02

Szűrők, reguláris kifejezések
AWK programozás: minták, vezérlési szerkezetek, tömbök, beépített függvények, reguláris kifejezések

Egyszerű szöveges szűrő

- grep – csak a mintához illeszkedő sorokat írja fájlból
- kapcsolók:
  - n: kiírja az illeszkedő sor sorszámát
  - c: megszámlálja azokat a sorokat, ahol illeszkedést talált

Feladat:

- /etc/mime.types -ban keressük meg a video kiterjesztéseket, hány ilyen sor van?

Összetett szűrő – Reguláris kifejezések

- Komplex mintaillesztés megadása
- Szövegen belül ***bárhol*** elkezdődő és ezen belül a ***leghosszabb*** illeszkedést értjük.
- Részletesebben a formális nyelvek (vagy számtud alapjai) kurzus témája, most csak mint egy eszköz nézzük meg
- Bashben használta: egrep paranccsal

Elemi KIFejezések megadása

- KARAKTER: közönséges karakter egy példányára illeszkedik
- \KARAKTER: spec karakter is közönséges karakter lesz, annak példányára illeszkedik
- (): üres szóra illeszkedik (üres zárójelpár)
- . bármilyen egyszerű karakter egy példányára illeszkedik (spec karaktereket kivéve)
- [HALMAZ]: felsorolt karakterek bármelyikének egy példányára illeszkedik, [TÓL-IG] hasonlóan, csak az intervallumba eső bármelyik karakterrel, [^HALMAZ] a nem felsorolt karakterek bármelyikének egy példányára illeszkedik. Például [a-z],[A-Z],[0-9]
- ^ sor elejére illeszkedik, $ sor végére illeszkedik

Összetett kifejezés

- iterálható:
  - KIF*: a KIF akárhány egymást követő példányra illeszkedik, de az üres szóra is
  - KIF+: a KIF legalább egy egymást követő példányra illeszkedik
  - KIF?: 0 vagy 1 előfordulásra illeszkedik (0 előfordulás az üres szó)
  - KIF{i}: pontosan i egymást követő példányra illeszkedik
  - KIF{i,}: legalább i egymást követő példányra illeszkedik
  - KIF{i,j}: legalább i egymást követő példányra illeszkedik, de legfeljebb j egymás után követőre (i<=j teljülése mellett)
- KIF1KIF2: összefűzhető (konkatenáció)
- KIF1|KIF2: legalább az egyik kifejezésre illeszkedik (logikai vagy, alternáció)
- Műveletek erőssége: iteráció, konkatenáció, alternáció. De csoportosítható: ((KIF1)(KIF2))

Reguláris kifejezés - bash

- egrep 'REGKIF' ALLOMANY:
  - csak a reguláris kifejezéshez illeszkedő sorokat írja ki a fájlból
  - ha nem adunk meg ALLOMANYt akkor stdin-ről olvas
  - reguláris kifejezések megértéséhez javasolt a --color kapcsoló használata (kiszínezi az illeszkedést)

egrep fontosabb kapcsolók

-c illeszkedések darabszáma

-n az illeszkedő sor sorszámát is kiírja

-v nem illeszkedő sorok

-f KIFFÁJL az illesztő kifejezéseket szöveges állományból olvassa (fájl minden sorában pontosan egy reguláris kifejezés lehet, VAGY)

-q eltünteti a kimenetet, visszatérési érték az exit státusz (ezzel lehet if-fel használni) :

- igaz, ha talál illeszkedést,
- hamis, ha nem

Példák

- írjunk reguláris kifejezést, ami az IP(v4) címekre illeszkedik
- írjunk reguláris kifejezést, ami paraméterül kap egy könyvtárat, ahonnan ls -l listázva válasszuk ki a könyvtárakra vonatkozó sorokat
- írjunk regularis kifejezest, ami olyan sorokra illeszkedik ami az alábbi négy szó valamelyikét kizárólagosan tartalmazza: asztalon, asztalra, asztalhoz, asztalrol

cut

- sorok kiválasztott részeit írja ki. Kapcsolók:
  - -b megadott bájtok közötti részt írja ki
  - -c megadott karakterszám közötti részt írja ki
  - **-d elválasztó jel (alapértelmezett a TAB)**

    -f elválasztással született megadott mezők közötti részt írja ki
- Feladat:
  - cutoljuk az ls -l tartalmát (pl első karakter)
  - cutoljuk a ls -1 tartalmából az állományneveket, kiterjesztéseket

AWK

- adatvezérelt szkriptnyelv
- text processing, adat kiterjesztés, tagolt adatok automatizált soronkénti feldolgozása
- a forrásállományt soronként beolvassa és feldolgozhatóvá teszi
- a sor egyes elemeire külön alapértelmezett változóként lehet hivatkozni (az elválasztási módszert definiálni lehet), rengeteg beépített változó érhető el
- BASH-nél lazább szintaxis

AWK futtatási módok

- AWK parancs, közvetlen programkódmegadás:
  - awk 'PROGRAMKÓD' FILE
  - példa: ls -l | awk '{print $1, $5}'
  - a programkód helyére minden indentálás nélkül beírod a programkódot → ez rettentően nehezen olvashatóvá teszi az egészet, csak a legegyszerűbb feladatok esetén javasolt !!!
- AWK programkód külső fájlból beolvasva:
  - awk -f CODEFILE FILE
  - ez a legkézenfekvőbb
- AWK program, mint futtatható AWK script:
  - parancsértelmező fejrész megadása:
    - #! /usr/bin/awk -f

      (ha nem ismeri akkor locate bin/awk)
  - ./AWK_SCRIPT FILE

Feladat: HelloWorld AWK futtatása ls -l kimenetén

AWK delimiter

- Az awk soronkénti műveletvégzést tesz lehetővé, a sorokat változókba tagolja egy *delimiter* által:

  $1 $2 $3 $4 $5 …... $NF

  (NF változó a sor mezőinek számát tárolja)

- Az awk alapértelmezés szerint a whitespace az elválasztás
- Megadható:
  - kapcsolóval: `awk -F";" CODEFILE FILE`
  - FS változónak értékül adva: `FS = ";"`
- Delimiter lehet bármilyen karakter (akár escape) vagy **reguláris kifejezés**

Feladat

- Írjunk awk scriptet, ami a delim.dat fájl különböző elválasztó jelek mentén választja el. Próbáljuk ki a különböző lehetőségeket.

Kifejezések

- +, –, *, /, %, ^
- ++, - -
- egymás mellé írás (string konkatenáció)
- mezőhivatkozás ($KIF)
- =, +=, -=,*=, /=, %=, ^=
- <, <=, >, >=, ==, != (stringek esetén lexikografikus)
- valtozo ~ / REGKIF /, valtozo !~ / REGKIF /
- INDEX in TÖMB (tömbindex létezése, true or false)
- !, &&, ||
- KIF1 ? KIF2 : KIF3

Egyszerű változók

- jelölése kisbetűkkel
- értékadás, hivatkozás C-hez hasonlóan
- dinamikusak: első használatkor jönnek létre
- típusuk az alapján állítódik be, amit értékül adunk:
  - numerikus
  - szöveges
  - tömb
- A szám szöveggé, a szöveg számmá automatikusan konvertálható (ha nem sikerül akkor 0)
  - de ez manuálisan is elvégezhető, kikényszeríthető:
    - str+0 (szöveg számmá)
    - num”” (szám szöveggé)

Változó értékek

- Szám:
  - egész: 12
  - valós, tizedesponttal: 12.0
  - valós, lebegőpont: 1.2e+1
- Szöveg:
  - ”SZOVEG\n” (escape karakterek használhatóak)
  - üres string: ””

Beépített változók

- $1 $2 $3 (módosíthatók, $-el ellátott változóval is hivatkozhatunk ezekre: $i)
- $0 – az egész sor egyben
- NF – number of fields
- NR – aktuális sor azonosítója
- FNR – több fájl esetén fájlonkénti aktuális sorának azonosítója
- FILENAME – bemeneti fájl neve (amit feldolgozunk)
- FS – delimiter (alapértelmezés szerint szóköz)
- OFS – kimeneti delimiter (alapértelmezés szerint szóköz)
- RS – sorhatároló karakter (alapértelmezetten \n)
- ORS – kimeneti sor delimiter (alapértelmezetten \n)
- IGNORECASE – ha nem nulla, akkor azonosak a kis és nagybetűk (alapértelmezés szerint mindig nulla)

  Feladat: próbáljuk ki a beépített változók viselkedését

egy AWK program felépítése

- Szabályokból épül fel:
  - MINTA{ AKCIÓ1; … AKCIÓ2; }
    - ha elhagyjuk a MINTA-t, akkor minden sorra végrehajtja
    - ha elhagyjuk a { AKCIÓ }-t akkor a MINTA teljesülése esetén {print $0} fog végrehajtódni (ha nem akarunk semmit, akkor {} -et kell hagynunk a MINTA után)
- Egyes parancsok végét ; -el jelöljük (soronként tagolva ez elhagyható)
- Nem érzékeny a szóköz, TAB tagolásra: akár az egész programkódot írhatjuk egyetlen sorba, de szét is tagolhatjuk valamilyen kódformázási protokoll alapján
- Kommentelés: #

Minták

- Soronkénti feltétel, ha teljesül, akkor végrehajtódik a blokk, különben nem.
- A változók felhasználhatóak a minta megadásakor
- Elemi minták:
  - MINTA: relációs kifejezés
  - (MINTA)
  - ! MINTA
  - / REGKIF / ha a reguláris kifejezés illeszkedik a sorban
  - BEGIN : awk program indulásakor elsőként fut le
  - END : awk program végén a kilépés előtt utoljára lefutó rész
    - Nem kombinálhatóak más mintákkal
- Összetett minták: &&, ||

Minták

- Soronként ellenőrzi és megadási sorrend alapján:
  - BEGIN{ AKCIÓ1; … AKCIÓ2; }
  - MINTA1{ AKCIÓ1; … AKCIÓ2; }

    .

    .

    .
  - MINTAn{ AKCIÓ1; … AKCIÓ2; }
  - END{ AKCIÓ1; … AKCIÓ2; }
- Feladat:

1) Köszöntő üzenet, számoljuk össze a sorokat és csak a végén írjuk ki.

2) Írjuk ki az első sor kivételével az összes sort

3) Az ls -l kimenetéből szűrjük ki a könyvtárakat és csak a nevüket írjuk ki

Vezérlési szerkezetek

- for (KIF1; KIF2; KIF3;) {}
- for (INDEX in NÉV) {}
  - nem sorrendben járja be a tömböt!
  - csak ha nem számít a sorrend, pl: halmaz
- break, continue
- exit
  - (ha nem az END-ben van, akkor az END szekció még lefut!!)

Vezérlési szerkezetek

- if (FELTÉTEL) {
  UTASÍTÁS
  } else if (FELTÉTEL) {
  UTASÍTÁS
  } else {
  UTASÍTÁS
  }
- while (FELTÉTEL) {}
- do {} while (FELTÉTEL)

Feladatok

1) Írjuk ki az összes szót a bocsanat.txt -ből
2) Írjuk ki az 10 karakternél hosszabb szavakat a bocsanat.txt -ből

Tömb - Asszociatív tömb

- Dinamikus indexelés, szöveges indexekkel:
  - nem kell előre megadni a méretet
  - tetszőleges szöveges indexszel használható
  - értékek típusában sem kell egyeznie (tömbön belül lehet szöveg és szám is)
  - ha olyan indexre hivatkozunk, ami nem létezik, akkor 0 vagy ”” lesz az eredmény
- NÉV[INDEX] = ÉRTÉK
- delete NÉV[INDEX] vagy delete NÉV

Tömb - feladat

- Számoljuk ki bocsanat.txt-ben lévő szavak előfordulását, majd írjuk ki
  - figyeljünk oda arra, hogy a különböző írásjelek miatt egy szó ne szerepeljen többször → reguláris kifejezés segítségével adjuk meg az elválasztó jeleket
  - üres szó előfordulása nem érdekel → töröljük a tömbünkből

Beépített szöveges függvények

- getline valtozo; – beolvas a STDIN-ről
- tolower(str), toupper(str) – kisbetűs/nagybetűs változata az str-nek
- length(str) – szöveg hossza
- index(str, substr) – a substr-et keresi a str-ben és visszatér a pozíciójával
- split(str, array, delim) – a delim mentén feldarabolja str-t és az array tömbbe tárolja el a darabokat
- substr(str, start_index) – a str start_index pozíciójától kezdődő részt adja vissza
- substr(str, start_index, len) – a str start_index pozíciójától kezdődve len hosszú részt ad vissza

Feladatok

- Egészítsük ki az előző feladatot az alábbiakkal: hagyja figyelmen kívül a kis/nagy betűket
- A split.txt-ből határozzuk meg azokat a sorokat, amelyeknél 2014. 10. 21-nél újabbak

Beépített numerikus függvények

- int(x) - egészre vágás (simán elhagyjuk a tizedes jegyeket)
- sqrt(x) - négyzetgyök
- exp(x) - (e ^ x)
- log(x) - természetes alapú logaritmus
- sin(x), cos(x) - sin, cos radiánon értelmezve
- rand() - 0 és 1 közötti véletlen számot ad

  Az intervallum [0,1). Tehát ha pl.: indexet akarunk vele dobni, akkor hozzá kell adnunk 1-et!