02

Szűrők
Reguláris kifejezések,
AWK

A félév hátralévő részének célja

Szöveges fájlok tartalmának a kezelése, manipulációja, automatizált módosítása

Reguláris kifejezések

- Komplex mintaillesztés megadása
- Szövegen belül ***bárhol***, ***legkorábban*** elkezdődő és ezen belül a ***leghosszabb*** illeszkedést értjük.
- Részletesebben a formális nyelvek (vagy számtud alapjai) kurzus témája, most csak mint egy eszköz nézzük meg
- Fontos: bár a cél azonos, de a Linuxos jelölése **teljesen más** mint a mintaillesztésnél tanultaknak!

Elemi KIFejezések megadása

- KARAKTER: közönséges karakter egy példányára illeszkedik
- \KARAKTER: spec karakter is közönséges karakter lesz, annak példányára illeszkedik
- (): üres szóra illeszkedik
- . bármilyen 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
- ^ sor elejére illeszkedik, $ sor végére illeszkedik

Összetett kifejezés

(1) 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)

(2) KIF1KIF2: összefűzhető (konkatenáció)

(3) KIF1|KIF2: legalább az egyik kifejezésre illeszkedik (logikai vagy, alternáció)

- Műveletek erőssége: iteráció, konkatenáció, alternáció. De csoportosítható: ((KIF1)(KIF2))

Reguláris kifejezés mint szűrő

- egrep REGKIF ALLOMANY:
  - csak a reguláris kifejezéshez illeszkedő sorokat írja ki a fájlból
  - ez egy szűrő
  - ha nem adunk meg ALLOMANYt akkor stdin-ről olvas
  - javasolt a reguláris kifejezések megadására a ' ' idézőjelek használata

egrep fontosabb kapcsolók

- -c illeszkedések darabszáma
- -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 igaz, ha talál illeszkedést, hamis ha nem (ezzel lehet if-fel használni)

Feladat: írjuk ki azoknak a sorokat, amikben szerepel a „nem” szó. Számoljuk meg. Kis/nagy betű!

Példák

- írjunk reguláris kifejezést, ami az IP(v4) címekre illeszkedik
- írjunk scriptet, ami egy e-mail címet fogad és megmondja, hogy helyes szerkezetű-e
- írjunk scriptet, ami paraméterül kap egy könyvtárat, ahonnan ls -l listázva válasszuk ki a könyvtárakra vonatkozó sorokat. a megoldáshoz használjunk függvényt

Megoldás: regkif.txt

További szűrők (otthon próbáld ki)

- grep: az egrep-pel azonos funkciók, viszont fordított jelölésű reguláris kifejezések: a reguláris kifejezések speciális karaktereit kell \-sel jelölni
- cut: sorok vágása megadott delimiter mentén
- sed: többfunkciós szűrő
  - reguláris kifejezések illesztése
  - keresés + csere
  - keresés + törlés

AWK

AWK

- Mintakereső és -feldolgozó program saját programozási nyelvvel
- 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

AWK felépítés - bevezetés

- 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)
- Nem érzékeny a szóköz, TAB tagolásra: akár az egész programkódot írhatjuk egyetlen sorba, de szét is tagolhatjuk valamilyen kódformázási protokoll alapján. Nagyon ritka a sintaxerror.
- 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
  - SZOVEG ~ / REGKIF / ha a reguláris kifejezés illeszkedik a szövegre
  - SZOVEG !~ / REGKIF /
  - 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; }

Példa: HelloWorld AWK

Kifejezések

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

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: hozzunk létre egy fájlt, amiben soronként más elválasztó jel van, adjunk meg rá reguláris kifejezést, ami az awk számára feldolgozhatóvá teszi.

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:
    - str+0 (szöveg számmá)
    - num”” (szám szöveggé)

Feladat: próbáljuk ki a különböző típusokat és a konverziót.

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

Vezérlési szerkezetek

- for (KIF1; KIF2; KIF3;) {}
- for (INDEX in NÉV) {}
  - nem javaslom, mert 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 lefut!!)

Vezérlési szerkezetek

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

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 meg a sorokat, majd azokat, amelyek nem üres sorok.
- Számoljuk ki 10_feladat.txt-ben lévő szavak előfordulását, majd írjuk ki
  - figyeljünk oda arra, hogy a különböző írásjelek miatt egy szó ne szerepeljen többször → szűrjük le az írásjeleket (delimiter segítségével)
  - üres szó előfordulása nem érdekel → töröljük a tömbünkből

Beépített szöveges függvények

- getline valtozo; – beolvas a STDIN-ről
- tolower(str), toupper() – kisbetűs/nagybetűs változata az str-nek
- legth(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
- Írjuk ki azokat a sorokat amelyekben megtalálható a „nem” szó
  - reguláris kifejezés használata nélkül Ötlet: index
  - reguláris kifejezéssel
- Az előbbi sorokat csak a „nem” szótól kezdve írjuk ki.
- Töltsünk be minden sort egy tömbbe, ahol minden karakter külön tömb-elemen helyezkedik el. Írjuk ki a „nem” szót, ezzel a karaktertömb segítségével!
- Előző feladat egyszerűen substr segítségével

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

Feladat

- Minden sorban kérjünk egy véletlen számot, amivel meghatározunk a sorban egy pozíciót:
  - a véletlen szám a sor hossza és 0 közötti legyen
  - kerekítsük egészre, hogy indexként tudjuk használni
- Írjuk ki a pozíción lévő karaktert