream>
---
> #include "func.hpp"
4c4
<       std::cout << "Hello World!\n";
---
>       func();
```

- Csak a new mappában van a func.hpp és func.cpp fájl
- main.cpp fájlok különböznek
  - 1c1 - első fájl első sorát a második fájl első sorára kell cserélni
    - a, c, d: add, change, delete
  - < - első fájl adott sora
  - > - második fájl adott sora

diff - context mód (-c kapcsoló)

```
vaghy@HAL9000:~/Desktop/G04$ diff old new -c
Only in new: func.cpp
Only in new: func.hpp
diff -c old/main.cpp new/main.cpp
*** old/main.cpp        2020-10-05 15:18:01.312440761 +0200
--- new/main.cpp        2020-10-05 15:25:48.640721726 +0200
***************
*** 1,6 ****
! #include <iostream>

  int main() {
!       std::cout << "Hello World!\n";
        return 0;
  }
--- 1,6 ----
! #include "func.hpp"

  int main() {
!       func();
        return 0;
  }
```

- Első fájl * jellel jelölve, második - jellel
- Mindkét fájlt az elsőtől a hatodik sorig írja ki
- ! jelzi a különböző sorokat

diff - unified mód (-u kapcsoló)

```
vaghy@HAL9000:~/Desktop/G04$ diff old new -u
Only in new: func.cpp
Only in new: func.hpp
diff -u old/main.cpp new/main.cpp
--- old/main.cpp        2020-10-05 15:18:01.312440761 +0200
+++ new/main.cpp        2020-10-05 15:25:48.640721726 +0200
@@ -1,6 +1,6 @@
-#include <iostream>
+#include "func.hpp"

 int main() {
-       std::cout << "Hello World!\n";
+       func();
        return 0;
 }
```

- Első fájl - jellel jelölve, második + jellel
- Mindkét fájlt az elsőtől a hatodik sorig írja ki
- - jelzi a törlendő sorokat, + a hozzáadandó sorokat

diff

- Néhány fontosabb kapcsoló
  - `-y` kapcsolóval eredmény mutatása egymás mellett (csak normal módban)
  - `-r` kapcsolóval rekurzív (szimbolikus linkeket is követ)
  - `-x` kapcsolóval kihagyhatunk fájlokat vagy mappákat, amik megfelelnek a megadott mintának
    - `diff old new -x *.hpp`
  - `-X` kapcsolóval fájlban megadhatjuk a kihagyandó fájlokat
- Fájlba irányíthatjuk a kimenetet

patch

- Apply a diff file to an original
- A diff parancs unified kimenetében megadott utasításokat végrehajtja
  - lehet pipe-olni, argumentumként átadni vagy átirányítani
- Néhány fontosabb kapcsoló
  - `-b` kapcsolóval backup fájlok létrehozása (.orig végződés)
  - `-d` kapcsolóval belép egy mappába
  - `--dry-run` kapcsolóval végrehajtás nélkül kiírja, hogy mit csinálna
  - `-i` kapcsolóval argumentumként megadhatjuk a patchfile-t
  - `-R` kapcsolóval visszacsinálja

G05F06

- Hasonlítsd össze az old és a new mappa tartalmát!
  - `diff old new`
- Próbáld ki több módban is!
- Hajtsd végre az előírt változásokat! Készíts backup-t is!
  - `diff old new -u | patch -d old -b`
  - `diff old new -u > project.patch`
  - `patch -d old -b -i ../project.patch`
  - `patch -d old -b < project.patch`

sed

- Stream editor for filtering and transforming text
- Bemenet lehet fájl, vagy pipe-olva
- Alapértelmezetten kiír minden sort standard outputra
  - `-n` kapcsolóval kikapcsolhatjuk
- Legalább egy parancsot meg kell adni
  - üres parancs: `sed '' input.txt`
- `-e` kapcsolóval megadhatunk több parancsot is
  - `sed -e '' -e '' input.txt`
- `-i` kapcsolóval in-place végzi el a parancsokat
  - `-i.bak` kapcsolóval in-place dolgozik és az eredetiből backup-t csinál .bak kiterjesztéssel

sed

- Specifikálhatjuk, hogy az adott parancsot mely sorokra hajtsa végre
  - sorok számozása `1`-től `$`-ig
  - `n,m`: n-edik sortól m-edik sorig
  - `n~m`: n-edik sortól minden m-edik sor (itt használható a nulla is)
  - `range!`: megadott range negálása
  - `/regex/`: reguláris kifejezésnek megfelelő sorokra
  - `/start_regex/,/stop_regex/` : első regexnek megfelelő sortól a második regexnek megfelelő sorig
  - lehet kombinálni is a számozás és a reguláris kifejezéseket
- Több parancs végrehajtása sorok kiválasztása után
  - `-e` kapcsolóval fapados, mert többször be kell írnom a szűrőt
  - `{}` között felsorolt parancsokat végrehajtja a szűrés után
    - új sor az elválasztó

sed

- `p` - print
- `d` - delete
- `a` - append
  - sor után beilleszt új sorba
  - `sed 'a\new line after each line' input.txt`
- `c` - change
  - megváltoztatja a sort
  - `sed '0~2 c\changed every second line' input.txt`
- `i` - insert
  - sor előtt beilleszt új sorba
  - `sed '/regex/ i\new line before lines that matched' input.txt`

sed

- `s` - substitute
  - `sed 's/old_regex/new_regex/' input.txt`
  - tetszőleges elválasztót használhatunk (ez akkor jó, ha a regex tartalmazza a / karaktert)
  - parancs végére írhatjuk a beállításokat
    - `'s/old_regex/new_regex/g'` összes találatot módosítja
    - `'s/old_regex/new_regex/2'` csak a második találatot módosítja
    - `'s/old_regex/new_regex/5g'` ötödik találattól kezdve összeset módosítja
- `r` - read
  - lehet megadott fájlból olvasni
  - szokásosan lehet kezelni a sorokat
- `w` - write
  - lehet megadott fájlba írni

G05F07

- A fruits.txt fájlban láthatsz egy listát, hogy adott gyümölcsökből mennyi áll rendelkezésre
- Jelezd az adott sor végén, ha elfogyott az adott gyümölcs!
  - `sed '/ 0 / s/$/ elfogyott:(/' fruits.txt`
- Próbáld ki in-place és csinálj backup-t is!
  - `sed -i.bak '/ 0 / s/$/ elfogyott:(/' fruits.txt`
- Töröld ki a sort és írd a helyére az üzenetet!
  - `sed '/ 0 / c\elfogyott:(' fruits.txt`
- Próbáld fájlból beolvasni a hiba üzenetet!
  - ```
    sed '/ 0 / {r fruits_error.txt
    d}' fruits.txt
    ```
    ```

sed

- Van lehetőségünk létrehozni egy sed script-t is
  - érdemes, ha van egy nagyon jól megírt eljárásunk
- Minden parancs új sorban
- `sed -f script.sed input.txt`
- Futtathatunk interpreter direktívával is
  - `#!/bin/sed -f`

G05F08

- Írd ki a Neptun kódokat és a jegyeket a class.txt fájlból!
- Írj egy sed script-t, ami a jegy alapján külön fájlokba teszi a hallgatókat!
  - bukó hallgatók
  - elégséges és közepes hallgatók
  - jó és kitűnő hallgatók
- Írj egy script-t, ami a megfelelő template levelek alapján leveleket ír!
  - levél neve legyen a Neptun kód
  - hallgató Neptun kódját a <NEPTUN> kulcsszó helyére kell írni
  - hallgató jegyét a <GRADE> kulcsszó helyére kell írni

Következő gyakorlat

- awk

Érdekességek, olvasnivalók

- Brian Kernighan interviews Ken Thompson
- Brian Kernighan: Where GREP Came From
- Myers' Difference Algorithm
- Nagyon jó sed tutorial

Operációs rendszerek alapjai

6. gyakorlat

Mai gyakorlat

- awk
- Feladatokhoz szükséges: G06.tar.gz

awk

- Alfred Aho, Peter Weinberger és Brian Kernighan írta
- Szöveg feldolgozás orientált nyelv
  - bemeneti stream-t az eredeti változtatása nélkül módosítja
- Használhatjuk
  - parancsként, a bemenetet pipe-olva, vagy argumentumként átadva (akár több fájl is)
  - interpreter direktívaként
    - sed-hez hasonlóan `-f` kapcsolóval megadhatjuk a kódot tartalmazó fájlt
- Interpretált nyelv
  - GNU Awk
  - New Awk
  - stb.

Programstruktúra

- awk program különböző részekre bontható
  - inicializáló rész (`BEGIN` kulcsszó)
    - egyetlen egyszer fut le, az első sor olvasása után
    - lehet többször használni (minden parancs egyszer fut el)
  - saját függvények definíciója
  - programtörzs
    - bizonyos mintázatok esetén adott parancsok lefuttatása
    - adott mintázathoz tartozó parancsok `{}`-ben, parancsok elválasztása `;`-el
    - üres mintázat minden sorra lefut
  - lezáró rész (`END` kulcsszó)
    - egyetlen egyszer fut le, az utolsó sor olvasása után
    - lehet többször használni (minden parancs egyszer fut le)

Szöveg lebontása

- Bemenet felosztva rekordokra és mezőkre
  - alapértelmezetten új sor a rekord elválasztó és whitespace a mező elválasztó, tehát egy rekord egy sornak felel meg, egy mező egy szónak
  - ez a működés átállítható
- Kiírás a `print` paranccsal
  - `$0` változó kiírja a rekordot
  - `$n` változó az n-edik mezőt írja ki
  - vesszővel elválasztva a mezők
  - precízebb kiíráshoz van `printf` is

G06F01

- Írj egy awk script-t, ami kiírja egy fájl összes sorát!
- Módosítsd a script-t, hogy kiírja egy fájl összes sorának első szavát!
- Próbáld ki a kezdes.txt fájlon!

Változók és kifejezések

- Numerikus konstansokat használhatunk hagyományos és tudományos módon
  - decimális, oktális (`0`-val kezdődik) és hexadecimális (`0x`-el kezdődik)
- String konstansokat “” között használhatunk
- Reguláris kifejezés konstansok `//` között, `@//` között strongly typed
- Változók használata szokásosan
  - előre definiálhatunk változókat, akár az inicializáló rész lefutása előtt a `-v foo=bar` szintaktikáva (csak parancssoros futtatás esetén)
- Előre definiált változók
  - `NR` az eddig feldolgozott rekordok száma
  - `NF` az adott rekord mezőinek száma
  - `RS`, `ORS`, `FS`, `OFS` a be- és kimeneti rekord- és mező szeparátor
    - alapértelmezetten rendre új sor, új sor, whitespace, whitespace

G06F02

- class.txt fájl múlt óráról tartalmazza a kedvenc tárgyad pontjait .csv formátumban
  - egy oszlop Neptun kód, két oszlop házi pontok, egy oszlop érdemjegy
- Írd egy awk script-t, ami kiírja a Neptun kódokat és a hozzájuk tartozó érdemjegyeket!
  - mező szeparátor átállítása az inicializáló blokkban
  - első és negyedik mező kiírása

Mintázatok

- Többféleképpen tudunk szűrni sorokat
  - `/regex/` esetén a megadott kifejezésnek megfelelő sorokat választja ki
  - `expression` esetén nem nulla vagy nem üres eredmény esetén a sor kiválasztása
    - kifejezések később bővebben
  - `begin_pattern,end_pattern`
- Mintázatok között értelmezhető a metszet (`&&`), unió (`||`) és negálás (`!`) művelete

G06F03

- Írj egy awk script-t, ami kiírja az ötös jegyet szerző hallgatókat a class.txt fájlból!
  - `5$` reguláris kifejezéssel szűrj

Kifejezések

- Szám és szöveg változók automatikus konverziója
  - ha valamiért nem működik jól magától, akkor expliciten kell konkatenálni számot üres string-el, vagy string-t összeadni nullával
- Számok között szokásos műveletek, értékadások és összehasonlítások
  - `+, -, *, /, %, ^, ++, --`
  - `=, +=, -=, *=, /=, %=, ^=`
  - `<, >, <=, >=, ==, !=`
- Szokásos logikai műveletek
  - `&&, ||, !`
  - ami nem nulla vagy nem üres, az `true`
- Regex-el összehasonlítás `~` és `!~` operátorral

G06F04

- Írj egy awk script-t, ami kiszámolja az átlagos érdemjegyet a class.txt fájlban!
  - inicializáló blokkban mező szeparátor beállítása
  - jegyek összeadása
  - lezáró blokkban összeg leosztása a feldolgozott rekordok számával (`NR`)

Elágazások

```
if (condition)
     commands
else if
     commands
else
     commands
```

```
condition ? true-expr : false-expr
```

```
switch (expression) {
case value:
     commands
     break
case regexp:
     commands
     break
default:
     commands
}
```

while

```
while (condition)
     commands

do
     commands
while (condition)
```

- `break` és `continue` szokásosan
- `next` befejezi az adott rekord feldolgozását
- `nextfile` befejezi az adott fájl feldolgozását
- `exit` befejezi a futást, opcionális visszatérési értéket is megadhatunk

for

```
for (initialization; condition; increment)
     commands

for (val in array)
     commands
```

- `break`, `continue`, `next`, `nextfile`, `exit` szokásosan

G06F05

- Switch segítségével számoljuk meg az ügyes (elégtelen és elégséges érdemjegy), ügyesebb (közepes vagy jó érdemjegy) és legügyesebb (jeles érdemjegy) hallgatókat a class.txt fájlban!
  - inicializáló blokkban mező szeparátor beállítása
  - hallgatók számlálása
  - lezáró blokkban kiírás

G06F06

- Írj egy awk script-t, ami kiszámolja egy adott szám faktoriálisát!
  - próbáld ki while és for ciklus segítségével is
  - `-v` kapcsolóval add a számot
    - `./G06F06.awk -v n=5`

Tárolók

- Asszociatív tömbök vannak csak
  - használható hagyományos tömbként, ha a kulcsok egész számok
  - többdimenziós tömböt is lehet gyártani kulcsok felsorolásával
  - gawk-ban tömbök tömbje is
- Tömb kulcs és érték lehet szám vagy string
- Deklaráció nélkül lehet egyből használni
- Szokásosan lehet indexelni és értékül adni
  - `array[bar]=foo`
- Lehet törölni adott elemet
  - `delete array[bar]`
- Iterálható

G06F07

- A matrix.txt fájl egy bűvös négyzetet tartalmaz .csv formátumban
  - minden sor, oszlop és a két átló összege ugyanannyi
- Írj egy awk script-t, ami ellenőrzi, hogy valóban bűvös négyzet-e!
  - inicializáló blokkban mező szeparátor beállítása
  - törzsben a mátrix 2D tömbbe mentése
  - lezáró blokkban a tulajdonság ellenőrzése

Beépített függvények

- Numerikus függvények
  - `atan2(y,x)`, `cos(x)`, `sin(x)`, `exp(x)`, `log(x)`, `sqrt(x)`, `int(x)`, `srand(x)` és `rand()`
- String függvények
  - `index(sub,str)`, `length(str)`, `sub(regex,replacement,target)`, `gsub(regex,replacement,target)`, `sprintf(format,expression,...)`, `substr(str,pos,len)`, `tolower(str)`, `toupper(str)`
- Shell elérése
  - `system(command)`
  - `print command | "/bin/bash"`
- Egyéb függvények: I/O műveletek, bitenkénti operátorok stb.

Saját függvények

- Hagyományos szintaxis

```
function bar(parameters) {
    commands
}
```

- Alapértelmezetten minden változó globális
  - lokális változót fel kell sorolni a szignatúrában, de nem kell átadni (konvenció szerint extra space-k)
    `function bar(    i)` meghívása `bar()`
- Tömbök referencia szerint, minden más érték szerint átadva
- Visszatérési értékek a `return` kulcsszóval
- Indirect function call a @ karakterrel
  - ha a `bar` változó értéke `foo`, akkor `@bar()` meghívja a `foo` függvényt

G06F08

- Írj két függvényt awk script-ben, ami megkeresi egy 2D tömb minimumát, illetve maximumát!
- Bemeneti paraméter legyen, hogy melyik függvényt hívod meg!
- Futtasd le a matrix.txt fájlon!
- Lépések
  - inicializáló blokkban mező szeparátor beállítása
  - törzsben a mátrix 2D tömbbe mentése
  - lezáró blokkban a bemenetnek megfelelő függvény meghívása indirect function call-al

Következő gyakorlat

- Terminál-internet interakció

Érdekességek, olvasnivalók

- GNU Awk User's Guide
- The AWK Programming Language book
- Ronald P. Loui: Why GAWK for AI?
- Computerphile - What's your Favourite Programming Language?
- How To Write Unmaintainable Code

Operációs rendszerek alapjai

7. gyakorlat

Mai gyakorlat

- Terminál-internet interakció

Security jelentése

- Fontos elkülöníteni a biztonság fogalmakat
- Ahogy az ssh (secure shell) esetében is láttuk, a korábbi rsh (remote shell) használatakor is lehet szabályozni a hozzáférést például jelszavakkal
    - tehát alapvetően aki nem ismer egy létező user-password párost, az nem tud belépni
    - azonban a kommunikáció nyíltan történik, tehát lehallgatható
- Secure változat esetén a bejelentkezést követően a kommunikáció is titkosított, tehát nem lehallgatható (pontosabban lehallgatható, csak értelmezhetetlen)
- Eredeti változatok gyakran nem is használhatóak, például rsh és rcp igazából egy szimbolikus link ssh-ra és scp-re

scp

- Secure copy protocol
  - rcp (remote copy) secure változata
  - ssh-t használ
  - `scp files target`
    - kapcsolók a hagyományos `cp`-hez hasonlóan, például `-r` kapcsolóval rekurzívan másol
- A protokollhoz nem tartozik RFC
- Manapság elavultnak és rugalmatlannak tartják, komoly használathoz sftp-t ajánlanak

sftp

- ftp (File Transfer Protocol) secure változata
  - másik biztonságos változata az ftps, ami a http-https pároshoz hasonlóal SSL/TLS-t is használ
- Létrehoz egy kapcsolatot, így tudunk navigálni a szerver fájlrendszerében is a szokásos parancsokkal (`ls`, `cd`) és a lokális fájlrendszerben is (`lls`, `lcd`)
- `put` és `get` parancsokkal fel- és letölthetünk fájlokat

G07F01

- Próbáld ki az sftp parancsot valamelyik szerveren!

wget

- Fájlok letöltésére alkalmas HTTP(S) és FTP(S) protokollok használatával
- Nem interaktív, kifejezetten alkalmas script-eken belüli használatra
- Rekurzív letöltésre, linkek követésére és időbélyegek figyelésére is alkalmas
- `wget [options] [URLs]`
  - URL-ben megadhatjuk a portot, illetve felhasználói adatokat is

    `http://host[:port]/directory/file` `http://user:password@host/path`

    `ftp://host[:port]/directory/file` `ftp://user:password@host/path`
- Logot standard outputra írja alapértelmezetten

wget kapcsolók

- `-b`/`--background`: háttérben indítja el, log a wget-log fájlba kerül (ha nem adunk meg mást)
- `-o`/`-a logfile`: megadott fájlba írja vagy fűzi a logot
- `-q`/`-v`/`-nv`: quite, verbose vagy a kettő közötti mód
- `-i file`: megadott fájlból olvassa a címeket
- `-t number`: próbálkozások számának megadása, alapból 20 (ha nincs súlyos hiba)
- `-c`/`--continue`: folytatja a letöltést
- `-P prefix`: mentés helyének megadása, alapértelmezetten `.`
- `-r -l5`: rekurzív letöltés bekapcsolása és mélység megadása (alapértelmezetten öt)
- `-O filename`: letöltött fájlokat konkatenálja és elmenti a megadott fájlba
- `--post-data 'key=value'`: kulcs-érték párokkal adat megadása
- `--post-file filename`

G07F02

- Töltsd le az alábbi közzétett Google táblázatot wget segítségével!
  - `wget -O filename.csv "link"`
  - (közzétenni a Fájl > Közzététel az interneten menüben lehet)

G07F03

- Töltsd ki az alábbi Google formot wget segítségével!
- A weboldal kódjában keresd meg a megfelelő mezők nevét, majd használd ezeket kulcsként a `--post-data` argumentumában!

G07F04

- DOI alapján tölts le egy cikket a sci-hub.do oldalról!
  - példa DOI: 10.1016/j.ifacol.2019.07.008
- Először próbáld ki az oldalt és döntsd el, hogyan lehetne kiszűrni a pdf fájlt!
- Az eddigi funkciókat használva igyekezz minél kevesebb sorból megoldani a problémát!
  - irányadó: one-liner

API hívások

- Egyszerűbb API hívásokat is végezhetünk wget segítségével
  - HTTP status code-oknak megfelelő cicás képek: http.cat
  - időjárás: wttr.in
    - többféle formátum van, például ez
- Akár lehet sütiket is elmenteni és betölteni
- A probléma, hogy bonyolultabb esetben JSON-ben szokás kommunikálni API-val, amit nehéz bash-ben kezelni
  - feltöltést még csak-csak össze lehet hozni, de bash-ben nem léteznek például beágyazott dict-ek
  - egyéb scriptnyelvekben (például Python, Perl, Ruby) ez egy kényelmes feladat

cURL

- wget-hez hasonló nem interaktív program
- Van mögötte egy könyvtár (libcurl), így többféle platformon és nyelven használható
- Többféle protokollt ismert
- Rugalmasabb feltöltés (nem csak HTTP POST kérések)
- Tömöríteni és kitömöríteni is tud

rsync

- Lokális és remote fájlmásolásra alkalmas, nagyon rugalmas, hatékony és robosztus
- Alapértelmezetten is csak a változott fájlokat másolja, a működésnek szinte minden aspektusa állítható
- Képes
  - megőrizni a tulajdonost, időbélyegeket stb.
  - megtartani a fogadó frissebb fájljait
  - backup-t készíteni
  - tömöríteni másolás közben
- Példa: Linux-on az egyik legjobban használható cloud szolgáltatás a Dropbox
  - telepítés után megadott mappába (alapértelmezetten ~/Dropbox) tölti le a fájlokat
  - daemon folyamatosan figyeli a változásokat, le- és feltölti az új fájlokat
  - rsync segítségével kifejezetten kényelmes

Egyéb parancsok

- `who` / `w` - bejelentkezett felhasználók listázása
  - többszöri bejelentkezést is jelez
- `finger` - alapvető információk az adott felhasználról
  - tempus-on nincs
- `write` - üzenet írása másik felhasználónak
  - `write vagmi` után megnyílik a csatorna és küldheted az üzeneteket
  - `write vagmi ttyname` paranccsal többször bejelentkezett hallgatónak adott terminájára írsz
  - lehet pipe-olni is fájlt
- `wall` - üzenet írása minden felhasználónak
  - tempus-on nincs
- `mesg` - adott terminálra való írás engedélyezése vagy blokkolása

Egyéb parancsok

- `find` - search for files in a directory hierarchy
  - rekurzívan keres adott tulajdonságnak megfelelő objektumot
  - gyakorlatilag az inode bármilyen tulajdonságra lehet szűrni
  - `-type b/c/d/p/f/l/s` kapcsolóval block / karakter / mappa / pipe / fájl / symlink / socket keresése
  - `-name regex` kapcsolóval pattern-nek megfelelő fájlokat keres
  - `-exec command {} ;`
    - a megtalált fájlokra végrehajtja az adott parancsot, ahol a fájlt a `{}` helyére helyettesíti be
    - ha parancssorból hívjuk meg, akkor `\;`-el kell lezárni
  - `-exec command {} +`
    - a megtalált fájlokat összefűzi és végrehajtja az adott parancsot, ahol a fájlt a `{}` helyére helyettesíti be
    - ha parancssorból hívjuk meg, akkor `\+`-el kell lezárni

Folyamatok

- Adott session során rengeteg folyamat fut
- Eddig főleg előtérben futtattunk, de láttuk már, hogy az & jellel futtathatunk háttérben is
- Minden folyamat rendelkezik egy egyedi azonosítóval, ez a PID
  - amikor háttérben futtatsz egy parancsok, akkor kiírja a PID-jét
- `top` / `htop` parancsokkal interaktívan listázhatjuk az összes futó parancsot
- `ps` paranccsal az aktuális folyamatokról kapunk egy snapshot-t
  - `ps aux` - kiírja az összes felhasználó tty nélküli folyamatait is, felhasználók vannak az első oszlopban
- `pstree -t` - részletesen listázza a folyamatfát

Folyamatok lezárása

- Szükségünk van a folyamat PID-jére
  - `ps aux | grep process`
  - `pgrep process`
  - `pidof process`
- `kill PID`
- Ha nincs szükség ennyire precíz lezárásra, akkor
  - `pkill process`
  - `killall process`
- `xkill` paranccsal grafikusan választhatunk a lezárni kívánt ablakok közül

Szálkezelés

- `jobs` paranccsal listázhatjuk a leszármazott folyamatokat
- `wait` paranccsal várhatunk az összes leszármazott folyamatra, vagy a felsorolt PID-ekre

Következő gyakorlat

- Feladatok ütemezése
- Egyéb fontos parancsok

Érdekességek, olvasnivalók

- Selenium
  - Rövid script tárgyfelvételhez
- BeautifulSoup

Operációs rendszerek alapjai

8. gyakorlat

Mai gyakorlat

- Környezeti változók
- Feladatok ütemezése
- Egyéb fontos parancsok
- Feladatokhoz szükséges: G08.tar.gz

export

- Exportálhatunk egy változót, amit megkapnak a child process-k is
  - a child process szemszögéből egy környezeti változó lesz
    - tehát módosíthatja, viszont a parent process az eredeti értéket fogja használni
    - olyan ez, mint az érték szerinti paraméterátadás
- Létrehozásra is alkalmas, de már létező változót is exportálhatunk
  - `export var=5`
  - `var=5`
    `export var`
- Kapcsolók
  - `-f`: függvény exportálása
  - `-p`: exportált objektumok listázása

G08F01

- Próbáld ki az `export` kulcsszó működését!
  - hozz létre egy script-t, ami kiírja egy adott (egyelőre nem létező) változó értékét
  - futtasd le a script-t és figyeld meg, hogy üres sort ír ki akkor is, ha terminálban definiálod a változót
  - exportáld a változót és futtasd újra a script-t
  - változtasd meg a változó értékét a script-ben és figyeld meg, hogy a terminálban nem változik

alias

- Megadhatunk egy másik nevet adott parancsnak
- Hosszú, sok flag-t igénylő, gyakran használt parancsok esetén érdemes használni
- `alias ll="ls -l"`
- Személy szerint nem ajánlom a gyakori használatát, mert elfelejted az alias mögötti jelentést és ha esetleg nem pont úgy kell meghívni a parancsot, akkor újra meg kell nézned a elfedett parancsot és kapcsolókat
- Script-be nem adódik át, nem lehet exportálni sem
    - csökkentené a script hordozhatóságát

dotfiles

- Konfigurációs fájlok, beállításokat tartalmaznak egyes programokkal kapcsolatban
  - vim-hez .vimrc
  - bash-hez .bashrc
- Az elnevezés arra utal, hogy ezek gyakran rejtett fájlok
- Dotfile-ok használata növeli a rendszered hordozhatóságát
  - nekem például van egy script-em, ami feltölti az összes fontos dotfile-t a felhőbe, meg egy script, ami letölti ezeket
  - újratelepítése előtt feltöltés, utána letöltés, így gyorsan és fájdalommentesen frissítettem
- Ezen felül ezek a fájlok teljesen disztribúció függetlenek

.bashrc

- Általában szokott lenni egy előre létrehozott .bashrc sok hasznos beállítással
  - ezeket a beállításokat kezdőként nem érdemes megváltoztatni
- Lehet függvényeket is definiálni, amiket ezentúl tudsz használni a terminálban
- Lehet változókat és függvényeket exportálni
  - például $PATH változtatásait így tudod maradandóvá tenni
- Lehet létrehozni alias-okat
- És még sok más

exec

- Lefuttat egy programot subshell nélkül
  - egész pontosan a program helyettesíteni fogja az aktuális shell-t
- Subshell létrehozásának mellőzése természetesen kíméli az erőforrásokat
- Akkor is hasznos, hogyha nem kell visszaadni a vezérlést a parent process-nek
  - például script-ben, ha tudod, hogy már nem akarsz visszatérni a script-be

G08F02

- Tegyük fel, hogy bash az alapértelmezett shell egy rendszeren (például cortex), de inkább sh-t szeretnél használni
- Vizsgáld meg, hogy mi történik, ha egyszerűen lefuttatod az sh parancsot!
- Vizsgáld meg, hogy mi történik, ha exec-el futtatsz!

date

- Print or set the system date and time
- Script-ben szokás használni például log vagy backup készítéshez stb.
- Rengeteg formátum elérhető
- `date +FORMAT`

cron

- Ütemezésre alkalmas program
- Szabályok és futtatandó parancsok segítségével lehet irányítani a daemon-t
- `crontab` paranccsal nyithatjuk meg a feladatokat tartalmazó fájlt
  - `-l` kapcsolóval kiírhatjuk a tartalmát
  - `-e` kapcsolóval szerkeszthetjük
    - első alkalommal választani kell, hogy milyen szerkesztővel nyissa meg
  - `-r` kapcsolóval törölhetjük
  - `-u` kapcsolóval megadhatjuk, hogy melyik felhasználó feladataival foglalkozunk
    - ezt csak superuser teheti meg
  - lehet superuser szintű parancsokat is végrehajtani, ezeket a `sudo crontab` paranccsal érthetjük el
- Hibákról vagy logokról tud e-mailt küldeni a felhasználóhoz rendelt vagy a MAILTO változóban lévő címre (egyetemi szerverekről hallgatók nem küldhetnek e-mailt)

crontab

- Üres és #-el kezdődő sorokat nem veszi figyelembe
- Szintaxis:
  - öt paraméter vagy speciális trigger
    - perc (0-59)
    - óra (0-23)
    - nap (1-31)
    - hónap (1-12 vagy három betűs angol rövidítés)
    - hét napja (0-6 vagy három betűs angol rövidítés)
    - speciális triggerek például: `@reboot`, `@yearly`, `@monthly`, `@weekly`, `@daily`, `@hourly`
    - felsorolás vesszővel, intervallum kötőjellel, minden érték wildcard-al, lépésköz /-el
  - parancs
- Cron nem szokásos bash-t hív meg, sok funkció (pl command substituion) nem elérhető

crontab példák

- minden percben
  - `* * * * *`
- minden páros percben
  - `*/2 * * * *`
- minden páratlan percben
  - `1-59/2 * * * *`
- minden nap éjfélkor
  - `0 0 * * *`
- minden nap reggel és este nyolckor
  - `0 8,20 * * *`

crontab példák

- minden hétköznap éjfélkor
  - `0 0 * * 1-5`
- minden hétvégén éjfélkor
  - `0 0 * * 6,0`
- minden hónap első és tizenötödik napján és szerdánként éjfélkor
  - `0 0 1,15 * 3`
- minden hónap első napján 14:15-kor
  - `15 14 1 * *`
- minden héten hétfőtől péntekig 22:00-kor
  - `0 22 * * 1-5`

G08F03

- Írj egy cron job-t, ami minden percben megnyit egy böngészőt a kedvenc weboldaladdal!
  - `* * * * * export DISPLAY=:0 && /bin/firefox --new-window google.com`

G08F04

- Írj egy cron job-t, ami minden percben megnyit egy terminált és kiírja, hogy Hello World! A terminál maradjon nyitva öt másodpercig, majd zárjon be!
  - `* * * * * export DISPLAY=:0 && /bin/gnome-terminal -- /bin/sh -c 'echo Hello World!; sleep 5'`
- Módosítsd úgy a cron job-t, hogy ne zárjon be a terminál!
  - `* * * * * export DISPLAY=:0 && /bin/gnome-terminal -- /bin/sh -c 'echo Hello World!; sleep 5; exec bash'`

G08F05

- Írj egy Hello World script-t a dátum kiírásának kiegészítésével!
- Írj egy cron job-t, ami minden percben meghívja ezt a script-t és átirányítja egy fájlba!
  - `* * * * * ~/hello.sh >> file`

time

- Run programs and summarize system resource usage
- Kimenetben kiírja az eltelt valós időt, user időt és system időt
  - valós idő a program futásának elejétől a végéig eltelt idő
  - user idő az, amit a CPU user módban töltött, például tömb iterálása
  - system idő az, amit a CPU system módban töltött, például exec hívása
- Formátum átállítható

xargs

- Build and execute command lines from standard input
- Standard inputról (tehát pipe-ból is) olvas argumentumokat és végrehajt adott parancsot
- `inputs | xargs command`
  - inputokra végrehajta a `command` parancsot, a `command inputs` parancshoz hasonlóan
  - lehet szabályozni, hogy hány soronként hajtsa végre a parancsot
    - például ha létrehozni kívánt mappák neveit adom be, mindegy, hogy egyesével vagy együtt

```
mkdir dir1
mkdir dir2 == mkdir dir1 dir2 dir3
mkdir dir3
```

xargs

- Fontosabb kapcsolók
  - `-a filename`: adott fájlból olvassa az argumentumokat
  - `-L num`: num soronként olvassa be a bemenetet
  - `-I replace-str`: paraméterek létrehozása, az argumentumot ide helyettesíti be
    - ekkor mindenképpen soronként olvas be, tehát `-L 1` kapcsoló be lesz kapcsolva
  - `-p`: interaktív mód, minden parancs lefuttatása előtt megkérdezi, hogy futtassa-e vagy sem
  - `-t`: verbose mód, minden parancs lefuttatásakor kiírja az adott parancsot

G08F06

- Töltsd le és csomagold ki a G08.tar.gz fájlt!
  - vagy másold le a cortex-en a /home/vagmi mappából
- Hasonlítsd össze a find parancs beépített végrehajtó kapcsolójának és az xargs parancsnak a futásidejét a kicsomagold fájlokon!
  - például egy grep végrehajtásával
- Próbáld ki többféle verzióban, például beépített végrehajtó esetén konkatenálással és anélkül, illetve az xargs parancs esetén különböző maximális sorokkal!

HF02 - 15 pont

- One-vs-rest multiclass perceptron osztályozót kell írni awk-ban
- Cortex szerveren /home/vagmi mappában HF02.tar.gz fájlban
  - garantáltan lineárisan szeparábilis, konvergálni fog az osztályozó
  - 2D adat egyébként, de a kód ezt ne használja ki
- Cortex 9500-s portján fut egy API, HTTP status kódokkal vagy json-el válaszol
  - tehát cortex szerveren http://localhost:9500/endpoint címen érhető el, wget vagy curl
- Osztályozó tanítása után kérni kell legalább egy új adatot a /HF02/new_data endpoint-ról
  - POST kérés json formátumban, aminek van “NEPTUN” mezője érvényes Neptun kóddal
- Prediktált osztályt fel kell tölteni a /HF02/check_prediction endpoint-ra, ami ellenőrzi a választ
  - POST kérés json formátumban, “NEPTUN” mezővel, “data” mezővel és “prediction” mezővel
- Megoldást tempus szerverre a /home/vagmi/Desktop/HF02 mappába kell feltölteni
  - ha több fájl (pl bash és awk script külön), akkor tömörítve

Érdekességek, olvasnivalók

- crontab.guru
  - nagyon jó gyakorlás

Operációs rendszerek alapjai

9. gyakorlat

Mai gyakorlat

- Here document-k és string-k
- Néhány egyéb parancs
- Gyakorlás

Here document

- Speciális kód blokk, egy inline fájlt lehet definiálni vele
  - fájl be lehet olvasni, lehet parancsba átirányítani stb.
- Szintaxis

```
command <<nameOfHereDoc
...
nameOfHereDoc
```

- Kifejezetten alkalmas például többsoros üzenetek írására

```
wall <<EOF
Hello
World
EOF
```

Here string

- Gyakorlatilag leegyszerűsített here document
- Parancs standard inputjára irányít
  - lehet ciklus bemenetére is irányítani
- Szintaxis

  ```
  command <<< $string
  ```
- Kifejezetten alkalmas például tömbök beolvasására

  ```
  readarray array <<< $(command)
  ```

G09F01

- Írj egy scriptet, ami logolja a bejelentkezett felhasználókat!
  - felhasználónkénti összefoglaló szükséges, például minden felhasználóhoz tartozik egy fájl
- Először listázd az aktív felhasználókat!
- Szűrd ki a többször bejelentkezett felhasználókat!
- Adott felhasználó fájljába írd ki az aktuális időt!
- Próbáld minél kevesebb sorban megoldani!
  - irányadó: one-liner

Egyéb parancsok

- `nohup`
  - run a command immune to hangups, with output to a non-tty
  - használat például szerveren, ssh-val
    - nem elég háttérben futtatni egy parancsot, mert az lezárul az ssh kapcsolattal együtt
- `spd-say`
  - send text-to-speech output request to speech-dispatcher

G09F02

- Írj ki egy fájlt visszafelé!

G09F03

- Írj egy script-t, ami standard inputról olvasott római számot átalakít decimális számmá!

G09F04

- Írj egy script-t, ami kicsomagol egy tömörített fájlt a kiterjesztése alapján!

G09F05

- Írj egy script-t, ami visszaszámol megadott másodpercet!
  - saját gépen próbáld ki, hogy leálláskor kiad valamilyen hangot

Érdekességek, olvasnivalók

- Project Euler
- HackerRank
- Awesome Bash

Operációs rendszerek alapjai

10. gyakorlat

Mai gyakorlat

- Extended regular expression
- Gyakorlás

Extended regular expression (ERE)

- Az eddig ismert reguláris kifejezésekkel csak keresni tudtunk
- Szeretnénk
  - a reguláris kifejezésben használni a találatot
  - szerkesztéseket is végrehajtani a megtalált objektumokon
  - például Python főverziószámot váltunk

    `print "Hello World!" -> print("Hello World")`
- A megoldás az ERE használata
- Legtöbb program képes rá a -E kapcsolóval (sed, grep stb.)
- Legtöbb programozási nyelv tud (E)RE-ket kezelni
- Legtöbb (jó) szövegszerkesztő tud (E)RE-ket kezelni

Backreference

- A `()` operátor segítségével eddig mintázatok unióját vettük (grouping)
  - `(a|b)+` megtalálja a ‘a’ és ‘b’ karakterek kombinációjával létrehozott nem üres szavakat
- Az operátor ennél többet csinál, el is menti a találatot (capturing)
- Az n-edik csoport találatára \n-ként hivatkozhatunk (backreference)
  - backreference-re is vonatkoznak a módosítók
  - példa
    - találat regex: `^print ("(.*)")$`
    - helyettesítő regex: `print(\1)`
- Ezen a honlapon lehet nagyon jól gyakorolni

G10F01

- Próbáld ki a Python főverziószámos példát!
  - A honlapon van lehetőség helyettesítésre is

G10F02

- Tervezz egy reguláris kifejezést, ami megtalálja az összes HTML címsort!

G10F03

- A MATLAB nyelv már képes kezelni összetett értékadás operátorokat
  - igazából nem
- Tehát `bar=bar+foo` helyett `bar+=foo`
- Tervezz egy reguláris kifejezést, ami elvégzi a helyettesítést!

G10F04

- Írj egy script-t, ami run-length kódolja/dekódolja a bemenetet!
  - ismétlődő karakterek helyettesítése az ismétlés számával, illetve egy karakter példánnyal
    - AABBB <-> 2A3B
- Kódolás esetén a bemenet csak betűkből állhat!
  - ^[a-zA-Z]+$
- Dekódolás esetén a bemenet számmal kezdődik, betűvel végződik és minden két szám között van betű.
  - ^([0-9]+[a-zA-Z])+$
- Implementáció során alkalmazzuk a bash előnyeit!
  - ne karakterenként dolgozzunk, inkább használjunk extended regex-t
  - ha mégis karakterenként, akkor érdemes awk-ban bash helyett
  - irányadó fájlméret: egy-egy sor a kódolás és dekódolás

G10F05

- Becsüld meg π értékét Monte Carlo módszerrel!
- Lépések, például egy awk script-ben
  - generálj sok (mondjuk 10^7) random (x,y) párt
  - számold meg azokat az (x,y) párokat, amikre x^2+y^2<=1
  - π jó közelítéssel 4xcount/10^7
- Most párhuzamosítsd!
  - indítsd el mondjuk 100 szálat, amik csak 10^5 párt generálnak
    - `&` jellel háttérben indul el a folyamat, `wait` kulcsszóval megvárja a folyamatokat
  - átlagold az eredményüket
- Hasonlítsd össze a futásidőket!

Szorgalmi HF03 (15 pont)

- Írj párhuzamosított merge sort-t egész számokra!
  - bemeneti struktúra tetszőleges (csv, tsv stb.)
- Tetszőleges nyelv használható a gyakorlaton tanultak közül
- Javasolt struktúra
  - rendező awk script
  - összefűző awk script
    - feltételezheted, hogy két rendezett tömb a bemenete
  - bash script
    - meghívja a rendező awk script-t a tömb két felére
    - kimenetekre meghívja az összefűző awk script-t
- Hasonlítsd össze a futásidőt a soros verzióval!

Érdekességek, olvasnivalók

- Regex gyakorló honlap
- Uncle Bob: Clean Code lessons
- Uncle Bob: The Future of Programming