Ismerjük meg a szabályos kifejezéseket!

A szövegfeldolgozás és szövegleírás terén aligha találunk a szabályos kifejezéseknél hatékonyabb eszközt.

Képzeljük el, hogy ki szeretnénk keresni egy nevet a telefonkönyvből, de nem emlékszünk a név pontos írásmódjára. Hosszú időt tölthetünk el az összes lehetséges név kikeresésével, hacsak nincs egy olyan eszközünk, amelyik kigyűjti azt a viszonylag kevés lehetőséget, ami a hiányos tudásunknak megfelelő keresési feltételre illeszkedik. A szabályos kifejezések pontosan ilyen eszközök.

A szabályos kifejezés egy olyan karakterlánc, ami egy másik karakterláncot vagy karakterlánccsoportot ír le. Számos alkalmazás aknázza ki ezt a lehetőséget, néhány ezek közül: Perl, sed, awk, egrep, sőt még az Emacs is (a cikk elolvasása után próbáljuk ki a Ctrl-Alt-% billentyűkombinációt).

Bizonyosfajta szabályos kifejezéseket valószínűleg mindannyian használtunk már. Az `ls *.pl` héjparancsban a ***.pl*** egy olyan szabályos kifejezés, ami a következő karakterláncokat írja le: tetszőleges számú tetszőleges karakter (*), utána egy pont (.), végül két adott karakter (pl).

A szabályos kifejezések előállításának szabályai az összes elképzelhető karakterlánc leírását lehetővé teszik, attól függetlenül, hogy azok milyen összetettek. Sajnos a valóságban a helyzet ennél bonyolultabb, ugyanis a szabályos kifejezéseknek legalább kétféle változata létezik: kiterjesztett és egyszerű. Ráadásul nem minden alkalmazás támogatja az összes szabályt.

Bevezetés a szabályos kifejezések használatába

Akkor mondjuk, hogy egy szabályos kifejezés illeszkedik egy karakterláncra, ha azt helyesen írja le. Egy adott szabályos kifejezés akárhány karakterláncra illeszkedhet. Az a szokás, hogy a szabályos kifejezéseket törtjelek közé =(/.../) írjuk. A következőkben a kiterjesztett szabályos kifejezésekkel fogunk dolgozni.

A legegyszerűbb szabályos kifejezés csak betűkből és számokból áll, az ilyen kifejezés az olyan összes karakterláncra illeszkedik, ami tartalmazza őt részkarakterláncként. Vegyük például a következő részletet a kedvenc Rossini-operámból: „Zitto, zitto, piano, piano, senza strepito e rumore.” A `/piano/` szabályos kifejezés illeszkedik a szövegre, mert ugyanazokat a karaktereket ugyanabban a sorrendben tartalmazza, mint amit a szabályos kifejezésben megadtunk.

A jobb megértés kedvéért játszhatunk a következő Perl-parancsfájllal. Változtassuk meg néhányszor a szabályos kifejezést:

```
#!/usr/bin/perl
$verse = "Zitto, zitto, piano, piano, senza "
.          "strepito e rumore";
if ($verse =~ /piano/) {
   print "Illeszkedik!\n";
} else {
   print "Nem illeszkedik!\n";
}
```

A Perl nyelvben az =~ műveleti jel két szabályos kifejezést hasonlít össze, és ha illeszkedést talál, „igaz” értéket ad vissza. Néhány karakter (a nevük metakarakter) nem egyszerű karakternek számít, hanem különleges célokra van fenntartva. Például a * (csillag) arra használatos, hogy egy karaktercsoport nulla vagy több előfordulására illeszkedjen. A karaktercsoportot, más néven atomot olyan módon adjuk meg, hogy az egyetlen egységnek kezelt karaktereket zárójelek közé zárjuk. A `/(piano,)*` szabályos kifejezés illeszkedik a példaszövegre, mert az atomot alkotó „piano,” karaktersorozat kétszer ismétlődik. Ha az atom csak egyetlen karakterből áll, akkor a zárójel elhagyható.

A * karakter jelentése a szabályos kifejezésekben eltér a héjbeli jelentésétől. A szabályos kifejezésekben a * módosító; a bal oldalán elhelyezkedő atom többszöri előfordulását írja le. Emiatt a „piano” karakterláncra illeszkedik a `p*` a héjban, de a `/p*/` szabályos kifejezés csak a p, pp, ppp-re, illetve az üres karakterláncra illeszkedik.

Az atom N és M közötti számú előfordulásának megadásához a {N,M} jelet használhatjuk. A {N} olyan karakterláncokra illeszkedik, amelyek az atom pontosan N darab ismétlődését tartalmazzák. A {N,} a legalább N darab ismétlődésre illeszkedik. A következő szabályos kifejezések illeszkednek:

```
/( piano,){0,10}/
/( piano,){1,2}/
/( piano,){2}/
```

Az első szabályos kifejezés természetesen illeszkedne a „piano, piano, piano” karakterláncra is. A + és a ? metakarakterek a {1,} és a {0,1} rövidítései.

Az illeszkedő zárójeles atomok különleges változókban tárolódnak, amiket a \ (per)jelet követő szám azonosít. A szabályos kifejezésben előforduló első zárójeles atom a `\1` változóban, a második a `\2` változóban tárolódik. Például a

```
/Z(itto), z\1, ( piano,)\2/
```

illeszkedik a fent megadott idézetre (`\1 = "itto"* Øs \2 **= "piano,"`).

A . (pont) metakarakter bármelyik karakterre illeszkedik, emiatt a `/.(itto),.\1/` szabályos kifejezés illeszkedik a „Zitto, zitto” és a „zitto, zitto” karakterláncra is. Ezenkívül illeszkedik még például a „Ritto, ritto” karakterláncra is, ami már nem ugyanazt jelenti. Ha a túlzott általánosítást el szeretnénk kerülni, a lehetséges karaktereket szögletes zárójelek között megadhatjuk `/[Zz](itto), [Zz]\1/`.

A kötőjel a szögletes zárójelek között karaktertartományt jelöl. Például a `/[a-z]/` az összes kisbetűre, a `/[A-Z]/` pedig az összes nagybetűre illeszkedik. A `/[a-zA-Z0-9_]/` minden betű, szám és aláhúzásjel karakterből felépülő karakterláncra illeszkedik.

A | (cső) metakarakter a különböző lehetőségek megadására szolgál. Működése hasonló a logikai VAGY művelethez. Emiatt a `/Zitto|zitto/` illeszkedik a „Zitto” és a „zitto” karakterláncra is.

A ^ és a $ (dollárjel) metakarakterek a karakterlánc elejére és végére illeszkednek. A szögletes zárójelen belül a ^ jel a tagadás művelete. Emiatt a `/[^a-z]itto/` illeszkedik a „Zitto” karakterláncra, de a „zitto” karakterláncra nem, mert a `[^a-z]` jelentése: „tetszőleges betű, ami nem kisbetű”. Ha a metakaraktert közönséges karakterként szeretnénk értelmeztetni, tegyünk elé egy fordított perjelet (\). Ekkor a szabályos kifejezést értelmező program tudni fogja, hogy ezt egyszerű karakterként kell értelmeznie.

A szabályos kifejezések használata

Értékelni fogjuk a szabályos kifejezés erejét, ha megvizsgáljuk a következő kis Perl-parancsfájlt, ami segít a rendszergazdáknak kiszűrni a sikertelen bejelentkezési kísérleteket.
A következő példákban szemléletes szabályos kifejezéseket fogok használni a különböző lehetőségek bemutatása céljából. Ugyanezeket a feladatokat egyszerűbb szabályos kifejezésekkel is megvalósíthatjuk.
Ha valakinek sikertelen a bejelentkezési kísérlete, a `syslogd` az alábbihoz hasonló üzenetet ír a */var/log/messaged* naplóállományba:

```
Jul 26 16:35:25 myhost su(pam_unix)[2549]:
➥authentication failure; logname=verdi uid=500
➥euid=0 tty= ruser=organtin rhost=  user=root
Jul 27 14:54:36 myhost login(pam_unix)[688]:
➥authentication failure; logname=LOGIN uid=0
➥euid=0 tty=tty1 ruser= rhost=  user=mozart
```

A bejegyzésből kiolvasható a bejelentkezési kísérlet időpontja, a felhasználó neve, aki egy másik felhasználó nevében be akart lépni – amennyiben ez az adat rendelkezésre áll –, és a célfelhasználó. Például a verdi felhasználó kétszer be akart lépni a rendszergazdaként, míg valaki mozart felhasználónévvel szeretett volna belépni a konzolról.
Vessünk egy pillantást az 1. listán (48. CD Magazin/Szabalyos könyvtár) bemutatott Perl-parancsfájlra: az a feladata, hogy beolvassa a */var/log/messages* állományt, azonosítsa az érdekesnek tűnő sorokat, és csak a lényeges adatokat gyűjtse ki.
Először is egyedül azokat a lényegi sorokat választjuk ki, amelyek illeszkednek az `/authentication failure/` szabályos kifejezésre (7. sor), minden egyebet eldobunk. Ezután minden sort egy olyan szabályos kifejezéssel kezelünk (8. sor), ami a következő karakterláncra illeszkedik: a karakterlánc eleje (^) pontosan három ({3}) betűből áll (`[a-zA-Z]`), ezt egy szóköz követi, majd legfeljebb két (+) karakter kerül sorra, amit számok (0–9, amit a Perlben a `\d` metakarakterrel is jelölhetnénk) vagy egy szóköz követ. A szóköz után tetszőleges számú (*) számjegy és pontosvessző lehet. A karakterlánc eddig leírt része zárójelben van, tehát a `\1` változóba kerül. Ezután tetszőleges számú karakter (.*) következhet a `"logname="` karakterlánc előtt. Ezt a karakterláncot tetszőleges számú betű vagy számjegy követheti. A zárójelpár következtében ez a `\2` változóban tárolódik. Végül tetszőleges számú karakter lehet a `"user="` karakterlánc előtt, amit tetszőleges számú betű és számjegy követhet. Ez az egész a `\3` változóba kerül.
Ebből a példából megtanulhatjuk, hogyan lehet részkarakterláncokat kivenni a karakterláncokból. Nem kell ismernünk a részkarakterláncok viszonylagos helyzetét, ha le tudjuk írni a kinézetüket.
A Perl hasznos szolgáltatása, hogy a szabályos kifejezések változóiból önműködően Perl-változókat hoz létre, amik a `$1`, `$2` stb. neveken érhetők el, miután a szabályos kifejezés illesztése megtörtént. A Perl hasznos jelek használatát is megengedi az ugyanazt jelentő Posix-megfelelő jelek mellett, ilyen például a már említett `\d` és a `\w`, ami megfelel a `[A-Za-z0-9_]` kifejezésnek. További tudnivalók a Perlről a súgóoldalon (man perlre) olvashatók.

Egyszerű szabályos kifejezések

Az egyszerű szabályos kifejezéseket számos más program is használja, például a sed vagy az egrep.
Az egyszerű szabályos kifejezésekben a |, a + és ? metakarakterek nem léteznek, és a gömbölyű, illetve kapcsos zárójeleket védeni kell, ha metakarakterként szeretnénk használni őket. A ^, a $ és a * metakarakterekre bonyolultabb szabály vonatkozik (további részletekért lásd: man 7 regex). A legtöbb esetben azonban kiterjesztett társaikhoz hasonlóan viselkednek. Gyakran a szabályos kifejezést a kiterjesztett alakban kényelmes szerkeszteni, és szükség esetén adni hozzá a védőkaraktereket.
A 2. listában (48. CD Magazin/Szabalyos könyvtár) bemutatott példa böngészőprogramban megtekinthető, html formátumú oldalt hoz létre a rendszernapló állományából. A HTML-elemek kiírása mellett, amelyek az oldal fejlécét és a táblázatot hozzák létre, kilistázzuk egy adott könyvtár összes állományát, és az eredményt a sed programba vezetjük, ami azt szabályos kifejezések segítségével átalakítja. A sed által is használt szöveghelyettesítés eléggé elterjedt:

```
s/kifejezØs/csere/
```

Ebben a kifejezés szabályos kifejezés, amit le kell cserélni.
A megadott szabályos kifejezés egy kilenc elemből álló karakterláncot ír le. Például a `[rwxds-]` az első elem lehetséges karaktereit adja meg. A karakterlánc ez utáni része betűket és számjegyeket tartalmaz, amiket perjelek tagolnak. Vegyük észre, hogy ebben az esetben a `(.*\/)(.*)` szabályos kifejezést használtuk. Az első csoport illeszkedik a perjelet megelőző összes karakterre, vagyis az elérési útra. A második csoport illeszkedik a többi karakterre (azaz a fájlnévre). Az elérési útban tetszőleges számú perjel lehet. A szabályos kifejezések (az egyszerűek és a kiterjesztettek is) mohók, azaz a lehető legtöbb karakterre próbálnak illeszkedni.
A parancsfájl eredménye a szabályos kimenetre íródik, amit egy adott állományba átirányíthatunk (például a `cron`-ból meghatározott időközönként), hogy megjelenjen a Weben.

Összegzés

A szabályos kifejezések messze a leghatékonyabb eszközök a szövegfeldolgozási és szövegleírási feladatokra, és Linux-környezetben számos alkalmazás támogatja őket. Sajnos a legnépszerűbb keresőmotorok (tudtommal) egyáltalán nem támogatják a szabályos kifejezéseket, éppen a bonyolultságuk miatt. Képzeljük csak el, milyen pontos lehetne a keresésünk, ha a keresett weboldalt szabályos kifejezéssel írnánk le!

Linux Journal 2003. május, 109. szám

Giovanni Organtini (g.organtini@roma1.infn.it)
A római egyetemen tartja a „Bevezetés a számítástechnikába és a programozásba fizikusok számára” című óráit. Évek óta használ Linuxot, szórakozáshoz és munkához egyaránt.